

Handbook of Combinatorial Optimization

Supplement Volume A

Handbook of Combinatorial Optimization

Supplement Volume A

Edited by

Ding-Zhu Du

*Department of Computer Science,
University of Minnesota, U.S.A.
and Institute of Applied Mathematics,
Academia Sinica, P. R. China*

and

Panos M. Pardalos

*Department of Industrial and Systems Engineering,
University of Florida, U.S.A.*



KLUWER ACADEMIC PUBLISHERS
DORDRECHT / BOSTON / LONDON

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN 0-7923-5924-0

Published by Kluwer Academic Publishers,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

Sold and distributed in North, Central and South America
by Kluwer Academic Publishers,
101 Philip Drive, Norwell, MA 02061, U.S.A.

In all other countries, sold and distributed
by Kluwer Academic Publishers,
P.O. Box 322, 3300 AH Dordrecht, The Netherlands.

Printed on acid-free paper

All Rights Reserved

© 1999 Kluwer Academic Publishers

No part of the material protected by this copyright notice may be reproduced or
utilized in any form or by any means, electronic or mechanical,
including photocopying, recording or by any information storage and
retrieval system, without written permission from the copyright owner.

Printed in the Netherlands.

Contents

Preface	vii
The Maximum Clique Problem	1
<i>Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo</i>	
Linear Assignment Problems and Extensions	75
<i>Rainer E. Burkard and Eranda Çela</i>	
Bin Packing Approximation Algorithms: Combinatorial Analysis	151
<i>Edward G. Coffman, Gabor Galambos, Silvano Martello, and Daniele Vigo</i>	
Feedback Set Problems	209
<i>Paola Festa, Panos M. Pardalos, and Mauricio G.C. Resende</i>	
Neural Networks Approaches for Combinatorial Optimization Problems	259
<i>Theodore B. Trafalis and Suat Kasap</i>	
Frequency Assignment Problems	295
<i>Robert A. Murphey, Panos M. Pardalos, Mauricio G. C. Resende</i>	
Algorithms for the Satisfiability (SAT) Problem	379
<i>Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah</i>	
The Steiner Ratio of L_p-planes	573
<i>Jens Albrecht and Dietmar Cieslik</i>	
A Cogitative Algorithm for Solving the Equal Circles Packing Problem	591
<i>Wenqi Huang, Yu-Liang Wu, and C. K. Wong</i>	
Author Index	607

Subject Index	633
----------------------------	------------

Preface

Combinatorial (or discrete) optimization is one of the most active fields in the interface of operations research, computer science, and applied mathematics. Combinatorial optimization problems arise in various applications, including communications network design, VLSI design, machine vision, airline crew scheduling, corporate planning, computer-aided design and manufacturing, database query design, cellular telephone frequency assignment, constraint directed reasoning, and computational biology. Furthermore, combinatorial optimization problems occur in many diverse areas such as linear and integer programming, graph theory, artificial intelligence, and number theory. All these problems, when formulated mathematically as the minimization or maximization of a certain function defined on some domain, have a commonality of discreteness.

Historically, combinatorial optimization starts with linear programming. Linear programming has an entire range of important applications including production planning and distribution, personnel assignment, finance, allocation of economic resources, circuit simulation, and control systems. Leonid Kantorovich and Tjalling Koopmans received the Nobel Prize (1975) for their work on the optimal allocation of resources. Two important discoveries, the ellipsoid method (1979) and interior point approaches (1984) both provide polynomial time algorithms for linear programming. These algorithms have had a profound effect in combinatorial optimization. Many polynomial-time solvable combinatorial optimization problems are special cases of linear programming (e.g. matching and maximum flow). In addition, linear programming relaxations are often the basis for many approximation algorithms for solving NP-hard problems (e.g. dual heuristics).

Two other developments with a great effect on combinatorial optimization are the design of efficient integer programming software and the availability of parallel computers. In the last decade, the use of integer programming models has changed and increased dramatically. Two decades ago, only problems with up to 100 integer variables could be solved in a computer. Today we can solve problems to optimality with thousands of integer variables. Furthermore, we can compute provably good approximate solutions to problems with millions of integer variables. These advances have been made possible by developments in hardware, software and algorithm design.

The Handbook of Combinatorial Optimization deals with several algorithmic approaches for discrete problems as well as with many combinatorial problems. We have tried to bring together almost every aspect of this enormous field with emphasis on recent developments. Each chapter in the Handbook is essentially expository in nature, but of scholarly treatment.

The Handbook of Combinatorial Optimization is addressed not only to researchers in discrete optimization, but to all scientists in various disciplines who use combinatorial optimization methods to solve problems. We are certain that experts in the field as well as nonspecialist readers will find the material of the Handbook stimulating and helpful.

We would like to take this opportunity to thank the authors, the anonymous referees, and the publisher for helping us produce these volumes of the Handbook of Combinatorial Optimization with state-of-the-art chapters. We would also like to thank Ms. Xiuzhen Cheng for making the Author Index and the Subject Index for this volume.

Ding-Zhu Du and Panos M. Pardalos

The Maximum Clique Problem

Immanuel M. Bomze

Institut für Statistik, Operations Research und Computerverfahren

Universität Wien

Universitätsstraße 5, A-1010 Wien, Austria

E-mail: immanuel.bomze@univie.ac.at

Marco Budinich

Dipartimento di Fisica

Università di Trieste

Via Valerio 2, I-34127 Trieste, Italy

E-mail: mbh@trieste.infn.it

Panos M. Pardalos

Center for Applied Optimization, ISE Department

University of Florida, Gainesville, FL 32611

E-mail: pardalos@ufl.edu

Marcello Pelillo

Dipartimento di Informatica

Università Ca' Foscari di Venezia

Via Torino 155, 30172 Venezia Mestre, Italy

E-mail: pelillo@dsi.unive.it

Contents

1	Introduction	2
1.1	Notations and Definitions	3
2	Problem Formulations	4
2.1	Integer Programming Formulations	5
2.2	Continuous Formulations	8

3 Computational Complexity	12
4 Bounds and Estimates	15
5 Exact Algorithms	19
5.1 Enumerative Algorithms	19
5.2 Exact Algorithms for the Unweighted Case	21
5.3 Exact Algorithms for the Weighted Case	25
6 Heuristics	27
6.1 Sequential Greedy Heuristics	28
6.2 Local Search Heuristics	29
6.3 Advanced Search Heuristics	30
6.3.1 Simulated annealing	30
6.3.2 Neural networks	32
6.3.3 Genetic algorithms	34
6.3.4 Tabu search	36
6.4 Continuous-based Heuristics	37
6.5 Miscellaneous	40
7 Selected Applications	41
7.1 Coding Theory: Hamming and Johnson Graphs	41
7.2 Geometry of Tiling: Keller's Conjecture	43
7.3 Problems Arising From Fault Diagnosis	43
7.4 Computer Vision and Pattern Recognition	45
8 Conclusions	47
References	

1 Introduction

The maximum clique problem is a classical problem in combinatorial optimization which finds important applications in different domains. In this paper we try to give a survey of results concerning algorithms, complexity, and applications of this problem, and also provide an updated bibliography. Of course, we build upon precursory works with similar goals [39, 232, 266].

1.1 Notations and Definitions

Throughout this paper, $G = (V, E)$ is an arbitrary undirected and weighted graph unless otherwise specified, where $V = \{1, 2, \dots, n\}$ is the vertex set of G (we use the terms *vertex* and *node* synonymously throughout), and $E \subseteq V \times V$ is the edge set of G . For each vertex $i \in V$, a positive weight w_i is associated with i , collected in the *weight vector* $w \in \mathbb{R}^n$. The symmetric $n \times n$ matrix $A_G = (a_{ij})_{(i,j) \in V \times V}$, where $a_{ij} = 1$ if $(i, j) \in E$ is an edge of G , and $a_{ij} = 0$ if $(i, j) \notin E$, is called the *adjacency matrix* of G . For any node v , let

$$N(v) = \{j \in V : a_{vj} = 1\}$$

denote the *neighborhood* of v in G , i.e. the set of all nodes adjacent to v .

The *complement graph* of $G = (V, E)$ is the graph $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(i, j) \mid i, j \in V, i \neq j \text{ and } (i, j) \notin E\}$. For a subset $S \subseteq V$, we define the weight of S to be $W(S) = \sum_{i \in S} w_i$, and call $G(S) = (S, E \cap S \times S)$ the *subgraph induced by S* .

A graph $G = (V, E)$ is *complete* if all its vertices are pairwise adjacent, i.e. $\forall i, j \in V$ with $i \neq j$, we have $(i, j) \in E$. A *clique* C is a subset of V such that $G(C)$ is complete. The *clique number* of G , denoted by $\omega(G)$ is the size of the maximum clique. The maximum clique problem asks for cliques of maximum cardinality (the cardinality of a set S , i.e., the number of its elements which will be denoted by $|S|$):

$$\omega(G) = \max\{|S| : S \text{ is a clique in } G\}.$$

The maximum weight clique problem asks for cliques of maximum weight. Given the weight vector $w \in \mathbb{R}^n$, the *weighted clique number* is the total weight of the maximum weight clique, and will be denoted by $\omega(G, w)$:

$$\omega(G, w) = \max\{W(S) : S \text{ is a clique in } G\}.$$

An *independent set* (*stable set*, *vertex packing*) is a subset of V , whose elements are pairwise nonadjacent. The maximum independent set problem asks for an independent set of maximum cardinality. The size of a maximum independent set is the *stability number* of G (denoted by $\alpha(G)$). The maximum weight independent set problem asks for an independent set of maximum weight. A *vertex cover* is a subset of V , such that every edge $(i, j) \in E$ has at least one endpoint i or j in the subset. The minimum vertex cover problem asks for a vertex cover of minimum cardinality. The

minimum weighted vertex cover problem asks for a vertex cover of minimum weight.

It is easy to see that S is a clique of G if and only if S is an independent set of \overline{G} , and if and only if $V \setminus S$ is a vertex cover of \overline{G} . Any result obtained for one of the above problems has its equivalent forms for the other problems. Hence $\alpha(G) = \omega(\overline{G})$, and, more generally, $\alpha(G, w) = \omega(\overline{G}, w)$.

We should distinguish a *maximum* clique (independent set) from a *maximal* clique (independent set). A maximal clique (independent set) is a clique (independent set) that is not a proper subset of any other clique (independent set). A maximum (weight) clique (independent set) is a maximal clique (independent set) that has the maximum cardinality (weight).

In the sequel, it will prove useful to consider Δ , the standard simplex in the n -dimensional Euclidean space \mathbb{R}^n :

$$\Delta = \left\{ x \in \mathbb{R}^n : x_i \geq 0 \text{ for all } i \in V, e^T x = 1 \right\}$$

where e^T denotes the transpose of the vector e consisting of unit entries (hence $e^T := (1, 1, \dots, 1)$ and $e^T x = \sum_{i \in V} x_i$). For a subset $S \subseteq V$ of vertices, we shall denote the face of Δ corresponding to S by

$$\Delta_S = \{x \in \Delta : x_i = 0 \text{ if } i \notin S\}.$$

Further, let us introduce the notion of a *characteristic vector* x^S which is the vector in Δ defined by $x_i^S = 1/|S|$ if $i \in S$ and $x_i^S = 0$ otherwise. Likewise, given a weight vector $w \in \mathbb{R}^n$, denote by $x^{S,w}$ the *weight barycenter* of Δ_S or, synonymously, the *weighted characteristic vector* of S , which is a vector with coordinates $x_i^{S,w} = w_i/W(S)$ if $i \in S$, and $x_i^{S,w} = 0$ otherwise. Of course, we have $x^S = x^{S,e}$.

2 Problem Formulations

The maximum clique problem has many equivalent formulations as an integer programming problem, or as a continuous nonconvex optimization problem. As with many problems of combinatorial optimization, using the appropriate formulation is of crucial importance in solving the problem. In addition, using different formulations, we gain more insight into the problem's complexity and we can prove interesting results.

2.1 Integer Programming Formulations

The simplest formulation of the maximum clique problem is the following *edge formulation*:

$$\begin{aligned} \max & \sum_{i=1}^n w_i x_i, \\ \text{s.t. } & x_i + x_j \leq 1, \forall (i, j) \in \overline{E}, \\ & x_i \in \{0, 1\}, i = 1, \dots, n. \end{aligned} \tag{1}$$

A polyhedral result concerning formulation (1) is due to Nemhauser and Trotter [244, 245]. In 1975 they found that if a variable x_i had integer value 1 in an optimal solution to the linear relaxation of (1), then $x_i = 1$ in at least one optimal solution to (1).

Theorem 2.1 (see [245], also [278]) *Let x be an optimum $(0, \frac{1}{2}, 1)$ -valued solution to the linear relaxation of (1), and let $P = \{j \mid x_j = 1\}$. Then there exists an optimum solution x^* to (1) such that $x_j^* = 1, \forall j \in P$.*

This theorem suggests an implicit enumerative algorithm for (1) via solving its linear relaxation problem. However, in most cases, few variables have integer values in an optimal solution to the linear relaxation of (1), and the gap between the optimal values of (1) and its linear relaxation problem is too large, which seriously restrict the use of this approach.

Let \mathcal{S} denote the set of all maximal independent sets of G . An alternative formulation is the following *independent set formulation*.

$$\begin{aligned} \max & \sum_{i=1}^n w_i x_i, \\ \text{s.t. } & \sum_{i \in S} x_i \leq 1, \forall S \in \mathcal{S}, \\ & x_i \in \{0, 1\}, i = 1, \dots, n. \end{aligned} \tag{2}$$

The advantage of formulation (2) over (1) is a smaller gap between the optimal values of (2) and its linear relaxation. However, since the number of constraints in (2) is exponential, solving the linear relaxation of (2) is not an easy problem. In fact, Grötschel et al. [150, 151] have shown that the linear relaxation problem of (2) is *NP-hard* on general graphs. They have also shown that the same problem is polynomially solvable on *perfect*

graphs. Furthermore, they have shown that a graph is perfect if and only if the optimal solution to the linear relaxation of (2) always takes integer values (see also [149] and [152]).

In 1990, Shor [293] considered an interesting formulation of the maximum weight independent set problem. Note that the maximum weight independent set problem can be formulated as

$$\begin{aligned} \min f(x) &= \sum_{i=1}^n w_i x_i, \\ \text{s.t. } x_i + x_j &\leq 1, \quad \forall (i, j) \in E, \quad x \in \{0, 1\}^n. \end{aligned} \tag{3}$$

The above formulation is equivalent to the following quadratically constrained global optimization problem

$$\begin{aligned} \min f(x) &= \sum_{i=1}^n w_i x_i, \\ \text{s.t. } x_i x_j &= 0, \quad \forall (i, j) \in E, \\ x_i^2 - x_i &= 0, \quad i = 1, 2, \dots, n. \end{aligned} \tag{4}$$

Applying dual quadratic estimates, Shor reported very good computational results using (4).

Next, we formulate the maximum clique, the maximum independent set and the maximum weight independent set problems as quadratic zero-one problems. We use I to denote the $n \times n$ identity matrix. To facilitate our discussion, define a transformation t from $\{0, 1\}^n$ to 2^V ,

$$t(x) = \{i \in V : x_i = 1\}, \quad \forall x \in \{0, 1\}^n.$$

Denote the inverse of t by t^{-1} . If $x = t^{-1}(S)$ for some $S \in 2^V$ then $x_i = 1$ if $i \in S$ and $x_i = 0$ if $i \notin S$, $i = 1, \dots, n$, i.e. $x = |S|x^S$ (cf. Section 1).

We can rewrite the maximum problem (1) as a minimization problem (when $x_i = 1$)

$$\begin{aligned} \min f(x) &= - \sum_{i=1}^n x_i, \\ \text{s.t. } x_i + x_j &\leq 1, \quad \forall (i, j) \in \overline{E}, \quad x \in \{0, 1\}^n. \end{aligned} \tag{5}$$

If x^* solves (5), then the set $C = t(x^*)$ is a maximum clique of G with $|C| = -z = -f(x^*)$.

Another way of stating the constraints for (5) is to make use of the fact that the quadratic expressions $x_i x_j = 0$ for all $(i, j) \in \overline{E}$, since for $x_i, x_j \in \{0, 1\}$, we have $x_i + x_j \leq 1$ if and only if $x_i x_j = 0$. The constraints in (5) can be removed by adding two times the quadratic terms to the objective function, which is now

$$f(x) = -\sum_{i=1}^n x_i + 2 \sum_{(i,j) \in \overline{E}, i > j} x_i x_j = x^T (A_{\overline{G}} - I)x.$$

The quadratic terms represent penalties for violations of $x_i x_j = 0$. This leads to the following theorem.

Theorem 2.2 *The maximum clique problem is equivalent to the following global quadratic zero-one problem*

$$\min f(x) = x^T A x, \quad (6)$$

$$\text{s.t. } x \in \{0, 1\}^n, \text{ where } A = A_{\overline{G}} - I.$$

If x^* solves (6), then the set C defined by $C = t(x^*)$ is a maximum clique of G with $|C| = -z = -f(x^*)$.

The off-diagonal elements of the matrix A are the same as the adjacency matrix of \overline{G} . Hence, formulations (5) and (6) are advantageous for dense graphs because a sparse data structure can be used (for details, see [264]; cf. also [123]). Following the equivalence of the maximum clique problem with the maximum independent set problem, we have

Theorem 2.3 *The maximum independent set problem is equivalent to the following global quadratic zero-one problem*

$$\min f(x) = x^T A x \quad (7)$$

$$\text{s.t. } x \in \{0, 1\}^n, \text{ where } A = A_G - I.$$

If x^* solves (7), then the set S defined by $S = t(x^*)$ is a maximum independent set of G with $|S| = -z = -f(x^*)$.

Next, we discuss the maximum weight independent set problem. The above theorems for the maximum clique problem and the maximum independent set problem can be regarded as a special case by taking $w = e$.

Theorem 2.4 *The maximum weight independent set problem is equivalent to the following global quadratic zero-one problem*

$$\min f(x) = x^T Ax, \quad (8)$$

$$\text{s.t. } x \in \{0, 1\}^n,$$

where $a_{ii} = -w_i$, $i = 1, \dots, n$, $a_{ij} = \frac{1}{2}(w_i + w_j)$, $\forall (i, j) \in E$, and $a_{ij} = 0$, $\forall (i, j) \in \overline{E}$.

Let x^* solve (8), then the set S defined by $S = t(x^*)$ is a maximum weight independent set of G with weight $W(S) = -z = -f(x^*)$.

In addition, we have the following relationship between the local minima of the quadratic problem and the maximal independent sets of a graph:

Theorem 2.5 *x is a discrete local solution to problem (8) if and only if x represents a maximal independent set of G .*

2.2 Continuous Formulations

In 1965, Motzkin and Straus [238] established a remarkable connection between the maximum clique problem and a certain standard quadratic programming problem providing an alternative proof of a slightly weaker version of the fundamental Turán theorem [311] (for a discussion see [54]). Let $G = (V, E)$ be an undirected (unweighted) graph, and let Δ denote the standard simplex in the n -dimensional Euclidean space \mathbb{R}^n , as well as Δ_S the face of Δ corresponding to a subset $S \subseteq V$ (cf. Section 1). Now, consider the following quadratic function

$$g(x) = x^T A_G x \quad (9)$$

where $A_G = (a_{ij})_{i,j \in V}$ is the adjacency matrix of G , and let x^* be a global maximizer of g on Δ . Motzkin and Straus proved that the clique number of G is related to $g(x^*)$ by the following formula:

$$\omega(G) = \frac{1}{1 - g(x^*)} \geq \frac{1}{1 - g(x)} \quad \forall x \in \Delta. \quad (10)$$

Additionally, they proved that a subset of vertices S is a maximum clique of G if and only if its characteristic vector x^S (see Section 1) is a global maximizer of g on Δ . In [274, 135], the Motzkin-Straus theorem has been extended by providing a characterization of *maximal* cliques in terms of *local*

maximizers of g on Δ . Moreover, in [135] Gibbons et al. characterize the first- and second-order optimality conditions of the Motzkin-Straus program and of a newly introduced parametrized version. A further generalization of the Motzkin-Straus theorem to hypergraphs can be found in [299].

One drawback associated with the original Motzkin-Straus formulation relates to the existence of spurious solutions, i.e., maximizers of g which are not in the form of characteristic vectors. This was observed empirically by Pardalos and Phillips [261] and more recently formalized by Pelillo and Jagota [274]. In principle, spurious solutions represent a problem since, while providing information about the size of the maximum clique, they do not allow us to easily extract its vertices.

The spurious solution problem has recently been solved by Bomze [56], by considering the following regularized version of function g :

$$\hat{g}(x) = x^T A_G x + \frac{1}{2} x^T x \quad (11)$$

which is obtained from (9) by substituting the adjacency matrix A_G of G with

$$\hat{A}_G = A_G + \frac{1}{2} I \quad (12)$$

where I is the identity matrix. The following is the spurious-free counterpart of the original Motzkin-Straus theorem [56].

Theorem 2.6 *Let S be a subset of vertices of a graph G , and let x^S be its characteristic vector. Then the following statements hold:*

- (a) *S is a maximum clique of G if and only if x^S is a global maximizer of the function \hat{g} over the simplex Δ . In this case, $\omega(G) = 1/2(1 - \hat{g}(x^S))$.*
- (b) *S is a maximal clique of G if and only if x^S is a local maximizer of \hat{g} in Δ .*
- (c) *All local (and hence global) maximizers x of \hat{g} over Δ are strict, and of the form $x = x^S$ for some $S \subseteq V$.*

Unlike the Motzkin-Straus formulation, the previous result guarantees that *all* maximizers of \hat{g} on Δ are strict, and are characteristic vectors of maximal/maximum cliques in the graph. In an exact sense, therefore, a one-to-one correspondence exists between maximal cliques and local maximizers of \hat{g} in Δ on the one hand, and maximum cliques and global maximizers

on the other hand. This solves the spurious solution problem in a definitive manner.

In a recent paper, Gibbons et al. [135] generalized the Motzkin-Straus theorem to the weighted case. They first reformulated the Motzkin-Straus problem as a minimization problem by considering the function

$$f(x) = x^T(I + A_{\bar{G}})x \quad (13)$$

where $A_{\bar{G}}$ is the adjacency matrix of the complement graph \bar{G} . It is straightforward to see that if x^* is a global minimizer of f in Δ , then we have:

$$\omega(G) = \frac{1}{f(x^*)}.$$

This is simply a different formulation of the Motzkin-Straus theorem. Given a weighted graph $G = (V, E)$ with weight vector w , they then considered the following classes of symmetric $n \times n$ matrices: let

$$\mathcal{M}(G) = \left\{ (b_{ij})_{i,j \in V} : b_{ij} \geq \frac{b_{ii} + b_{jj}}{2} \text{ if } (i, j) \notin E, b_{ij} = 0, \text{ otherwise} \right\},$$

and put

$$\mathcal{M}(G, w) = \left\{ B = (b_{ij})_{i,j \in V} \in \mathcal{M}(G) : b_{ii} = \frac{1}{w_i} \text{ and } b_{ij} = b_{ji} \text{ for all } i, j \right\}.$$

Now the following (generally indefinite) quadratic program is introduced in [135]:

$$\begin{aligned} & \text{minimize} && f(x) = x^T B x \\ & \text{subject to} && x \in \Delta \end{aligned} \quad (14)$$

where $B \in \mathcal{M}(G, w)$.

Using a proof technique suggested by Lovász (cf. [219]), the following result is shown, which establishes a correspondence between maximum weight cliques and global minimizers of (14).

Theorem 2.7 *Let G be an arbitrary weighted graph with positive weight vector $w \in \mathbb{R}^n$. Then, for any $B \in \mathcal{M}(G, w)$ we have: $\omega(G, w) = \frac{1}{f(x^*)}$ where x^* is a global minimizer of program (14).*

It can be seen that a subset S of vertices of a weighted graph G is a maximum weight clique if and only if its weighted characteristic vector $x^{S,w}$ (cf. Section 1) is a global minimizer of (14). Notice that the matrix $I + A_{\bar{G}}$

belongs to $\mathcal{M}(G, e)$. In other words, the Motzkin-Straus theorem turns out to be a special case of the preceding result.

As in the unweighted case, the existence of spurious solutions entails the lack of one-to-one correspondence between the solutions of the continuous problem and those of the original, discrete one. Bomze et al. [62] have recently characterized these spurious solutions and have introduced and studied a regularized version which avoids this kind of problems, exactly as in the unweighted case (for proofs see also [58]): instead of the Motzkin-Straus class $\mathcal{M}(G, w)$ here a different class $\mathcal{C}(G, w)$ of matrices is considered to be used as input data for problem (14): let

$$\mathcal{C}(G) = \{(c_{ij})_{i,j \in V} : c_{ij} \geq c_{ii} + c_{jj} \text{ if } (i, j) \notin E, c_{ij} = 0, \text{ otherwise}\},$$

and consider, given the weights w ,

$$\mathcal{C}(G, w) = \left\{ C = (c_{ij})_{i,j \in V} \in \mathcal{C}(G) : c_{ii} = \frac{1}{2w_i} \text{ and } c_{ij} = c_{ji} \text{ for all } i, j \right\}.$$

Theorem 2.8 *Let G be an arbitrary graph with positive weight vector $w \in \mathbb{R}^n$, and consider a matrix $C \in \mathcal{C}(G, w)$ in place of B for problem (14). Then the following assertions hold:*

- (a) *A vector $x \in \Delta$ is a local solution to problem (14) if and only if $x = x^{S,w}$, where S is a maximal clique.*
- (b) *A vector $x \in \Delta$ is a global solution to problem (14) if and only if $x = x^{S,w}$, where S is a maximum weight clique.*

Moreover, all local (and hence global) solutions to (14) are strict.

The class $\mathcal{C}(G, w)$ is isomorphic to the positive orthant in $\binom{n}{2} - |E|$ dimensions. This class is a polyhedral pointed cone with its apex given by the matrix $C(w)$ with entries

$$c_{ij}(w) = \begin{cases} \frac{1}{2w_i} & \text{if } i = j, \\ \frac{1}{2w_i} + \frac{1}{2w_j} & \text{if } i \neq j \text{ and } (i, j) \notin E, \\ 0 & \text{otherwise.} \end{cases} \quad (15)$$

Observe that in the unweighted case, $C(e) = ee^T - \hat{A}_G = \hat{A}_{\bar{G}}$, the regularized adjacency matrix of the complement graph \bar{G} . This reflects the elementary property that an independent set of G is a clique of \bar{G} . So,

while the local maximizers of $x^T \hat{A}_G x$ over Δ are exactly the barycenters x^C of maximal cliques C of G , the local minimizers of $x^T \hat{A}_G x$ over Δ are exactly the barycenters x^S of maximal independent sets S of G . Compare with Theorem 2.5 at the end of the preceding section. Note that within the Motzkin-Straus class $\mathcal{M}(G, e)$, there is no matrix with this straightforward interpretation.

3 Computational Complexity

The maximum clique problem is one of the first problems shown to be NP -complete [194], which means that, unless $P = NP$, a fact which is widely believed to be false, exact algorithms are guaranteed to return a solution only in a time which increases exponentially with the number of vertices in the graph [126]. Of course, the maximum independent set and the minimum vertex cover problems are NP -complete too. For a recent complexity result on the class of planar counting problems which includes the minimum vertex cover problem see [182].

Interest therefore has soon shifted towards characterizing the approximation properties of this problem (see [255, 70, 16] for an introduction to approximation complexity in optimization problems). Early works in this area go back to mid-1970's when Garey and Johnson [125] proved that if the maximum clique problem admits a polynomial-time approximation algorithm (i.e., it is approximable within a constant factor), then it has a polynomial-time approximation scheme (namely, it is approximable within *any* arbitrarily small factor). This is concisely expressed by saying that if the maximum clique problem belongs to the class APX, then it belongs to PTAS.

Johnson et al. [190] showed in 1988, based on one of their results given below, the nonexistence of a polynomial-delay algorithm for enumerating maximal cliques in reverse lexicographic order (if $P \neq NP$).

Theorem 3.1 *Given a graph G and a maximal independent set S , it is coNP-complete to prove whether S is the lexicographically last maximal independent set of G .*

In [257, 258], Papadimitriou and Yannakakis introduced the complexity classes $MAX\ NP$ and $MAX\ SNP$. They showed that all problems in $MAX\ NP$ admit a polynomial-time approximation algorithm, and that

many natural problems are complete in $MAX\ SNP$, under an approximation preserving reduction called the L-reduction. For example, the vertex cover problem (for constant degree graphs), max cut problem, dominating set problem, and the MAX 3-SAT problem are such complete problems [323]. Recently, this result has been extended to the vertex cover problem for cubic graphs [109].

If the solution to any of these complete problems can be approximated to arbitrary small constant factors, then the optimum solution to any problem in the class can be approximated to arbitrarily small constant factors. The question of whether such approximation schemes can be found for the complete problems in this class was left unresolved.

By using randomized reductions, Berman and Schnitger [45, 46] have shown that even an $O(n^\varepsilon)$ -approximation algorithm for maximum clique (for some small enough ε) would yield a *randomized* polynomial-time approximation scheme for MAX 3-SAT, where n is the number of vertices in the graph (see also Feige et al. [111], where a connection between approximation complexity and interactive proof systems is discussed). In [7], Alon et al. derandomized Berman and Schnitger's reduction by using the so-called *expander graphs* (also used by Arora and Safra in [14]), and obtained the following result: there exists an $\varepsilon > 0$ such that if the maximum clique problem is approximable within $O(n^\varepsilon)$ then MAX 3-SAT is in PTAS.

Panconesi and Ranjan [253, 254] expanded on the work of Papadimitriou and Yannakakis. They introduced the complexity class $MAX\ \Pi_1$, and proved that all complete problems in this class do not admit a polynomial-time approximation algorithm, unless $P = NP$. They also showed that the maximum clique problem does not belong to $MAX\ NP$ but is in the lowest subclass of $MAX\ \Pi_1$, see also [313].

In 1991, Crescenzi, Fiorini and Silvestri [97] proved that all problems in $MAX\ NP$ are strongly reducible to the maximum clique problem. As a consequence, if the maximum clique is approximable within a constant factor, then all problems in $MAX\ NP$ can be approximated within any arbitrarily small factor. This provided evidence that the maximum clique problem does not have a polynomial-time approximation algorithm. Stronger evidence of this fact was given independently in the same year, when Feige et al. [111] (see also [112]) proved that if there is a polynomial-time algorithm that approximates the maximum clique problem within a factor of $2^{\log^{1-\varepsilon} n}$, then any NP problem can be solved in “quasi-polynomial” time (i.e., in $2^{\log^{O(1)} n}$ time).

A breakthrough in approximation complexity is the result by Arora et al. [13], [14]. It is shown that the maximum number of satisfiable clauses in a 3-SAT formula (MAX 3-SAT) cannot be approximated to arbitrary small constants (unless $P = NP$), thus resolving the open question in [257, 258]. This immediately shows the difficulty of finding good approximate solutions to all the above listed problems. In particular, it is shown that no polynomial-time algorithm can approximate the maximum clique size within a factor of n^ε ($\varepsilon > 0$), unless $P = NP$ (by using the results of Feige et al. [111]). The work of Arora et al. stimulated much research, and many investigators have progressively refined the exact approximation ratio for which approximating maximum clique becomes intractable [40, 42, 113, 41, 160].

The best polynomial-time approximation algorithm for the maximum clique problem was developed by Boppana and Halldórsson [66], and achieves an approximation ratio of $n^{1-o(1)}$. In [161], Hastad shows that this is actually the best we can achieve, by proving that, unless $NP = coR$, the maximum clique problem cannot be approximated within a factor of $n^{1-\varepsilon}$, for any $\varepsilon > 0$.

Although these complexity results characterize worst case instances, they nevertheless indicate that the maximum clique problem is indeed a very difficult problem to solve.

Some other results in the literature concerning the approximation of the maximum clique/independent set problem on arbitrary or special graphs can be found in [88], [90], [283], [97], [66] [240], [84], [212], [291], [85], [196].

If we restrict ourselves to graphs with special structure, then in many cases the maximum clique/independent set problem can be solved in polynomial time. For example, Balas et al. [25] introduced several classes of graphs and showed that the maximum weight clique problem can be solved in polynomial time on them. Balas and Yu [32] discuss classes of graphs that have polynomially many maximal cliques. On those graphs, the maximum weight clique problem can also be solved in polynomial time.

A well known class of graphs where the maximum clique problem is polynomially solvable, is the class of perfect graphs [43]. A graph G is called *perfect* if every induced subgraph of G has the property that the size of its maximum clique equals the minimum number of independent sets needed to cover all the vertices (commonly called a *coloring* in the literature). Since the complement graph of a perfect graph is also perfect, the maximum clique problem can be solved in polynomial time on perfect graphs and their complements. The class of perfect graphs contains many well known graphs

in the literature [145], among them *bipartite* graphs, *interval* graphs, and *triangulated* graphs [127], [119], [287], [288], [306]. Examples of more recently found perfect graphs are *Meyniel graphs* [233], [78], *quasi parity* graphs [234], *weakly triangulated* graphs [163], [164], *perfectly orderable* graphs [95], and *unimodular* graphs [165].

A class of graphs that is closely related to the perfect graphs is the *t-perfect* graphs. This class of graphs was defined in [92]. Polynomial algorithms for the maximum weight independent set problem on *t-perfect* graphs exist [151]. The class of *t-perfect* graphs contains *bipartite graphs*, *series-parallel graphs* [106], [92], [67], and *strongly t-perfect graphs* [133]. For a polynomial time algorithm for solving the maximum weight independent set problem on a bipartite graph $G(V_1, V_2)$ see the book by Lawler [209].

Other special classes of graphs where the maximum clique/independent set problem have been studied in the literature can be found in [23], [48], [49], [79], [83], [88], [89], [91], [90], [107], [114], [128], [129], [146], [147], [153], [170], [179], [180], [181], [197], [196], [202], [223], [227], [228], [235], [242], [248], [256], [281], [289], [290], [303], [86], [102], and [324].

We should note here that the weighted or unweighted version of the maximum clique problem, the maximum independent set problem, and the minimum vertex cover problem may, with respect to hardness, not be equivalent on graphs with special structures.

4 Bounds and Estimates

We now present bounds for the clique number $\omega(G)$ and discuss also the complexity of their calculation; most of them are based on properties of the matrix A_G .

A graph is said to be connected if each pair of nodes is joined by a path of edges. Since the undirected graphs we are considering can always be subdivided into connected subgraphs, we consider here only connected graphs. These have adjacency matrices with the property of being irreducible.

Let m be the number of edges of the graph and $\delta = \frac{2m}{n^2}$ the density of 1's in A_G ; for connected graphs we have $2\frac{n-1}{n^2} \leq \delta \leq \frac{n-1}{n}$. The request that a graph with a clique number $\omega(G)$ be connected gives the simple bound [12]

$$\omega(G) \leq \frac{3 + \sqrt{9 - 8(n - m)}}{2}. \quad (16)$$

Let $\lambda_1, \dots, \lambda_n$ be the eigenvalues of A_G and $\rho(A_G) = \max_{i=1,\dots,n} |\lambda_i|$ its spectral radius. A frequently cited upper bound, appeared for the first time

in 1967 [316], is

$$\omega(G) \leq \rho(A_G) + 1 \quad (17)$$

the equality holding if and only if the graph is complete. To prove this relation we use (10): let x^S be the characteristic vector of a maximum clique, then $(x^S)^T A_G x^S = 1 - \frac{1}{\omega(G)}$ and also $(x^S)^T x^S = \frac{1}{\omega(G)}$. Bound (17) derives from the general property $\frac{x^T A_G x}{x^T x} \leq \rho(A_G)$. With (17) one can apply to $\omega(G)$ all the bounds valid for $\rho(A_G)$ (for a partial list see [75]).

Since A_G is irreducible, symmetric and with non-negative elements, all its eigenvalues are real and the largest one, the *Perron root* $\lambda_P := \rho(A_G)$, is simple and dominating (see e.g. [176] pp. 507 ff.). Moreover there may be at most one negative eigenvalue $-\lambda_P$ with the same absolute value, and this happens if and only if the graph is bipartite (see [100], Theorems 3.11 and 3.4). Finally, all the components of the only *Perron eigenvector* x_P (such that $A_G x_P = \lambda_P x_P$) are strictly positive, i.e. $x_P > 0$.

Straightforward calculation of Perron root and eigenvector is not the most efficient method for large n : it is faster to exploit the exponentially fast convergence of $\lim_{m \rightarrow \infty} \left(\frac{1}{\lambda_P} A_G\right)^m = x_P x_P^T$ valid if A_G is primitive (when this does not happen one takes the primitive matrix $\hat{A}_G = A_G + \frac{1}{2}I$). Successive squaring of A_G allows to perform this calculation, with arbitrary precision, essentially in $O(n^3)$ [176].

Let N_{-1} be the number of eigenvalues of A_G that do not exceed -1 and N_0 the number of zero eigenvalues. Amin and Hakimi [12] proved that

$$\omega(G) \leq N_{-1} + 1 < n - N_0 + 1, \quad (18)$$

with equality holding if the graph is complete multipartite; also the latter bound can be calculated in $O(n^3)$.

A geometrical formulation of the maximum clique problem [76] in complex space produces the bound

$$\omega(G) \leq \frac{n + \bar{N}_0}{2} \quad (19)$$

where \bar{N}_0 is the number of zero eigenvalues of the adjacency matrix of the complement graph \bar{G} . Also in this case the calculation of \bar{N}_0 can be done in $O(n^3)$.

While bound (16) can be calculated in $O(n)$, bounds (17), (18) and (19) require more work but are usually much tighter. Experimentally, most of the times (18) is the sharpest bound, but nothing can be said in general because

one can devise graphs for which bounds (17) or (19) are respectively the best bound. Consequently the safest strategy is to calculate all bounds and to choose the sharpest.

A different bound derives from the ‘Sandwich theorem’ on which Knuth published a review [204]. This theorem focuses on the *Lovász number* [218] (cf. also [292]) $\theta(\overline{G})$ and states that this number is sandwiched between the two *NP*-hard quantities, the clique number $\omega(G)$ and the *chromatic number* $\chi(G)$, which is the minimum number of colors needed to color the vertices of G :

$$\omega(G) \leq \theta(\overline{G}) \leq \chi(G). \quad (20)$$

Since $\theta(\overline{G})$ is computable in polynomial time, e.g. with interior point methods like *semidefinite programming* (see, e.g. [140, 203], and for special sparsity methods related to the maximum clique problem [123]) via the (dual) eigenvalue bound identity [166]

$$\theta(\overline{G}) = \min\{\lambda_{\max}(ee^T + Y) : Y \in \mathcal{Y}(G)\}$$

with

$$\mathcal{Y}(G) = \{Y(n \times n) \text{ matrix} : Y^T = Y, Y_{ij} = 0 \text{ if } (i, j) \in E\},$$

the Sandwich theorem shows also how, for perfect graphs (for which $\omega(G) = \chi(G)$ holds) the maximum clique number can be computed in polynomial time.

Bounding $\omega(G)$ from below is harder; a simple bound derives from application of (10) to e , which gives $\frac{e^T}{n} A_G \frac{e}{n} = \frac{2m}{n^2} \leq 1 - \frac{1}{\omega(G)}$ and thus

$$\omega(G) \geq \frac{1}{1 - \delta} > 1. \quad (21)$$

Wilf [318] obtained a sharper bound considering Perron eigenvector x_P properly normalized with $s := e^T x_P$, yielding via (10) $\frac{(x_P)^T}{s} A_G \frac{x_P}{s} = \frac{\lambda_P}{s^2} \leq 1 - \frac{1}{\omega(G)}$ and so

$$\omega(G) \geq \frac{\lambda_P}{s^2 - \lambda_P} + 1 \geq \frac{\lambda_P}{n - \lambda_P} + 1 \quad (22)$$

with equality holding if and only if the graph is complete; the rightmost inequality is easily derived from $x_P^T x_P = 1$ and the Cauchy/Schwarz inequality which gives $s^2 = (e^T x_P)^2 \leq (e^T e)(x_P^T x_P) = n$.

If one knows the full set of eigenvalues and eigenvectors of A_G this bound can be improved strictly. In fact with the full set of eigenvectors one can

always build $x^* \in \Delta$ such that $g_* := (x^*)^T A_G x^* > \frac{\lambda_P}{s^2}$ (see [75] for details) and (10) immediately gives

$$\omega(G) \geq \frac{1}{1 - g_*}. \quad (23)$$

Bound (21) is easy to compute while (22), requiring Perron eigenvector and eigenvalue, is a little harder. To find g_* of (23) one needs the full set of eigenvalues and eigenvectors of A_G and, consequently, this bound is the hardest to calculate but also, provably, the sharpest one. A final, obvious, remark, is that, being $\omega(G)$ integer, any non integer bound can be sharpened applying the appropriate ceiling $\lceil \rceil$ or floor $\lfloor \rfloor$ operators.

There exist fortunate cases in which these bounds almost solve the maximum clique problem, like, for example, the 64-node graph ‘hamming6-2’ of the DIMACS challenge [189]. Application of (18) and (23) allows to state that for this graph $32 \leq \omega(G) \leq 33$.

Considering just a subset of all possible graphs one can exploit the characteristics of the given subset to obtain sharper bounds. A much studied case is that of randomly generated graphs. For these graphs there exists a well established theory [53] and research proceeds in several directions; for example see [55], [122], [193], [229] and [236] for eigenvalues of random graphs. In the specific field of maximum cliques a well known result, due to Matula, accurately predicts the size of the maximum clique when the number of vertices n is sufficiently large [230]. In particular Matula was able to prove that the probability that

$$\lfloor M(n, \delta) \rfloor \leq \omega(G) \leq \lceil M(n, \delta) \rceil \quad (24)$$

tends to 1 when $n \rightarrow \infty$ and where

$$M(n, \delta) = 2 \log_{1/\delta} n - 2 \log_{1/\delta} \log_{1/\delta} n + 2 \log_{1/\delta} \frac{e}{2} + 1. \quad (25)$$

δ being the density of the random graphs under consideration. There is also another result [55] about the smallest maximal clique and it shows that its size is almost surely $M(n, \delta)/2$ in the limit of large n .

Another interesting result for these graphs is known as the Jerrum conjecture [187] (see also [10] for recent, related results). It states that in large random graphs of density $\delta = 1/2$ there is no polynomial time algorithm that, with probability greater than 1/2, can find a clique larger than the smallest maximal clique.

5 Exact Algorithms

5.1 Enumerative Algorithms

The first algorithm for enumerating all cliques of an arbitrary graph in the literature is probably due to Harary and Ross [158]. In 1957, they proposed an inductive method that first identified all the cliques of a special graph with no more than three cliques. Then the problem on general graphs is reduced to this special case. Their work was stimulated by the matrix manipulation problem of sociometric data to find a complete identification of cliques.

Early works following that of Harary and Ross can be found in [222, 268, 65, 226, 38, 108]. What Paull and Unger [268], and Marcus [226] proposed were algorithms to minimize the number of rows in a flow table for a sequential switching function. Bonner addresses in [65] the clustering problem in information systems. Bednarek and Taulbee [38] proposed algorithms for generating all maximal chains of a set with a binary relation defined on it. Although these problems come from different fields and apparently deal with different problems, they are solving the same problem of enumerating all cliques of a graph. With the technology at that time, these early algorithms could only be tested on special graphs.

In 1970, Auguston and Minker [15] investigated several graph theoretic clustering techniques used in information systems. In their work, the algorithm of Bierstone and that of Bonner were tested. The method used in both algorithms was called the *vertex sequence method* or *point removal method*. This method produces cliques of G from the cliques of $G \setminus \{v\}$ with $v \in V$. From their computational results, they found the algorithm of Bierstone was more efficient. The original work of Bierstone was not published. The version of Bierstone's algorithm contained in Auguston and Minker [15] had two errors that were corrected by Mulligan and Corneil [239] in 1972.

Then in 1973, two new algorithms using the *backtracking method* were proposed by Akkoyunlu [8], and by Bron and Kerbosch [73]. The advantage of the backtracking method is the elimination of the redundancy in generating the same clique. What was more important for these two algorithms was their polynomial storage requirements. For example, the Bron and Kerbosch algorithm requires at most $\frac{1}{2}n(n+3)$ storage space. Bron and Kerbosch tested their algorithm on graphs of 10 to 50 vertices and densities ranging from 10% to 95%. Here the density was defined as the probability of a pair of vertices being connected. They found their algorithm was much

more efficient than Bierstone's algorithm. One very interesting phenomenon from their test was the ratio of CPU time over the number of cliques of the graph, as they put it, "hardly dependent on the size of the graph". Bron and Kerbosch's algorithm is *Algorithm 457* in the ACM collection.

More enumerative algorithms were proposed in the 70's following that of Bron and Kerbosch, among them [250, 249, 231, 191, 192, 210, 310, 132]. The algorithm of Osteen and Tou [250] was an improved version of the point removal method. Osteen's [249] algorithm was designed for a special class of graphs. The algorithm of Meeusen and Cuyvers [231] started with decomposing a graph into subgraphs satisfying *the chain of subsets in G* requirement. Such a decomposition had the property that every clique is contained completely in at least one subgraph. Based on this property, they proposed an algorithm to find all cliques of a graph. The work of Johnston [192] contains a family of algorithms that are variations of Bron and Kerbosch's algorithm. By comparing several algorithms computationally, Johnston [192] concluded that the Bron and Kerbosch algorithm was one of the most efficient algorithms.

Tsukiyama et al. [310] proposed an enumerative algorithm that combined the approaches used in [15, 73], and by Akkoyunlu [8]. The result was an algorithm with time complexity of $O(nm\mu)$ and storage requirement of $O(n+m)$, where n, m, μ are the number of vertices, edges and maximal cliques of a graph. As pointed out in [310], this bound is stronger than the earlier bound of $O(\mu^2)$ from [15]. The algorithm of Gerhards and Lindenberg [132] started with partial cliques related to fixed vertices of G . Then, cliques were generated from these partial cliques. Their computational results suggested their proposed algorithm was as efficient as that in [73] for general graphs, but more efficient on sparse graphs.

In 1980's, other proposed algorithms include those in [215, 214, 87, 309, 190].

Loukakis and Tsouros [215] proposed a depth-first enumerative algorithm that generated all maximal independent sets lexicographically. They compared their algorithm with the algorithms of [73] and [310]. Their computational results on graphs of up to 220 vertices suggested the superior efficiency of their algorithm: which was two to fifteen times faster than that in [73], and three times faster than that in [310]. Two years later, Loukakis [214] claimed an additional improvement of a factor three speed-up compared to [215], tested on graphs of 30 to 220 vertices with densities from 10% (for small graphs) to 90% (for large graphs).

In 1988, Johnson et al. [190] proposed an algorithm that enumerated

all maximal independent sets in lexicographic order. The algorithm has an $O(n^3)$ delay between the generation of two subsequent independent sets; cf. the complexity result Theorem 3.1. Chiba and Nishizeki's [87] algorithm lists all cliques with time complexity of $O(a(G)m\mu)$, where $a(G)$ is the *arboricity* of graph G . This is an improvement over the time complexity in [310].

Finally, Tomita et al. [309] proposed a modified Bron and Kerbosch [73] algorithm and claimed its time complexity to be $O(3^{n/3})$. As they pointed out, this was the best one could hope for since the Moon and Moser graphs [237] have $3^{n/3}$ maximal cliques.

5.2 Exact Algorithms for the Unweighted Case

If our goal is to find a maximum clique or just the size of a maximum clique, a lot of work can be saved from the above enumerative algorithms. Because once we find a clique, we only need to enumerate cliques better than the current best clique. Modifying the enumerative algorithms based on this argument results in various implicit enumerative algorithms. This argument can also be used in designing implicit enumerative algorithms.

The most well known and commonly used implicit enumerative method for the maximum clique problem is the *branch and bound* method. Background information on how branch and bound method works can be found in, for example, [28] and [246]. The key issues in a branch and bound algorithm for the maximum clique problem are:

1. How to find a good lower bound, i.e. a clique of large size?
2. How to find a good upper bound on the size of maximum clique?
3. How to branch, i.e., break a problem into smaller subproblems?

Implicit enumerative algorithms for the maximum clique/independent set problem started in the 1970's by Desler and Hakimi [105], Tarjan [304], and Houck [177]. These early works were improved in 1977 by [305] and [178]. Tarjan and Trojanowski proposed in [305] a recursive algorithm for the maximum independent set problem. They show their algorithm has a time complexity of $O(2^{n/3})$. This time bound illustrates that it is possible to solve a *NP*-complete problem much faster than the simple, enumerative approach. In the same year, Chvátal used a certain type of *recursive proofs* in [93] to show the upper bound on the stability number "has length at least $O(c^n)$ ", where $c > 1$ is a constant. The work of Houck and Vemuganti [178] exploited the relationship between the maximum independent set and a special class of bipartite graphs. They used this relationship to find an initial solution in their algorithm for the maximum independent set problem.

Most algorithms in the literature for the maximum clique/independent set problem were proposed in the 1980's. For example, in 1982, Loukakis and Tsouros [216] proposed a *tree search* algorithm that finds the size of a maximum independent set. Then in 1984, Ebenegger et al. [110] proposed another algorithm for finding the stability number of a graph. Their approach is based on the relationship between the maximization of a pseudo-Boolean function and the stability number of a graph. Computational tests on graphs with up to 100 vertices were reported in [110].

One of the most important contributions in the 1980's on practical algorithms for the maximum clique problem is due to Balas and Yu [31]. In their algorithm, the implicit enumeration was implemented in a new way. The idea of their approach is as follows. First, find a maximal induced triangulated subgraph D of G . Once D is found, find a maximum clique of D . This clique provides a lower bound and a feasible solution to the maximum clique problem. Then, they used a heuristic coloring procedure to extend D to a larger (maximal) subgraph that had no clique better than the current best clique. The importance of this second step is that it helps to reduce the number of subproblems generated from each node of the search tree, which in turn, reduces the size of the whole search tree. They solved the maximum clique problem on graphs of up to 400 vertices and 30,000 edges. Comparing their algorithm with other such algorithms, they found their algorithm not only generated a smaller search tree, but also required less CPU time.

In 1986, Kikusts [199] proposed a branch and bound algorithm for the maximum independent set problem based on a new recursive relation for the stability number of a graph G . Namely,

$$\alpha(G) = \max\{1 + \alpha(G \setminus [\{v\} \cup N(v)]), \alpha'_v\}, \quad (26)$$

where $\alpha'_v = \max\{|I| : I \subseteq G \setminus \{v\} \text{ is independent with } |I \cap N(v)| \geq 2|\}$. This relation is different from the recursive relation

$$\alpha(G) = \max\{1 + \alpha(G \setminus [\{v\} \cup N(v)]), \alpha(G \setminus \{v\})\}, \quad (27)$$

traditionally used in designing branch and bound algorithms for the maximum independent set problem. Intuitively, relation (26) is stronger than relation (27). However, the trade off is a more complicated situation in (26). Some computational results were provided in [199] without comparison to other algorithms.

Also in 1986, Robson [286] proposed a modified recursive algorithm of Tarjan and Trojanowski [305]. Robson showed through a detailed case analysis that his algorithm had a time complexity of $O(2^{0.276n})$. This is an

improvement over the time complexity $O(2^{n/3})$ of [305]. Here we want to mention the complexity proof of a similar recursive algorithm by Wilf [317]. Although Wilf's complexity of $O(1.39^n)$ is not as tight as that of [305], his proof is much simpler. Also in [317], Wilf proved (under certain probabilistic assumptions) that the average number of independent sets in a graph with n vertices is given by:

$$I_n = \sum_{k=0}^n \binom{n}{k} 2^{-k(k-1)/2}. \quad (28)$$

Using (28), it can be shown that the average complexity of a backtracking algorithm for the maximum independent set problem is subexponential, because I_n grows at the rate of $O(n^{\log n})$.

In late 1980's, new algorithms were proposed in [307, 130, 131] and in [265] (published in 1992). The algorithm of Tomita et al. [307] uses a greedy coloring algorithm to get an upper bound on the size of the maximum clique. Some computational results can be found in [131] and [307]. Gendreau et al. [130, 131] use an implicit enumerative algorithm. In [131], the branching rule (the selection of the next vertex to branch) is based on the number of triangles a vertex belongs to. The algorithm of Pardalos and Rodgers [265] is based on an unconstrained quadratic zero-one programming formulation of the maximum clique problem. In their work, the merit of two different branching rules, *greedy* and *nongreedy*, are tested.

In the 1990's, more algorithms were proposed, for example in [261, 121, 80, 21, 19, 321, 98].

Pardalos and Phillips [261] formulate the maximum clique problem as an indefinite quadratic global optimization problem with linear constraints. The algorithm of Friden et al. [121] is a branch and bound algorithm for the maximum independent set problem, employing tabu search techniques in finding lower and upper bounds. Carraghan and Pardalos [80] propose an implicit enumerative algorithm which is very efficient for sparse graphs (see also [262]). Their branching rule corresponds to the *nongreedy* rule described in [265]. Using this algorithm, they are able to solve problems on graphs of 500 vertices. Some test instances of graphs with 1000 and 2000 vertices are also examined. Since the algorithm is transparent and the code is publicly available, it can serve as a benchmark for comparing different algorithms.

Babel and Tinhofer propose in [21] a branch and bound algorithm for the maximum clique problem. The main ingredient of their algorithm is the use of a fast and relatively good heuristic for the minimum coloring

problem proposed by Brelaz [71]. The coloring heuristic is called the *degree of saturation largest first* (DSATUR). Applying DSATUR to a graph, one can find an upper bound on the size of the maximum clique as well as a maximal clique (thus, a lower bound). Babel and Tinhofer exploit this distinct feature and apply DSATUR at each node of the search tree. They tested their algorithm on graphs of 100 to 400 vertices with varying densities. In [19], Babel further refines and improves the algorithm of [21].

Also in 1991, based on the fact that a fractional coloring solution provides a tighter upper bound than an integer coloring solution for the maximum clique problem, a heuristic for the fractional coloring problem is proposed by Xue in [321] and used in a branch and bound algorithm for the maximum clique problem. Substantial reduction in the search tree size and the improvement in efficiency of the branch and bound algorithm are observed because of the use of this new bounding procedure. Details about the method and how to extended it to the weighted case can also be found in [30].

Della Croce and Tadei reformulate in [98] the maximum clique problem into a multivariate binary knapsack problem (cf. also [252]) and combine a greedy algorithm with branch and bound methods. In [69], Bourjolly et al. propose a column-generation method embedded in a branch and bound algorithm, to obtain competitive lower bounds for the maximum clique problem, and also to obtain valid cuts for the linear relaxation of the minimal vertex cover problem. Bourjolly and coworkers presented in [68] a new approach for minimizing general quadratic 0-1 functions related to the satisfiability of a sequence of Boolean expressions. They developed lower bounding procedures for minimizing such functions and interpreted them in the context of the maximum independent set problem. These bounding rules were then incorporated in a standard branch and bound algorithm which was tested on various DIMACS benchmark graphs (cf. Section 6).

The continuous formulation in Theorem 2.6 allows to convert any finite exact procedure for solving indefinite quadratic programming problems globally, e.g. that in [60], into an exact algorithm. More adapted to our problem are solvers specifically dedicated to global optimization of quadratic forms over a simplex, which problems also recently were introduced under the name *standard quadratic problems* [57, 58]. The algorithms proposed there typically find a local solution for this sort of problems, and then either generate a certificate for global optimality, or produce an improving feasible point from which a new (monotonic) local search can be started. Both this escape step and the optimality certificate rely on the *copositivity* property of

symmetric matrices which generalizes semi-definiteness: a $(n \times n)$ matrix Q is said to be copositive, if it generates a quadratic form taking no negative values on the positive orthant, or, equivalently, if

$$x^T Q x \geq 0 \quad \forall x \in \Delta.$$

In [57] it is shown that a maximal clique S (corresponding to a local maximizer x^S of $x^T \hat{A}_G x$ over Δ) is a maximum clique if and only if the matrix $Q_S = (2|S|-1)ee^T - 2|S|\hat{A}_G$ is copositive (observe that Q_S only depends on $|S|$). If, however, $x^T Q_S x < 0$ for some $x \in \Delta$, then $x^T \hat{A}_G x > (x^S)^T \hat{A}_G (x^S)$ so that an escape direction is found. Now while checking copositivity is again NP -hard, the full arsenal of finite procedures for this goal can be employed, see [57] and the references therein. In [57] also some experiments are conducted which show that even suboptimal local solutions serve well as heuristics in hard cases (i.e. large and dense graphs) where the (in principle finite) algorithm must be stopped prematurely.

A similar, very recent approach called *Genetic Engineering via Negative Fitness (GENF)* [64] also focuses on copositivity analysis, which is attacked by a block pivoting argument. Despite of the similarity in nomenclature, GENF is *not* a genetic algorithm (cf. Section 6.3.3), but rather a deterministic, finite and exact recursive procedure which produces simultaneously large cliques and large stable sets for the same graph (large stable sets are needed to obtain good pivots for efficient dimensional reduction).

5.3 Exact Algorithms for the Weighted Case

Algorithms for finding a maximum weight independent set of an arbitrary graph started in 1975 by Nemhauser and Trotter [245]. They considered the polyhedron relationships between the edge formulation (1) of the maximum weight independent set problem and its linear relaxation problem. Their main results were given as theorem 2.1 in section 2.1 of the present paper. Based on this result, they proposed an algorithm for the maximum weight independent set problem.

In 1977, Balas and Samuelsson [27] proposed an algorithm that solved the minimum vertex covering problem (a weighted version was also announced in [27]). Their algorithm was based on the relationship between the integer dual feasible solution and an equivalent linear programming for the vertex covering problem. Labeling procedures were designed to generate and improve vertex covers. When these labeling procedures could not continue,

branch and bound was used. The computational tests in [245] and [27] were conducted on unweighted graphs of up to 50 vertices.

In 1983, Loukakis and Tsouros [217] proposed an algorithm for the maximum weight independent set problem. It seems that almost nothing else appeared in the literature until late 1980's and early 1990's. Recently published algorithms we are aware of for the maximum weight clique/independent set problem are due to Pardalos and Desai [260], Balas and Xue [29], and Nemhauser and Sigismondi [243].

The algorithm proposed by Pardalos and Desai [260] was based on an unconstrained quadratic 0-1 formulation of the maximum weight independent set problem. Their algorithm (for maximum weight independent set) used the *nongreedy* search strategy described in [265]. With this algorithm they were able to solve problems of up to 500 vertices with different densities. In an unpublished paper [81], Carraghan and Pardalos test a weighted version of their algorithm from [80], and it turns out that this is more efficient than that of Pardalos and Desai in [260].

Balas and Xue [29] extend the algorithm of Balas and Yu [31] to the weighted case. To accomplish this, a minimum weighted coloring of a triangulated graph is needed. Although the minimum weighted coloring problem on triangulated graphs is known to be in class P (see [145]), there was no algorithm in the literature that had reasonable time complexity. In [29] a combinatorial algorithm for this problem with a time complexity of $O(n^2)$ is proposed and used to extended the algorithm of [31] to the weighted case. Computational results (graphs of size up to 2000 vertices are solved on a workstation) show that the size of the search tree is greatly reduced and the CPU time is much smaller than other such algorithms, especially for large, dense graphs.

In [30], Balas and Xue propose a fast heuristic for the weighted fractional coloring problem and used this heuristic as an upper bounding procedure in a branch and bound algorithm for the maximum weight clique problem. Comparing with the method in [29], computational results show the reduction in search tree size and the improved efficiency of the resulting algorithm.

The algorithm of Nemhauser and Sigismondi [243] uses the polyhedron approach. They attack the problem by first solving the linear relaxation of the corresponding integer programming problem. If the optimal solution to the relaxation problem is integer, we are done. Otherwise, sets of valid inequalities are generated and added into the relaxed problem to cut off the current fractional solution. Attention is focused to some classes of facet defining inequalities for the maximum independent set problem. Since not

all facets for the clique/independent set polytope are known, there is no guarantee that a fractional solution would always be cut off. When this happened, Nemhauser and Sigismondi switch in [243] to a branch and bound method. From their computational test, they found too many iterations in solving the linear relaxation problems. The largest graphs they tried to solve had up to 120 vertices. Similar approaches, partly only for the unweighted case, can be found in [244, 18, 24].

In an algorithm published in 1994, Babel [20] uses a branch and bound approach as follows: upper and lower bounds for the maximum weight of cliques are found by coloring the weighted graph, where the number of colors represents the total sum of all weights. So to process a graph of order 500 with weights out of $\{1, \dots, 10\}$ we may have to deal with up to 5000 colors. The branching part of Babel's algorithm divides the bounded search-tree into smaller subproblems, the branching decisions depending on a specific order of all possible remaining nodes.

Similarly to the unweighted case, the continuous formulation of Theorem 2.8 provides exact procedures for the maximum weight clique problem out of any finite (global) indefinite quadratic program solver. Now, as easily can be seen, for any clique S of G , and any matrix $C \in \mathcal{C}(G, w)$, we have $(x^S)^T C(x^S) = 1/[2W(S)]$. If S is a maximal clique, then x^S is a local minimizer of the quadratic form generated by C over Δ , and so S is a maximum weight clique if and only if the matrix $Q_{S,w} = 2W(S)C - ee^T$ is copositive. Observe that $Q_{S,w}$ depends on S and w only via $W(S) = \sum_{i \in S} w_i$. Specializing $C = C(e)$ we see that in the unweighted case $w = e$ we again get $Q_{S,e} = 2|S|(ee^T - \hat{A}_G) - ee^T = Q_S$, and regarding copositivity detection the arguments at the end of the preceding section apply as well. This approach is followed in [62], with some empirical evidence that (local) solutions obtained even in case of premature stopping of the algorithm (which in principle is finite) may again yield satisfying approximations. Also, the GENF approach [64] is applicable here though the maximizer yielding auxiliary pivots have not the same nice interpretation as in the unweighted case.

6 Heuristics

Because of the computational complexity of the maximum clique problem, which as seen in Section 3 is also hard to approximate, much effort has recently been directed towards devising efficient heuristics, for which no formal guarantee of performance may be provided, but which are anyway of

interest in practical applications.

In a branch and bound algorithm for the maximum clique problem, its lower bounding procedure usually provides a maximal clique which can be used to approximate the maximum clique of G . Since different branch and bound algorithms tend to have different bounding procedures, they provide different heuristics for the maximum clique problem. On the other hand, there are many heuristics in the literature explicitly designed to find approximation solutions to the maximum clique problem. These heuristics are usually more complicated than the lower bounding procedures from a branch and bound algorithm. In this section we provide a description of these procedures.

Lacking (almost by definition) a general theory of how these algorithms work, their evaluation is essentially based on massive experimentation. In order to facilitate comparisons among different heuristics, a set of benchmark graphs arising from different applications and problems has recently been constructed in conjunction with the 1993 DIMACS challenge on cliques, coloring and satisfiability [189]. These include graphs arising from coding theory [159], artificially generated graphs with known clique size, graphs in which the expected clique number is much smaller than the actual one [72], etc. These data are available at the following WWW address:

<http://dimacs.rutgers.edu/Challenges/index.html>

along with additional useful material.

6.1 Sequential Greedy Heuristics

Many approximation algorithms in the literature for the maximum clique problem are called *sequential greedy heuristics*. These heuristics generate a maximal clique through the repeated addition of a vertex into a partial clique, or the repeated deletion of a vertex from a set that is not a clique. Kopf and Ruhe [205] named these two classes of heuristics the *Best in* and the *Worst out* heuristics. Decisions on which vertex to be *added in* or *moved out* next are based on certain indicators associated with candidate vertices. For example, a possible *Best in* heuristic constructs a maximal clique by repeatedly *adding in* a vertex that has the largest degree among candidate vertices. In this case, the indicator is the degree of a vertex. On the other hand, a possible *Worst out* heuristic can start with the whole vertex set V . It will repeatedly remove a vertex out of V until V becomes a clique.

Kopf and Ruhe [205] further divided the above two classes of heuristics into *New* and *Old* (*Best in or Worst out*) heuristics. Namely, if the indicators are updated every time a vertex is added in or moved out, then the heuristic is called a *New* heuristic. Otherwise it is called an *Old* heuristic. We can find in the literature that many heuristics for the maximum clique problem fall in one or the other classes. See for example, the approximation algorithm of Johnson [188], and the approximation algorithm of Tomita et al. [308]. The differences among these heuristics are their choice of indicators and how indicators are updated. A heuristic of this type can run very fast.

6.2 Local Search Heuristics

A common feature of the sequential heuristics just described is that they all find only one maximal clique. Once a maximal clique is found, the search stops. We can view this type of heuristics from a different point of view. Let us define \mathcal{S}_G to be the system consisting of all the maximal cliques of G . What a sequential greedy heuristic does is to find one set in \mathcal{S}_G , hoping it is (close to) the optimal set. This suggests us a possible way to improve our approximation solutions, namely, expand the search in \mathcal{S}_G . For example, once we find a set $S \in \mathcal{S}_G$, we can search its neighbors to improve S . This leads to the class of the *local search heuristics* [2].

In a local search heuristic, if we search more neighbors of $S \in \mathcal{S}_G$, we increase the chance of finding a better solution. Depending on the neighborhood definition of a set $S \in \mathcal{S}_G$, and how the search is performed, different local search heuristics result. A well known class of local search heuristics in the literature is the *k-interchange* heuristics. They are based on the *k-neighbor* of a feasible solution. In the case of the maximum clique problem, a *k*-neighbor of $S \in \mathcal{S}_G$ is defined as follows. A set $C \in \mathcal{S}_G$ is a *k*-neighbor of S if $|C \Delta S| \leq k$, where $k \leq |S|$. A *k*-interchange heuristic first finds a maximal clique $S \in \mathcal{S}_G$. Then it searches all the *k*-neighbors of S and outputs the best clique found. As one will expect, the main factors for the complexity of this class of heuristics are the size of the neighborhood and the searches involved. For example, in the *k*-interchange heuristic, the complexity grows roughly with $O(n^k)$.

The solution quality of a local search heuristic directly depends on the starting set $S \in \mathcal{S}_G$ and the neighborhood of S . To improve the quality of its solution, we need to increase the neighborhood of S (the starting set) to include a “better” set. If we want to look at various sets spread over \mathcal{S}_G , we need to have a very large neighborhood. The problem is when the size of

the neighborhood increases, the search effort increases so rapidly that one could not afford it.

A class of heuristics designed to search various sets of \mathcal{S}_G is called the *randomized heuristics*. The main ingredient of this class of heuristics is the part that finds a random set in \mathcal{S}_G . A possible way to do that is to include some random factors in the generation of a set of \mathcal{S}_G . A randomized heuristic runs a heuristic (with random factors included) a number of times to find different sets over \mathcal{S}_G . For example, we can randomize a sequential greedy heuristic and let it run N times. The complexity of a randomized heuristic depends on the complexity of the heuristic and the number N .

An elaborated implementation of the randomized heuristic for the maximum independent set problem can be found in Feo et al. [115] where local search is combined with randomized heuristic. Their computational results indicated that their approach was effective in finding large cliques of randomly generated graphs. For example, for randomly generated graphs with 1000 vertices and 50% density, their approach found cliques of size 15 or larger in most cases. Here, 15 is a bound derived from the probabilistic analysis of this class of graphs [53, 55, 122]. A different implementation of a randomized algorithm for the maximum independent set problem can be found in [9].

6.3 Advanced Search Heuristics

Local search algorithms are only capable of finding *local* solutions of an optimization problem. In the past few years, many powerful variations of the basic local search procedure have been developed which try to avoid this problem, many of which are inspired from various phenomena occurring in nature. Examples of such algorithms are simulated annealing, neural networks, genetic algorithms and DNA computing. Because of its importance it is not surprising that these techniques have been applied to the maximum clique problem.

6.3.1 Simulated annealing

In condensed-matter physics, the term “annealing” refers to a physical process to obtain a pure lattice structure, where a solid is first heated up in a heat bath until it melts, and next cooled down slowly until it solidifies into a low-energy state. During the process, the free energy of the system is minimized. Simulated annealing, introduced in 1983 by Kirkpatrick, Gelatt

and Vecchi [201], is a randomized neighborhood search algorithm based on the physical annealing process. Here, the solutions of a combinatorial optimization problem correspond to the states of the physical system, and the cost of a solution is equivalent to the energy of the state.

In its original formulation, simulated annealing works essentially as follows. Initially, a tentative solution in the state space is somehow generated. A new neighboring state is then produced from the previous one and, if the value of the cost function f improves, the new state is accepted, otherwise it is accepted with probability $\exp\{\Delta f/\tau\}$, where Δf is the difference of the cost function between the new and the current state, and τ is a parameter usually called the *temperature* in analogy with physical annealing, which is varied carefully during the optimization process. The algorithm proceeds iteratively this way until a stopping condition is met. One of the critical aspects of the algorithm relates to the choice of the proper “cooling schedule,” i.e., how to decrease the temperature as the process evolves. While a logarithmically slow cooling schedule (yielding an exponential time algorithm) provably guarantees the exact solution, faster cooling schedules, producing acceptably good results, are in widespread use. Introductory textbooks describing both theoretical and practical issues of the algorithm are [208] and [1].

Aarts and Korst [1], without presenting any experimental result, suggested the use of simulated annealing for solving the independent set problem, using a *penalty function* approach. Here, the solution space is the set of all possible subsets of vertices of the graph G , and the problem is formulated as one of maximizing the cost function $f(V') = |V'| - \lambda|E'|$, where $|E'|$ is the number of edges in $G(V')$, and λ is a weighting factor exceeding 1.

Jerrum [187] conducted a theoretical analysis of the performance of a clique-finding *Metropolis* process, i.e., simulated annealing at fixed temperature, on random graphs. He proved that the expected time for the algorithm to find a clique that is only slightly bigger than that produced by a naive greedy heuristic, grows faster than any polynomial in the number of vertices. This suggests that “true” simulated annealing would be ineffective for the maximum clique problem.

Jerrum’s conclusion seems to be contradicted by practical experience. In [173], Homer and Peinado compare the performance of three heuristics, namely the greedy heuristic developed by Johnson [188], a randomized version of Boppana and Halldórsson’s subgraph-exclusion algorithm [66], and simulated annealing, over very large graphs. The simulated annealing algorithm was essentially that proposed by Aarts and Korst, with a simple

cooling schedule. This penalty function approach was found to work better than the method in which only cliques are considered, as proposed by Jerrum [187]. The algorithms were tested on various random graphs as well as on DIMACS benchmark graphs. The authors ran the algorithms over an SGI workstation for graphs with up to 10,000 vertices, and on a Connection Machine for graphs with up to 70,000 vertices. The overall conclusion was that simulated annealing outperforms the other competing algorithms; it also ranked among the best heuristics for maximum clique presented at the 1993 DIMACS challenge.

6.3.2 Neural networks

Artificial neural networks (often simply referred to as “neural networks”) are massively parallel, distributed systems inspired by the anatomy and physiology of the cerebral cortex, which exhibit a number of useful properties such as learning and adaptation, universal approximation, and pattern recognition (see [167], [162] for an introduction).

In the mid-1980’s Hopfield and Tank [174] showed that certain feedback continuous neural models are capable of finding approximate solutions to difficult optimization problems such as the traveling salesman problem [174]. This application was motivated by the property that the temporal evolution of these models is governed by a quadratic Lyapunov function (typically called “energy function” because of its analogy with physical systems) which is iteratively minimized as the process evolves. Since then, a variety of combinatorial optimization problems have been tackled within this framework. The customary approach is to formulate the original problem as one of energy minimization, and then to use a proper relaxation network to find minimizers of this function. Almost invariably, the algorithms developed so far incorporate techniques borrowed from statistical mechanics, in particular mean field theory, which allow one to escape from poor local solutions. We mention the articles [213, 277] and the textbook of Takefuji [302] for surveys of this field, and a journal special issue [184] for recent advances. In [1], an excellent introduction to a particular class of neural networks (the Boltzmann machine) for combinatorial optimization is provided.

Early attempts at encoding the maximum clique and related problems in terms of a neural network were already done in the late 1980’s by Ballard et al. [34], Godbeer et al. [139], Ramanujam and Sadayappan [284], Aarts and Korst [1], and Shrivastava et al. [294] (see also [295]). However, little or no experimental results were presented, thereby making it difficult to evaluate

the merits of these algorithms. In [211], Lin and Lee used the quadratic zero-one formulation from [264] as the basis for their neural network heuristic. On random graphs with up to 300 vertices, they found their algorithm to be faster than the implicit enumerative algorithm in [80], while obtaining slightly worse results in terms of clique size.

Grossman [148] proposed a discrete, deterministic version of the Hopfield model for maximum clique, originally designed for an all-optical implementation. The model has a threshold parameter which determines the character of the stable states of the network. The author suggests an annealing strategy on this parameter, and an adaptive procedure to choose the network's initial state and threshold. On DIMACS graphs the algorithm performs satisfactorily but it does not compare well with more powerful heuristics such as simulated annealing.

Jagota [183] developed several variations of the Hopfield model, both discrete and continuous, to approximate maximum clique. He evaluated the performance of his algorithms over randomly generated graphs as well as on harder graphs obtained by generating cliques of varying size at random and taking their union. Experiments on graphs coming from the Solomonoff-Levin, or "universal" distribution are also presented in [185]. The best results were obtained using a stochastic steepest descent dynamics and a mean-field annealing algorithm, an efficient deterministic approximation of simulated annealing. These algorithms, however, were also the slowest, and this motivated Jagota et al. [186] to improve their running time. The mean-field annealing heuristic was implemented on a 32-processor Connection Machine, and a two-temperature annealing strategy was used. Additionally, a "reinforcement learning" strategy was developed for the stochastic steepest descent heuristic, to automatically adjust its internal parameters as the process evolves. On various benchmark graphs, all their algorithms obtained significantly larger cliques than other simpler heuristics but ran slightly slower. Compared to more sophisticated heuristics, they obtained significantly smaller cliques on average but were considerably faster.

Other attempts at solving the maximum clique problem using Hopfield-style neural networks can be found in [303], [124], [320], [47]. Pelillo [271] takes a completely different approach to the problem, by exploiting the Motzkin-Straus continuous formulation and the dynamical properties of the so-called relaxation labeling networks. His algorithm is the prototype of the replicator-equation based procedures described in Section 6.4.

6.3.3 Genetic algorithms

Genetic algorithms are parallel search procedures inspired from the mechanisms of evolution in natural systems [172, 141]. In contrast to more traditional optimization techniques, they work on a population of points, which in the genetic algorithm terminology, are called chromosomes or individuals. In the simplest and most popular implementation, chromosomes are simply long strings of bits. Each individual has an associated “fitness” value which determines its probability of survival in the next “generation:” the higher the fitness, the higher the probability of survival. The genetic algorithm starts out with an initial population of members generally chosen at random and, in its simplest version, makes use of three basic operators: reproduction, crossover and mutation. Reproduction usually consists of choosing the chromosomes to be copied in the next generation according to a probability proportional to their fitness. After reproduction, the crossover operator is applied between pairs of selected individuals to produce new offsprings. The operator consists of swapping two or more sub-segments of the strings corresponding to the two chosen individuals. Finally, the mutation operator is applied, which randomly reverses the value of every bit within a chromosome with a fixed probability. The procedure just described is sometimes referred to as the “simple” genetic algorithm [141].

One of the earliest attempts to solve the maximum clique problem using genetic algorithms was done in 1993 by Carter and Park [82]. After showing the weakness of the simple genetic algorithm in finding large cliques, even on small random graphs, they introduced several modifications in an attempt to improve performance. However, despite their efforts they did not get satisfactory results, and their general conclusion was that genetic algorithms need to be heavily customized in order to be competitive with traditional approaches, and that they are computationally very expensive. In a later study [267], genetic algorithms were proven to be less effective than simulated annealing. At almost the same time Bäck and Khuri [22], working on the maximum independent set problem, arrived at the opposite conclusion. By using a straightforward, general-purpose genetic algorithm called GENEsYs and a suitable fitness function which included a graded penalty term to penalize infeasible solutions, they got interesting results over random and regular graphs with up to 200 vertices. These results indicate that the choice of the fitness function is crucial for genetic algorithms to provide satisfactory results.

Murthy et al. [241] also experimented with a genetic algorithm using a

novel “partial copy crossover,” and a modified mutation operator. However, they presented results over very small (i.e., up to 50 vertices) graphs, thereby making it difficult to properly evaluate the algorithm.

Bui and Eppley [77] obtained encouraging results by using a hybrid strategy which incorporates a local optimization step at each generation of the genetic algorithm, and a vertex-ordering preprocessing phase. They tested the algorithm over some DIMACS graphs getting results comparable to that in [134] (cf. Section 6.4).

Instead of using the standard binary representation for chromosomes, Foster and Soule [118] employed an integer-based encoding scheme. Moreover, they used a time weighting fitness function similar in spirit to those of Carter and Park [82]. The results obtained are interesting, but still not comparable to those obtained using more traditional search heuristics.

Fleurent and Ferland [117] developed a general-purpose system for solving graph coloring, maximum clique, and satisfiability problems. As far as the maximum clique problem is concerned, they conducted several experiments using a hybrid genetic search scheme which incorporates tabu search (described in Section 6.3.4) and other local search techniques as alternative mutation operators. The results presented are encouraging, but running time is quite high.

In [169], Hifi modifies the basic genetic algorithm in several aspects: (a) a particular crossover operator creates two new different children; (b) the mutation operator is replaced by a specific heuristic feasibility transition adapted to the weighted maximum stable set problem. This approach is also easily parallelizable. Experimental results on randomly generated graphs and also some (unweighted) instances from the DIMACS testbed [189] are reported to validate this approach.

Finally, Marchiori [225] has recently developed a simple heuristic-based genetic algorithm which consists of a combination of the simple genetic algorithm and a naive greedy heuristic procedure. Unlike previous approaches, here there is a neat division of labour, the search for a large subgraph and the search for a clique being incorporated into the fitness function and the heuristic procedure, respectively. The algorithm outperforms previous genetic-based clique finding procedures over various DIMACS graphs, both in terms of quality of solutions and speed.

We note that the GENF procedure proposed in [64] does not fall into the category of genetic algorithms, despite similarity in nomenclature. Both approaches share the modeling idea that the objective is interpreted as fitness and the decision variables could be interpreted as characteristics of a

population subject to selection. The most important difference is that the (random) mutation prevailing in genetic algorithms has no counterpart in the GENF algorithm, where escaping from inefficient local solutions is done deliberately and deterministically (similar to real-world genetic engineering).

6.3.4 Tabu search

Tabu search, introduced independently by Glover [136], [137] and Hansen and Jaumard [156], is a modified local search algorithm, in which a prohibition-based strategy is employed to avoid cycles in the search trajectories and to explore new regions in the search space. At each step of the algorithm, the next solution visited is always chosen to be the best *legal* neighbor of the current state, even if its cost is worse than the current solution. The set of legal neighbors is restricted by one or more *tabu lists* which prevent the algorithm to go back to recently visited solutions. These lists are used to store historical information on the path followed by the search procedure. Sometimes the tabu restriction is relaxed, and tabu solutions are accepted if they satisfy some *aspiration level* condition. The standard example of a tabu list is one which contains the last k solutions examined, where k may be fixed or variable. Additional lists containing the last modifications performed, i.e., changes occurred when moving from one solution to the next, are also common. These types of lists are referred to as *short-term memories*; other forms of memories are also used to *intensify* the search in a promising region or to *diversify* the search to unexplored areas. Details on the algorithm and its variants can be found in [138] and [168].

In 1989, Friden et al. [120] proposed a heuristic for the maximum independent set problem based on tabu search. The size of the independent set to search for is fixed, and the algorithm tries to minimize the number of edges in the current subset of vertices. They used three tabu lists: one for storing the last visited solutions and the other two to contain the last introduced/deleted vertices. They showed that by using hashing for implementing the first list and choosing a small value for the dimensions of the other two lists, a best neighbor may be found in almost constant time. The tabu-search-based branch and bound algorithm presented by the same authors in [121] has already been mentioned in Section 5.

In [131, 298], three variants of tabu search for maximum clique are presented. Here the search space consists of complete subgraphs whose size has to be maximized. The first two versions are deterministic algorithms in which no sampling of the neighborhood is performed. The main difference

between the two algorithms is that the first one uses just one tabu list (of the last solutions visited), while the second one uses an additional list (with an associated aspiration mechanism) containing the last vertices deleted. Also, two diversification strategies were implemented. The third algorithm is probabilistic in nature, and uses the same two tabu lists and aspiration mechanism as the second one. The differences lie in a random sampling of the neighborhood, and also in the possibility of multiple vertex deletion in the current solution. Here no diversification strategy was used. In [131, 298] results on randomly generated graphs were presented and the algorithms were shown to be very effective. More recently, Soriano and Gendreau [297] tested their algorithms over the DIMACS benchmark graphs and the results confirmed the early conclusions.

Recently, Battiti and Protasi [37] extended the tabu search framework by introducing a reactive local search method. They modified a previously introduced reactive scheme by exploiting the particular neighborhood structure of the maximum clique problem. In general, reactive schemes aim at avoiding the manual selection of control parameters by means of an internal feedback loop. Battiti and Protasi's algorithm adopts such a strategy to automatically determine the so-called prohibition parameter k , i.e., the size of the tabu list. Also an explicit memory-influenced restart procedure is activated periodically to introduce diversification. The search space consists of all possible cliques, as in Friden et al.'s approach, and the function to be maximized is the clique size. The worst case computational complexity of this algorithm is $O(\max\{n, m\})$ where n and m are the number of nodes and edges of the graph respectively. They noticed, however, that in practice, the number of operations tends to be proportional to the average degree of the nodes of the complement graph. They tested their algorithm over many DIMACS benchmark graphs obtaining better results than those presented at the DIMACS workshop in competitive time.

6.4 Continuous-based Heuristics

Recently, there has been much interest around the Motzkin-Straus and related continuous formulations of the maximum clique problem (see Section 2). They suggest in fact a fundamentally new way of solving the maximum clique problem, by allowing us to shift from the discrete to the continuous domain in an elegant manner. As recently pointed out [259], continuous formulations of discrete optimization problems turn out to be particularly attractive. They not only allow us to exploit the full arsenal of

continuous optimization techniques, thereby leading to the development of new algorithms, but may also reveal unexpected theoretical properties.

In [261], Pardalos and Phillips developed a global optimization approach based on the Motzkin-Straus formulation and implemented an iterative clique retrieval process to find the vertices of the maximum clique. However, due to its high computational cost they were not able to run the algorithm over graphs with more than 75 vertices. More recently, Pelillo [271] used *relaxation labeling algorithms* to approximately determine the size of the maximum clique using the original Motzkin-Straus formulation. These are parallel, distributed algorithms developed and studied in computer vision and pattern recognition, which are also surprisingly related to *replicator equations*, a class of dynamical systems widely studied in evolutionary game theory and related fields [171]. The model operates in the simplex Δ and possesses a quadratic Lyapunov function which drives its dynamical behavior. It is these properties that naturally suggest using them as a local optimization algorithm for the Motzkin-Straus program. The model is especially suited for parallel implementation, and is attractive for its operational simplicity, since no parameters need to be determined. Extensive simulations over random graphs with up to 2000 vertices have demonstrated the effectiveness of the approach and showed that the algorithm outperforms previous neural network heuristics.

In order to avoid time-consuming iterative procedures to extract the vertices of the clique, Gibbons, Hearn and Pardalos [134] have proposed a heuristic which is based on a parametrized formulation of the Motzkin-Straus program. They consider the problem of minimizing the function

$$h(x) = \frac{1}{2}x^T A_{\bar{G}}x + \left(\sum_{i=1}^n x_i - 1 \right)^2$$

on the domain:

$$S(k) = \left\{ x \in \mathbb{R}^n : \sum_{i=1}^n x_i^2 \leq \frac{1}{k}, \text{ and } x_i \geq 0, i = 1, \dots, n \right\}$$

where k is a fixed parameter. Let x^* be a global minimizer of h on $S(k)$, and let $V(k) = h(x^*)$. In [134] it is proved that $V(k) = 0$ if and only if there exists an independent set S of \bar{G} with size $|S| \geq k$. Moreover, the vertices of \bar{G} associated with the indices of the positive components of x^* form an independent set of size greater than or equal k .

These properties motivated the following procedure to find a maximum independent set of \overline{G} or, equivalently, a maximum clique of G . Minimize the function h over $S(k)$, for different values of k between predetermined upper and lower bounds. If $V(k) = 0$ and $V(k+1) \neq 0$ for some k , then the maximum clique of G has size k , and its vertices are determined by the positive components of the solution. Since minimizing h on $S(k)$ is a difficult problem, Gibbons and coworkers developed a heuristic based on the observation that by removing the nonnegativity constraints, the problem is that of minimizing a quadratic form over a sphere, a problem which is solvable in polynomial-time. However, in so doing a heuristic procedure is needed to round the approximate solutions of this new problem to approximate solutions of the original one. Moreover, since the problem is solved approximately, we have to find the value of the spherical constraint $1/k$ which yields the largest independent set. A careful choice of k is therefore needed. The resulting algorithm was tested over various DIMACS benchmark graphs [189] and the results obtained confirmed the effectiveness of the approach.

In [61], replicator equations are used in conjunction to the spurious-free formulation given in Theorem 2.6 to find maximal cliques of G . Note that here the nodes comprising the clique are directly given by the positive components of the converged vectors, and no iterative procedure is needed to determine them, as in [261]. The results obtained over a set of DIMACS benchmark graphs [189] were encouraging, especially considering that replicator equations do not incorporate any mechanism to escape from local optimal solutions. This suggests that the basins of attraction of the global solution w.r.t. the quadratic functions g and \hat{g} occurring in (9) and Theorem 2.6 are quite large; for a thorough empirical analysis see also [64]. One may wonder whether a subtle choice of initial conditions and/or a variant of the dynamics may significantly improve the results, but experiments in [63] indicate this is not the case. In [59] the properties of the following function are studied

$$\hat{g}_\alpha(x) = x^T A_G x + \alpha x^T x$$

and a heuristic is proposed which is based on the modification of the parameter α in the course of the optimization process.

Finally, Bomze et al. [62] have recently used replicator equations to find maximal weight cliques in weighted graphs, using Theorem 2.8.

6.5 Miscellaneous

Another type of heuristics that finds a maximal clique of G is called the *subgraph approach* (see [31]). It is based on the fact that a maximum clique C of a subgraph $G' \subseteq G$ is also a clique of G . The subgraph approach first finds a subgraph $G' \subseteq G$ such that the maximum clique of G' can be found in polynomial time. Then it finds a maximum clique of G' and use it as the approximation solution. The advantage of this approach is that in finding the maximum clique $C \subseteq G'$, one has (implicitly) searched many other cliques of G' ($S_{G'} \subseteq S_G$). Because of the special structure of G' , this implicit search can be done efficiently. In Balas and Yu [31], G' is a maximal induced triangulated subgraph of G . Since many classes of graphs have polynomial algorithms for the maximum clique problem, the same idea also applies there. For example, the class of edge-maximal triangulated subgraphs was chosen in [23], [321], and [322]. Some of the greedy heuristics, randomized heuristics and subgraph approach heuristics are compared in [321] and [322] on randomly generated weighted and unweighted graphs.

Various new heuristics were presented at the 1993 DIMACS challenge devoted to clique, coloring and satisfiability [189]. In particular, Balas and Niehaus [26] proposed an algorithm which is based on the observation that finding the maximum clique in the union of two cliques can be done using bipartite matching techniques. Goldberg and Rivenbrugh [142] used restricted backtracking to provide a tradeoff between the size of the clique and the completeness of the search. Mannino and Sassano [224] proposed an edge projection technique to obtain a new upper bound heuristic for the maximum independent set problem. This procedure was used, in conjunction with Balas and Yu's branching rule [31], to develop an exact branch and bound algorithm which works well especially on sparse graphs.

Abbattista et al. [3] developed a new population-based optimization heuristic inspired by the natural behavior of human or animal scouts in exploring unknown regions, and applied it to maximum clique. The results obtained over a few DIMACS graphs are comparable with those obtained using continuous-based heuristics but are inferior to those obtained by reactive local search.

Recently, DNA computing [5] has also emerged as a potential technique for the maximum clique problem [251], [325]. The major advantage of DNA computing is its high parallelism, but at present the size of graphs this algorithm can handle is limited to a few tens.

Additional heuristics for the maximum clique/independent set and re-

lated problems on arbitrary or special class of graphs can be found in [89], [91], [94], [116]. Among others, further (partly randomized) parallel algorithms for the (weighted) maximum clique problem are proposed in [195], [220], [101], [143], [144], [81], [263], [6], and [99].

7 Selected Applications

In many applications, the underlying problem can be formulated as a maximum clique problem while in others a subproblem of the solution procedure consists of finding a maximum clique. This necessitates the development of fast exact and approximate algorithms for the problem.

The proliferation of massive data sets brings with it a series of special computational challenges. Many of these data sets can be modeled as very large multidigraphs with a special set of edge attributes that represent special characteristics of the application at hand. Understanding the structure of the underlying digraph is essential for storage organization and information retrieval. In [4] experiments with data from telecommunications traffic, the corresponding multigraph has 53,767,087 vertices and over 170 million of edges. A giant connected component with 44,989,297 vertices was computed. The maximum clique problem is considered in this giant component. Similar computational challenges with very large graphs appear in several other practical applications.

The application areas considered below are diverse. For example, we will present a class of graphs from which we can prove or disprove Keller's conjecture; a famous problem in geometry, a part of which is still open. Another example arises from coding theory where one wishes to find binary codes as large as possible that can correct a prespecified number of errors. The problem can be solved by solving the maximum clique problem in a corresponding graph. We also indicate how the maximum clique problem occurs in fault diagnosis models. A further important application area of the maximum clique problem discussed here is computer vision and pattern recognition.

Other applications can be found, e.g., in [17], [269], [104], [270], [103], [312], [319], [314], [35], [266], [315], [223].

7.1 Coding Theory: Hamming and Johnson Graphs

In this section we will describe how coding theory problems can be interpreted as maximum clique problems on certain graphs. In Coding Theory,

one wishes to find a binary code as large as possible that can correct a certain number of errors for a given size of the binary words (vectors), see [74, 296]. In order to correct errors, the code must consist of binary words among which any two differ in a certain number of positions so that a misspelled word can be detected and corrected. A misspelled word is corrected by replacing it with the word from the code that differs the least from the misspelled one.

The *Hamming distance* between the binary vectors $u = (u_1, u_2, \dots, u_n)$ and $v = (v_1, v_2, \dots, v_n)$ is the number of indices i such that $1 \leq i \leq n$ and $u_i \neq v_i$. We denote the Hamming distance by $dist(u, v)$.

It is well known that a binary code consisting of a set of binary vectors any two of which have Hamming distance greater or equal to d can correct $\lfloor \frac{d-1}{2} \rfloor$ errors [221]. Thus, what a coding theorist would like to find is the maximum number of binary vectors of size n with Hamming distance d . We denote this number by $A(n, d)$.

Another problem arising from Coding Theory, closely related to the one mentioned above, is to find a weighted binary code, that is, to find the maximum number of binary vectors of size n that have precisely w 1's and the Hamming distance of any two of these vectors is d . This number is denoted by $A(n, w, d)$. A binary code consisting of vectors of size n , weight w and distance d , can correct $w - \frac{d}{2}$ errors [221].

Now the *Hamming graph* $H(n, d)$, of size n and distance d , is defined as the graph with vertex set the binary vectors of size n , in which two vertices are adjacent if their Hamming distance is *at least* d . Then, $A(n, d)$ is the size of a maximum clique in $H(n, d)$.

The graph $H(n, d)$ has 2^n vertices, $2^{n-1} \sum_{i=d}^n \binom{n}{i}$ edges and the degree of each vertex is $\sum_{i=d}^n \binom{n}{i}$.

Next we define the Johnson graph, $J(n, w, d)$, with parameters n , w and d , as the graph with vertex set the binary vectors of size n and weight w , where two vertices are adjacent if their Hamming distance is *at least* d . Then, similar to Hamming graph, the size of the weighted code, $A(n, w, d)$, equals the size of the maximum clique in $J(n, w, d)$.

The graph $J(n, w, d)$ has $\binom{n}{w}$ vertices, $\frac{1}{2} \binom{n}{w} \sum_{k=\lceil \frac{d}{2} \rceil}^w \binom{w}{k} \binom{n-w}{k}$ edges and the degree of each vertex is $\sum_{k=\lceil \frac{d}{2} \rceil}^w \binom{w}{k} \binom{n-w}{k}$.

7.2 Geometry of Tiling: Keller's Conjecture

A family of hypercubes with disjoint interiors whose union is the Euclidean space \mathbb{R}^n is a *tiling*. A lattice tiling is a tiling for which the centers of the cubes form a lattice [300].

In the beginning of the century, Minkowski conjectured that in a lattice tiling of \mathbb{R}^n by translates of a unit hypercube, there exist two cubes that share a $(n - 1)$ -dimensional face. About fifty years later, Hajós [154] proved Minkowski's conjecture.

At 1930, Keller [198] suggested that Minkowski's conjecture holds even in the absence of the lattice assumption. Ten years later Perron [276] proved the correctness of Keller's conjecture for $n \leq 6$. Since then, many papers have been devoted to prove or disprove this conjecture [301] and recently, Lagarias and Shor [207] proved that Keller's conjecture fails for $n \geq 10$. Thus, it is left to prove whether the conjecture holds for $n = 7, 8, 9$.

We define the Keller Graph Γ_n as a graph with vertex set

$$V_n = \{(d_1, d_2, \dots, d_n) : d_i \in \{0, 1, 2, 3\}, i = 1, 2, \dots, n\}$$

where two vertices $u = (d_1, d_2, \dots, d_n)$ and $v = (d'_1, d'_2, \dots, d'_n)$ in V_n are adjacent if and only if

$$\exists i, 1 \leq i \leq n : d_i - d'_i \equiv 2 \pmod{4} \quad (29)$$

and

$$\exists j \neq i, 1 \leq j \leq n : d_j \neq d'_j. \quad (30)$$

In [96], Corrádi and Szabó presented a graph theoretic equivalent of Keller's conjecture. It is shown that, there is a counterexample to Keller's conjecture if and only if there exist a $n \in \mathbb{N}^+$ such that Γ_n has a clique of size 2^n .

Γ_n has 4^n vertices, $\frac{1}{2}4^n(4^n - 3^n - n)$ edges and the degree of each node is $4^n - 3^n - n$. Γ_n is very dense and has at least $8^n n!$ different maximum cliques. It can be shown [207] that the maximum clique size of Γ_n is less than or equal to 2^n .

7.3 Problems Arising From Fault Diagnosis

A crucial problem in studying the reliability of large multiprocessor systems, is the problem known as system-level fault diagnosis. The task is to identify all faulty processors (units) in the system. The classical approach to fault

diagnosis was originated over thirty years ago by Preparata, Metze and Chien [280], leading to a fault diagnosis model known as the PMC model.

In the PMC model each unit can test some other units and it is assumed that fault-free units always give the correct results while faulty ones are unpredictable and can output any results. Furthermore, it is assumed that the number of faulty units never exceed some upper bound t . Upon completion of all tests, the results are gathered by a monitoring unit which computes the status of all units based on the gathered results.

The assumption that a fault-free unit always detects faulty units may seem a little optimistic. Also, the upper bound assumption may restrict the model to unrealistic situations. Further, the PMC model is accurate only if the upper bound t does not exceed the number of neighbors of any unit. For large systems however, the connectivity might be fairly low, making it quite probable that the number of faulty units exceed the number of neighbors for some units.

Yet another assumption in the PMC model, the existence of a central monitoring unit, makes it less reliable. In order to overcome this problem, distributed fault-tolerance was introduced. The goal of this approach is to find a way to let every fault-free unit to be able to determine the status of every other unit.

The above observations have led to several different models one of which was introduced by Blough [50]. In his model, processors test each other and fault-free units always detect other fault-free processors correctly, while they detect faulty processors with a fixed probability less than 1. No assumption is made about how faulty units behave as testers.

In [44], Berman and Pelc study a realistic approach to fault diagnosis by simultaneously relaxing all the three assumptions from the PMC model described above. Their model is based on a probabilistic model presented by Blough performed in a distributed fashion. Consequently, a processor can never be sure that the information it receives is correct. Berman and Pelc define a system design represented by a class of graphs, G_n . They show that the probability of correct diagnosis of fault processors for such systems, happens with probability at least $1 - n^{-1}$. The algorithm they propose is based on a model where a test by a fault-free unit on a faulty one does not detect a fault with probability q , while they assume that fault-free units never detect faults in each other.

For a given parameter c , a *c-fat ring* is the graph $G = (V, E)$ defined as

follows. Let

$$k = \lfloor \frac{|V|}{c \log |V|} \rfloor$$

and let W_0, \dots, W_{k-1} be a partition of V such that

$$c \log |V| \leq |W_i| \leq 1 + \lceil c \log |V| \rceil \text{ for } i = 0, 1, \dots, k-1. \quad (31)$$

For $u \in W_i$ and $v \in W_j$ we have $(u, v) \in E$ if and only if $u \neq v$ and $|i - j| \in \{0, 1, k - 1\}$.

A major step in the algorithm proposed in [44] is to find the maximum clique of a c -fat ring. Therefore Hasselberg et al. [159] construct a c -fat ring generator and perform some computations to see how the maximum clique algorithms perform on such graphs.

7.4 Computer Vision and Pattern Recognition

Many fundamental problems in computer vision and pattern recognition can be formulated as the problem of matching *relational structures* [33]. In the computer vision terminology, a relational structure is a triple $S = (U, \mathcal{P}, \mathcal{R})$, where U is a set of units, $\mathcal{P} = \{P_1, \dots, P_l\}$ is a set of properties, and $\mathcal{R} = \{R_1, \dots, R_k\}$ is a set of (binary) relations over the units. Generalizations involving higher-order relations are also common, but for the sake of simplicity we shall only consider the binary case here. Note that the notion of a relational structure is essentially equivalent to that of a *pseudograph* employed in graph theory [157]. A relational structure becomes a graph, in the traditional sense, when the relation set \mathcal{R} contains a single relation, and the property set \mathcal{P} is empty. The relation may be symmetric, in which case we obtain an undirected graph.

Consider two relational structures $S' = (U', \mathcal{P}', \mathcal{R}')$ and $S'' = (U'', \mathcal{P}'', \mathcal{R}'')$. A pair of units (u', u'') , one from S' and the other from S'' , is said to be *good* if all properties that hold for u' hold for u'' as well, and vice versa, that is if

$$P'_i(u') \Leftrightarrow P''_i(u'')$$

for all $i = 1, \dots, l$, where $P'_i \in \mathcal{P}'$ and $P''_i \in \mathcal{P}''$. Similarly, two good pairs (u', u'') and (v', v'') , with $u' \neq v'$ and $u'' \neq v''$, are said to be *compatible* if

$$R'_j(u', v') \Leftrightarrow R''_j(u'', v'') \text{ and } R'_j(v', u') \Leftrightarrow R''_j(v'', u'')$$

for all $j = 1, \dots, k$, where $R'_j \in \mathcal{R}'$ and $R''_j \in \mathcal{R}''$. A *match* between S' and S'' is any relation $\mu \subseteq U' \times U''$ such that all its assignments are good and

mutually compatible. A match is *maximal* if it is not included in any other match, and is *maximum* if it has largest cardinality. The relational structure matching problem is just the problem of finding a maximum match between two relational structures. When the relational structures being matched are graphs the problem becomes the maximum common subgraph problem, which is known to be *NP*-complete [126]. Obviously, for this reason, the relational structure matching problem too is *NP*-complete.

Ambler et al. [11] (see also [36] and [206]) introduced the notion of *association graph* as an auxiliary structure for matching relational structures. The association graph of two relational structures S' and S'' is the undirected graph $G = (V, E)$ defined as

$$V = \{(u', u'') \in U' \times U'' : (u', u'') \text{ is good}\}$$

and

$$E = \{((u', u''), (v', v'')) \in V \times V : (u', u'') \text{ and } (v', v'') \text{ are compatible}\}.$$

It is clear that, given the way we have constructed the association graph, the notions of match, maximal match, and maximum match turn out to coincide with those of clique, maximal clique, and maximum clique of the association graph, respectively. The problem of matching two relational structures is therefore equivalent to the maximum clique problem.

In many practical applications, properties as well as relations may be assigned one or more numerical attributes. For example, if units represents segmented regions in an image, the property "circular" can be associated with the diameter of the circle; or, the relation which establishes that two regions are adjacent can be assigned a numerical value specifying the (relative) amount of common boundary. It is straightforward to generalize the association graph idea described above to the case of *attributed* relational structures. To this end, we simply need to specify some similarity measure between attribute vectors; the topology of the association graph is then defined according to the degree of similarity between corresponding property's and relation's attributes. An alternative approach is to construct a *weighted* association graph (see, e.g., [175]). In this case, a pair is declared good using the same criterion as in the unattributed case, and the corresponding vertex in the association graph is assigned a positive weight which represents the (overall) similarity between the attribute vectors. In this case, one is interested in determining a clique having largest total weight and this is the maximum weight clique problem.

In [273] and [272], Pelillo uses the association graph framework and the Motzkin-Straus formulation (see Section 2.2) to develop a new quadratic programming formulation for graph isomorphism and related relational structure matching problems. Replicator equations, mentioned in Section 6.4, are used to solve the program and the experimental results obtained show how on the graph isomorphism problem the replicator dynamical system outperforms more sophisticated heuristics based on mean-field theory.

Computer vision and pattern recognition problems for which the maximum clique formulation has proven to be effective include, for example, stereo correspondence [175], object recognition [51], [52], [155], point pattern matching [247], and motion analysis [282], [279]. The framework has recently been extended to handle the problem of matching hierarchical relational structures, such as trees, and applied to the problem of shape matching [275].

8 Conclusions

Over the past four decades, research on the maximum clique and related problems has yielded many interesting and profound results. However, a great deal remains to be learned about the maximum clique problem.

This paper provides an expository survey on complexity, algorithms and applications of the maximum clique problem. Furthermore, an extensive up-to-date bibliography is included. However, the present activity in work related to the maximum clique problem is so extensive that a survey of this nature is outdated before it is written.

Acknowledgments

The authors would like to thank A. Panconesi for reading an early version of this chapter, and providing useful comments and suggestions, as well as M. Dür for valuable hints.

References

- [1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, J. Wiley & Sons, Chichester, UK, 1989.

- [2] E. Aarts and J.K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, J. Wiley & Sons, Chichester, UK, 1997.
- [3] F. Abbattista, F. Bellifemmine and D. Dalbis, The Scout algorithm applied to the maximum clique problem, in *Advances in Soft Computing—Engineering and Design*, Springer, Berlin, 1998.
- [4] J. Abello, P.M. Pardalos and M.G.C. Resende, On maximum clique problems in very large graphs, in *External Memory Algorithms, DIMACS Series, AMS* (J. Abello and J. Vitter, Editors) 1999.
- [5] L.M. Adleman, Molecular computation of solutions to combinatorial optimization, *Science*, Vol. 266: 1021–1024, 1994.
- [6] F. Alizadeh, A sublinear-time randomized parallel algorithm for the maximum clique problem in perfect graphs, in: A. Aggarwal (ed.), *Discrete Algorithms, Proc. 2nd ACM-SIAM Symposium*, Vol. 2: 188–194, 1991.
- [7] N. Alon, U. Feige, A. Wigderson and D. Zuckerman, Derandomized graph products, *Comput. Complex.*, Vol. 5: 60–75, 1995.
- [8] E.A. Akkoyunlu, The enumeration of maximal cliques of large graphs, *SIAM J. Comput.*, Vol. 2: 1–6, 1973.
- [9] N. Alon, L. Babai and A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, *J. Algorithms*, Vol. 7: 567–583, 1986.
- [10] N. Alon, M. Krivelevich and B. Sudakov, Finding a large hidden clique in a random graph, in: *Proc. SODA '98—Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco CA*, January 25–27, 1998.
- [11] A.P. Ambler, H.G. Barrow, C.M. Brown, R.M. Burstall and R.J. Popplestone, A versatile computer-controlled assembly system, in *Proc. 3rd Int. J. Conf. Artif. Intell.*: 298–307, 1973.
- [12] A.T. Amin and S.L. Hakimi, Upper bounds on the order of a clique of a graph, *SIAM J. Appl. Math.*, Vol. 22: 569–573, 1972.
- [13] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and hardness of approximation problems, *Proc. 33rd Ann. Symp. Found. Computer Sci.*: 14–23, 1992.

- [14] S. Arora and S. Safra, Probabilistic checking of proofs: A new characterization of NP, *Proc. 33rd Ann. Symp. Found. Computer Sci.*: 2–13, 1992.
- [15] J.G. Auguston and J. Minker, An analysis of some graph theoretical cluster techniques, *J. ACM*, Vol. 17: 571–588, 1970.
- [16] G. Ausiello, P. Crescenzi and M. Protasi, Approximation solution of NP optimization problems, *Theor. Comput. Sci.*, Vol. 150: 1–55, 1995.
- [17] G. Avondo-Bodeno, *Economic Applications of the Theory of Graphs*, Gordon and Breach Science Publishers, New York, 1962.
- [18] S.M. Baas, M.C. Bonvanie and A.J. Verschoor, A relaxation method for the set packing problem using polyhedron characteristic, *Technical Report 699*, University of Twente, Netherlands, 1988.
- [19] L. Babel, Finding maximum cliques in arbitrary and in special graphs, *Computing*, Vol. 46: 321–341, 1991.
- [20] L. Babel, A fast algorithm for the maximum weight clique problem, *Computing*, Vol. 52: 31–38, 1994.
- [21] L. Babel and G. Tinhofer, A branch and bound algorithm for the maximum clique problem, *ZOR-Methods and Models of Operations Research*, Vol. 34: 207–217, 1990.
- [22] T. Bäck and S. Khuri, An evolutionary heuristic for the maximum independent set problem, *Proc. 1st IEEE Conf. Evolutionary Comput.*: 531–535, 1994.
- [23] E. Balas, A fast algorithm for finding an edge-maximal subgraph with a TR-formative coloring, *Discr. Appl. Math.*, Vol. 15: 123–134, 1986.
- [24] E. Balas, S. Ceria, G. Corneujoles and G. Pataki, Polyhedral methods for the maximum clique problem, in [189]: 11–28, 1996.
- [25] E. Balas, V. Chvátal and J. Nešetřil, On the maximum weight clique problem, *Math. Oper. Res.*, Vol. 12: 522–535, 1987.
- [26] E. Balas and W. Niehaus, Finding large cliques in arbitrary graphs by bipartite matching, in [189]: 29–51, 1996.

- [27] E. Balas and H. Samuelsson, A Node Covering Algorithm, *Naval Research Logistics Quarterly*, Vol. 24: 213–233, 1977.
- [28] E. Balas and P. Toth, Branch and bound methods, In: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (Eds.), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, J. Wiley & Sons, Chichester, UK: 361–403, 1985.
- [29] E. Balas and J. Xue, Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs, *SIAM J. Comput.*, Vol. 2: 209–221, 1991. “Addendum,” *SIAM J. Comput.*, Vol. 21: 1000, 1992.
- [30] E. Balas and J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, *Algorithmica* Vol. 15: 397–412, 1996.
- [31] E. Balas and C.S. Yu, Finding a maximum clique in an arbitrary graph, *SIAM J. Comput.*, Vol. 14: 1054–1068, 1986.
- [32] E. Balas and C.S. Yu, On graphs with polynomially solvable maximum-weight clique problem, *Networks*, Vol. 19: 247–253, 1989.
- [33] D.H. Ballard and M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, N.J., 1982.
- [34] D.H. Ballard, P.C. Gardner and M.A. Srinivas, Graph problems and connectionist architectures, *Technical Report TR 167*, Dept. Computer Science, University of Rochester, 1987
- [35] F. Barahona, A. Weintraub, and R. Epstein, Habitat dispersion in forest planning and the stable set problem, *Oper. Res.* Vol. 40, Supp. 1: S14–S21, 1992.
- [36] H.G. Barrow and R.M. Burstall, Subgraph isomorphism, matching relational structures and maximal cliques, *Inform. Proc. Lett.*, Vol. 4: 83–84, 1976.
- [37] R. Battiti and M. Protasi, Reactive local search for the maximum clique problem, *Technical Report TR-95-052*, International Computer Science Institute, Berkeley, CA, 1995.

- [38] A.R. Bednarek and O.E. Taulbee, On maximal chains, *Roum. Math. Pres et Appl.*, Vol. 11: 23–25, 1966.
- [39] L.W. Beineke, A survey of packings and coverings in graphs, in: G. Chartrand and S. Kapoor (Eds.), *Many Facets of Graph Theory*, Springer: 45–53, Berlin, 1969.
- [40] M. Bellare, O. Goldreich, C. Lund and A. Russell, Efficient probabilistically checkable proofs and application to approximation, *Proc. 25th Ann. ACM Symp. Theory of Comput.*: 294–304, 1993.
- [41] M. Bellare, O. Goldreich and M. Sudan, Free bits, PCPs and non-approximability—Towards tight results, *SIAM J. Comput.* Vol. 27: 804–915, 1997.
- [42] M. Bellare and M. Sudan, Improved nonapproximability results, *Proc. 26th Ann. ACM Symp. Theory of Comput.*: 184–193, 1994.
- [43] C. Berge and V. Chvátal (Eds.), Topics on Perfect Graphs, *Ann. Discr. Math.*, Vol. 21, 1984.
- [44] P. Berman and A. Pelc, Distributed fault diagnosis for multiprocessor systems, *Proc. of the 20th Annual Int. Symp. on Fault-Tolerant Computing* (Newcastle, UK): 340–346, 1990.
- [45] P. Berman and G. Schnitger, On the complexity of approximating the independent set problem, in *Proc. 1989 Symposium on Theoret. Aspects of Comp. Sci.*, Springer, Lecture Notes in Computer Sciences Vol. 349: 256–268, Berlin, 1989.
- [46] P. Berman and G. Schnitger, On the complexity of approximating the independent set problem, *Inform. and Comput.*, Vol. 96: 77–94, 1992.
- [47] A. Bertoni, P. Campadelli and G. Grossi, A discrete neural algorithm for the maximum clique problem: Analysis and circuit implementation, presented at *WAE'97: Int. Workshop on Algorithm Engineering*, Venice, Italy, 1997.
- [48] B.K. Bhattacharya, P. Hell and J. Huang, A linear algorithm for maximum weight cliques in proper circular arc graphs, *SIAM J. Discr. Math.* Vol. 9: 274–289, 1996.

- [49] B.K. Bhattacharya and D. Kaller, An $O(m+n \log n)$ algorithm for the maximum clique problem in circular-arc graphs, *J. Algorithms* Vol. 25: 33–358, 1997.
- [50] D.M. Blough, Fault detection and diagnosis in multiprocessor systems, *Ph.D. Thesis, The Johns Hopkins University*, 1988.
- [51] R.C. Bolles and R.A. Cain, Recognizing and locating partially visible objects: The locus-feature-focus method, *Int. J. Robotics Res.*, Vol. 1: 57–82, 1982.
- [52] R.C. Bolles and P. Horaud, 3DPO: A three-dimensional part orientation system, *Int. J. Robotics Res.*, Vol. 5: 3–26, 1986.
- [53] B. Bollobás, *Random graphs*, Academic Press, New York, NY, 1985.
- [54] B. Bollobás, *Modern graph theory*, Springer, New York, NY, 1998.
- [55] B. Bollobás and P. Erdős, Cliques in Random Graphs, *Math. Proc. Cambridge Philos. Soc.*, Vol. 80: 419–427, 1976.
- [56] I.M. Bomze, Evolution towards the maximum clique, *J. Glob. Optim.*, Vol. 10: 143–164, 1997.
- [57] I.M. Bomze, Global escape strategies for maximizing quadratic forms over a simplex, *J. Global Optim.* Vol. 11: 325–338, 1997.
- [58] I.M. Bomze, On standard quadratic optimization problems, *J. Glob. Optim.* Vol. 13: 369–387, 1998.
- [59] I.M. Bomze, M. Budinich, M. Pelillo and C. Rossi, Annealed replication: A new heuristic for the maximum clique problem, submitted for publication, 1998.
- [60] I.M. Bomze and G. Danner, A finite algorithm for solving general quadratic problems, *J. Global Optim.* Vol. 4: 1–16, 1994.
- [61] I.M. Bomze, M. Pelillo, and R. Giacomini, Evolutionary approach to the maximum clique problem: Empirical evidence on a larger scale, in *Developments in Global Optimization*, I.M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos (eds.), Kluwer: 95–108, Dordrecht, 1997.

- [62] I.M. Bomze, M. Pelillo, and V. Stix, Approximating the Maximum Weight Clique: An Evolutionary Game Theory Approach, manuscript in preparation, 1998.
- [63] I.M. Bomze and F. Rendl, Replicator dynamics for the evolution towards the maximum clique: Variants and experiments, in: *High Performance Algorithms and Software in Nonlinear Optimization*, R. De Leone, A. Murlì, P.M. Pardalos and G. Toraldo (eds.), Kluwer: 53–67, Dordrecht, 1998.
- [64] I.M. Bomze and V. Stix, Genetical engineering via negative fitness: Evolutionary dynamics for global optimization, to appear in: *Ann. Oper. Res.* Vol. 90, 1999.
- [65] R.E. Bonner, On some clustering techniques, *IBM J. Res. Develop.*, Vol. 8: 22–32, 1964.
- [66] R. Boppana and M.M. Halldórsson, Approximating maximum independent sets by excluding subgraphs, *BIT*, Vol. 32: 180–196, 1992.
- [67] M. Boulala and J.P. Uhry, Polytope des Indépendants dans un Graphe Serie-Parallele, *Discr. Math.*, Vol. 27: 225–243, 1979.
- [68] J.-M. Bourjolly, P. Gill, G. Laporte and H. Mercure, An exact quadratic 0-1 algorithm for the stable set problem, in [189]: 53–73, 1996.
- [69] J.-M. Bourjolly, G. Laporte and H. Mercure, A combinatorial column generation algorithm for the maximum stable set problem, *Oper. Res. Lett.* Vol. 20: 21–29, 1997.
- [70] D.P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice-Hall, Englewood Cliffs, N.J., 1994.
- [71] D. Brelaz, New methods to color the vertices of a graph, *Commun. ACM*, Vol. 22: 251–256, 1979.
- [72] M. Brockington and J.C. Culberson, Camouflaging independent sets in quasi-random graphs, in [189]: 75–88, 1996.
- [73] C. Bron and J. Kerbosch, Algorithm 457: Finding all cliques of an undirected graph, *Commun. ACM*, Vol. 16: 575–577, 1973.

- [74] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane and W. D. Smith, A new Table of constant weight codes, *IEEE Trans. Inform. Theory*, Vol. 36: 1334–1380, 1990.
- [75] M. Budinich, Bounds on the maximum clique of a graph, *submitted* (http://www.ts.infn.it/~mbh/MC_Bounds.ps.Z).
- [76] M. Budinich and P. Budinich, A Clifford algebra formulation of the maximum clique problem of a graph, preprint in preparation (will be available from <http://www.ts.infn.it/~mbh/PubNN.html>).
- [77] T.N. Bui and P.H. Eppley A hybrid genetic algorithm for the maximum clique problem, *Proc. 6th Int. Conf. Genetic Algorithms*: 478–484, 1995.
- [78] M. Burlet and J. Fonlupt, Polynomial algorithm to recognize a Meyniel graph, W.R. Pulleyblank (ed.), *Progress in Combinatorial Optimization*, Academic Press, Toronto: 69–99, 1984.
- [79] J.E. Burns, The maximum independent set problem for cubic planar graph, *Networks*, Vol. 9: 373–378, 1989
- [80] R. Carraghan and P.M. Pardalos, An exact algorithm for the maximum clique problem, *Oper. Res. Lett.*, Vol. 9: 375–382, 1990.
- [81] R. Carraghan and P.M. Pardalos, A parallel algorithm for the maximum weight clique problem, *Technical Report CS-90-40*, Dept. of Computer Science, Penn. State Univ., 1990.
- [82] B. Carter and K. Park, How good are genetic algorithms at finding large cliques: An experimental study, *Technical Report BU-CS-93-015*, Computer Science Dept., Boston University, 1993.
- [83] M.-S. Chang, Y.-H. Chen, G.J. Chang, and J.-H. Yan, Algorithmic aspects of the generalized clique-transversal problem on chordal graphs, *Discr. Appl. Math.*, Vol. 66: 189–203, 1996.
- [84] R.C. Chang, On the query complexity of clique size and maximum satisfiability, *J. Comput. Syst. Sci.*, Vol. 53: 298–313, 1996.
- [85] R.C. Chang, W.I. Gasarch, C. Lund, On bounded queries and approximation, *SIAM J. Comput.*, Vol. 26: 188–209, 1997.

- [86] R.C. Chang and H.S. Lee, Finding a maximum set of independent chords in a circle, *Inform. Proc. Lett.*, Vol. 41: 99–102, 1992.
- [87] N. Chiba and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.*, Vol. 14: 210–223, 1985.
- [88] N. Chiba, T. Nishizeki and N. Saito, An approximation algorithm for the maximum independent set problem on planar graphs, *SIAM J. Comput.*, Vol. 11: 663–675, 1982.
- [89] N. Chiba, T. Nishizeki and N. Saito, An algorithm for finding a large independent set in planar graphs, *Networks*, Vol. 13: 247–252, 1983.
- [90] E. Choukhmane and J. Franco, An approximation algorithm for the maximum independent set problem in cubic planar graphs, *Networks*, Vol. 16: 349–356, 1986.
- [91] M. Chrobak and J. Naor, An efficient parallel algorithm for computing a large independent set in a planar graph, *Algorithmica*, Vol. 6: 801–815, 1991.
- [92] V. Chvátal, On certain polytopes associated with graphs, *J. Combin. Theory B*, Vol. 18: 138–154, 1975.
- [93] V. Chvátal, Determining the stability number of a graph, *SIAM J. Comput.*, Vol. 6: 643–662. 1977.
- [94] V. Chvátal, A greedy heuristic for the set-covering problem, *Math. Oper. Res.*, Vol. 4: 233–23, 1979.
- [95] V. Chvátal, Perfectly orderable graphs, *Ann. Discr. Math.*, Vol. 21: 63–65, 1985.
- [96] K. Corradi and S. Szabo, A combinatorial approach for Keller’s conjecture, *Periodica Mathematica Hungarica*, Vol. 21: 95–100, 1990.
- [97] P. Crescenzi, C. Fiorini and R. Silvestri, A Note on the approximation of the MAX CLIQUE problem, *Inform. Process. Lett.* 40: 1-5, 1991.
- [98] F. Della Croce and R. Tadei, A multi-KP modeling for the maximum-clique problem, *Europ. J. Oper. Res.*, Vol. 73: 555–561, 1994.

- [99] V.-D. Cung, S. Dowaji, B. Le Cun, T. Mautor and C. Roucairol, Concurrent data structures and load balancing strategies for parallel branch-and-bound/ A^* algorithms, in S.N. Bhatt (ed.), *Parallel algorithms, 3rd DIMACS Implementation Challenge, October 17-19, 1994, AMS DIMACS Ser. Discrete Math. Theor. Comput. Sci.* Vol. 30: 141–162, 1997.
- [100] D.M. Cvetković, M. Doob and H. Sachs, *Spectra of Graphs*, Academic Press, New York, NY, 1980.
- [101] E. Dahlhaus and M. Karpinski, A fast parallel algorithm for computing all maximal cliques in a graph and the related problems, in: Algorithm Theory. Proc. 1st Scand. Workshop, Halmstad/Sweden 1988. Lect. Notes Comput. Sci. Vol. 318: 139–144, 1988.
- [102] C. De Simone and A. Sassano, Stability number and bull-and chair-free graphs, *Discr. Appl. Math.*, Vol. 41: 121–129, 1993.
- [103] V. Degot and J.M. Hualde, De l'utilisation de la notion de clique en matière de typologie des populations (in French), R.A.I.R.O., Vol. 9: 5–18, 1975.
- [104] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
- [105] J.F. Desler and S.L. Hakimi, On finding a maximum internally stable set of a graph, *Proc. of Fourth Annual Princeton Conference on Information Sciences and Systems*, Vol. 4: 459-462, Princeton, NJ, 1970.
- [106] G.A. Dirac, A property of 4-chromatic graphs and some remarks on critical graphs, *J. London Math. Soc.*, Vol. 27: 85–92, 1952.
- [107] G.A. Dirac, On rigid circuit graphs, *Abh. Math. Sem.*, Univ. Hamburg, Vol. 25: 71–76, 1961.
- [108] P. Doreian, A note on the detection of cliques in valued graphs, *Sociometry*, Vol. 32: 237–242, 1969.
- [109] D.-Z. Du, B. Gao and W. Wu, A special case for subset interconnection designs, *Discr. Appl. Math.*, Vol. 78: 51–60, 1997.

- [110] C. Ebenegger, P.L. Hammer, and D. de Werra, Pseudo-boolean functions and stability of graphs, *Ann. Discr. Math.*, Vol. 19: 83–98, 1984.
- [111] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, Approximating the maximum clique is almost NP -complete, *Proc. 32nd IEEE Symp. on Foundations of Computer Science*: 2–12, 1991.
- [112] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, Interactive proofs and the hardness of approximating cliques, *J. ACM*, Vol. 43: 268–292, 1996.
- [113] U. Feige and J. Kilian, Two prover protocols—Low error at affordable rates, *Proc. 26th Ann. ACM Symp. Theory of Comput.*: 172–183, 1994.
- [114] S. Felsner, R. Mueller and L. Wernisch, Trapezoid graphs and generalizations, geometry and algorithms, *Discr. Appl. Math.*, Vol. 74: 13–32, 1997.
- [115] T.A. Feo, M.G.C. Resende and S.H. Smith, A greedy randomized adaptive search procedure for maximum independent set, *Oper. Res.*, Vol. 42: 860–878, 1994.
- [116] M.L. Fisher and L.A. Wolsey, On the greedy heuristic for continuous covering and packing problems, *SIAM J. Alg. Discr. Meth.*, Vol. 3: 584–591, 1982.
- [117] C. Fleurent and J.A. Ferland, Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability, in [189]: 619–652, 1996.
- [118] J.A. Foster and T. Soule, Using genetic algorithms to find maximum cliques, *Technical Report LAL 95-12*, Dept. of Computer Science, U. Idaho, 1995.
- [119] A. Frank, Some polynomial algorithms for certain graphs and hypergraphs, *Proc. 5th British Combin. Conf.*: 211–226, 1976.
- [120] C. Friden, A. Hertz, and D. de Werra, STABULUS: A technique for finding stable sets in large graphs with tabu search, *Computing*, Vol. 42: 35–44, 1989.

- [121] C. Friden, A. Hertz and M. de Werra, TABARIS: An exact algorithm based on tabu search for finding a maximum independent set in a graph, *Comput. Oper. Res.*, Vol. 17: 437–445, 1990.
- [122] A. Frieze, On the independence number of random graphs, *Discr. Math.*, Vol. 81: 171–175, 1990.
- [123] K. Fujisawa, M. Kojima, and K. Nakata, Exploiting sparsity in primal-dual interior-point methods for semidefinite programming, *Math. Programming*, Vol. 79: 235–253, 1997.
- [124] N. Funabiki, Y. Takefuji, and K.C. Lee, A neural network model for finding a near-maximum clique, *J. Parallel Distrib. Comput.*, Vol. 14: 340–344, 1992.
- [125] M. Garey and D. Johnson, The complexity of near-optimal coloring, *J. ACM*, Vol. 23: 43–49, 1976.
- [126] M. Garey and D. Johnson, *Computers and Intractability—A guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
- [127] F. Gavril, Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, *SIAM J. Comput.*, Vol. 1: 180–187, 1972.
- [128] F. Gavril, Algorithms for a maximum clique and a maximum independent set of a circle graph, *Networks*, Vol. 3: 261–273, 1973.
- [129] F. Gavril, Algorithms on circular arc graphs, *Networks*, Vol. 4: 357–369, 1974.
- [130] M. Gendreau, J.C. Picard and L. Zubieta, An efficient implicit enumeration algorithm for the maximum clique problem, A. Kurzhanski et al. (eds), *Lecture Notes in Economics and Mathematical Systems*, Vol. 304: 70–91, Springer, Berlin 1988.
- [131] A. Gendreau, L. Salvail, and P. Soriano, Solving the maximum clique problem using a tabu search approach, *Ann. Oper. Res.*, Vol. 41: 385–403, 1993.
- [132] L. Gerhards and W. Lindenberg, Clique detection for nondirected graphs: Two new algorithms, *Computing*, Vol. 21: 295–322, 1979.

- [133] A.M.H. Gerards and A. Schrijver, Matrices with the Edmonds-Johnson property, *Combinatorica*, Vol. 6: 403–417, 1986.
- [134] L.E. Gibbons, D.W. Hearn and P.M. Pardalos, A continuous based heuristic for the maximum clique problem, in [189]: 103–124, 1996.
- [135] L.E. Gibbons, D.W. Hearn, P.M. Pardalos, and M.V. Ramana, Continuous characterizations of the maximum clique problem, *Math. Oper. Res.*, Vol. 22: 754–768, 1997.
- [136] F. Glover, Tabu search—Part I, *ORSA J. Comput.*, Vol. 1: 190–260, 1989.
- [137] F. Glover, Tabu search—Part II, *ORSA J. Comput.*, Vol. 2: 4–32, 1990.
- [138] F. Glover and M. Laguna, Tabu search, in *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves (ed.), Blackwell, Oxford, UK: 70–141, 1993.
- [139] G.H. Godbeer, J. Lipscomb, and M. Luby, On the computational complexity of finding stable state vectors in connectionist models (Hopfield nets), *Technical Report 208/88*, Dept. of Computer Science, University Toronto, 1988.
- [140] M.X. Goemans, Semidefinite programming in combinatorial optimization, *Math. Programming*, Vol. 79: 143–161, 1997.
- [141] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [142] M.K. Goldberg and R.D. Rivenburgh, Constructing cliques using restricted backtracking, in [189]: 89–101, 1996.
- [143] M. Goldberg and T. Spencer, A new parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, Vol. 18: 419–427, 1989.
- [144] M. Goldberg and T. Spencer, Constructing a maximal independent set in parallel, *SIAM J. Discr. Math.*, Vol. 2: 322–328, 1989.
- [145] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, NY, 1980.

- [146] M.C. Golumbic and P.L. Hammer, Stability in circular-arc-graphs, *J. Algorithms*, Vol. 9: 314–320, 1988.
- [147] G.R. Grimmett and W.R. Pulleyblank, Random near-regular graphs and the node packing problem, *Oper. Res. Lett.*, Vol. 4: 169–174, 1985.
- [148] T. Grossman, Applying the INN model to the max clique problem, in [189]: 125–146, 1996.
- [149] M. Grötschel, L. Lovász and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, Vol. 1: 169–197, 1981.
- [150] M. Grötschel, L. Lovász and A. Schrijver, Relaxations of vertex packing, *J. Combin. Theory B*, Vol. 40: 330–343, 1986.
- [151] M. Grötschel, L. Lovász and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, 2nd Ed., Springer, Berlin, 1993.
- [152] M. Grötschel, L. Lovász and A. Schrijver, Polynomial algorithms for perfect graphs, *Ann. Discr. Math.*, Vol. 21: 325–356, 1989.
- [153] U.I. Gupta, D.T. Lee and J.Y.T. Leung, Efficient algorithms for interval graphs and circular-arc graphs, *Networks*, Vol. 12: 459–467, 1982.
- [154] G. Hajós, Sur la factorisation des abeliens, *Casopis* Vol. 74: 189–196, 1950.
- [155] M.-H. Han and D. Jang, The use of maximum curvature points for the recognition of partially occluded objects, *Pattern Recognition*, Vol. 23: 21–33, 1990.
- [156] P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem, *Computing*, Vol. 44: 279–303, 1990.
- [157] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [158] F. Harary and I.C. Ross, A procedure for clique detection using the group matrix, *Sociometry*, Vol. 20: 205–215, 1957.

- [159] J. Hasselberg, P.M. Pardalos and G. Vairaktarakis, Test case generators and computational results for the maximum clique problem, *J. Global Optim.*, Vol. 3: 463–482, 1993.
- [160] J. Hastad, Testing of the long code and hardness clique, *Proc. 28th Ann. Symp. Theory of Comput.*: 11–19, 1996.
- [161] J. Hastad, Clique is hard to approximate within $n^{1-\varepsilon}$, *Proc. 37th Ann. Symp. Found. Comput. Sci.*: 627–636, 1996.
- [162] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.
- [163] R.B. Hayward, Weakly triangulated graphs, *J. of Combin. Theory B*, Vol. 39: 200–209, 1985.
- [164] R. Hayward, C.T. Hoang and F. Maffray, Optimizing weakly triangulated graphs, *Graphs and Combinatorics*, Vol. 5: 339–349, 1989. See also *Erratum*, Vol. 6: 33–35, 1990.
- [165] I. Heller, On linear systems with integral valued solutions, *Pacific J. Math.*, Vol. 7: 1351–1364, 1957.
- [166] C. Helmberg, F. Rendl, R.J. Vanderbei and H. Wolkowicz, An interior-point method for semidefinite programming, *SIAM J. Optim.*, Vol. 6: 342–361, 1996.
- [167] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, CA, 1991.
- [168] A. Hertz, E. Taillard and D. de Werra, Tabu search, in [2]: 121–136.
- [169] M. Hifi, A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems, *J. Oper. Res. Soc.* Vol. 48: 612–622, 1997.
- [170] C.W. Ho and R.C.T. Lee, Efficient parallel algorithms for finding maximal cliques, clique trees, and minimum coloring of chordal graphs, *Inform. Proc. Lett.*, Vol. 28: 301–309, 1988.
- [171] J. Hofbauer and K. Sigmund, *Evolutionary Games and Population Dynamics*, Cambridge University Press, Cambridge, UK, 1998.

- [172] J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
- [173] S. Homer and M. Peinado, Experiments with polynomial-time CLIQUE approximation algorithms on very large graphs, in [189]: 147–167, 1996.
- [174] J.J. Hopfield and D.W. Tank, “Neural” computation of decisions in optimization problems, *Biol. Cybern.*, Vol. 52: 141–152, 1985.
- [175] R. Horaud and T. Skordas, Stereo correspondence through feature grouping and maximal cliques, *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 11: 1168–1180, 1989.
- [176] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.
- [177] D.J. Houck, On the vertex packing problem, Ph.D. dissertation, The Johns Hopkins University, Baltimore, 1974.
- [178] D.J. Houck and R.R. Vemuganti, An algorithm for the vertex packing problem, *Oper. Res.*, Vol. 25: 773–787, 1977.
- [179] W.L. Hsu, Maximum weight clique algorithms for circular-arc graphs and circle graphs, *SIAM J. Comput.*, Vol. 14: 224–231, 1985.
- [180] W.L. Hsu, The coloring and maximum independent set problems on planar perfect graphs, *J. ACM*, Vol. 35: 535–563, 1988.
- [181] W. Hsu, Y. Ikura and G.L. Nemhauser, A polynomial algorithm for maximum weighted vertex packings on graphs without long odd cycles, *Math. Programming*, Vol. 20: 225–232, 1981.
- [182] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan and R.E. Stearns, The complexity of planar counting problems, *SIAM J. Comput.*, Vol. 27: 1142–1167, 1998.
- [183] A. Jagota, Approximating Maximum Clique with a Hopfield Network, *IEEE Trans. Neural Networks*, Vol. 6: 724–735, 1995.
- [184] A. Jagota (ed.), Neural Networks for Combinatorial Optimization, *J. Artif. Neural Networks*, Vol. 2, 1995.

- [185] A. Jagota and K.W. Regan, Performance of neural net heuristics for maximum clique on diverse highly compressible graphs, *J. Global Optim.*, Vol. 10: 439–465, 1997.
- [186] A. Jagota, L. Sanchis, and R. Ganesan, Approximately solving maximum clique using neural networks and related heuristics, in [189]: 169–204, 1996.
- [187] M. Jerrum, Large cliques elude the Metropolis process, *Random Structures and Algorithms*, Vol. 3: 347–359, 1992.
- [188] D.S. Johnson, Approximation algorithms for combinatorial problems, *J. Comput. Syst. Sci.*, Vol. 9: 256–278, 1974.
- [189] D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Vol. 26, American Mathematical Society, 1996 (see also <http://dimacs.rutgers.edu/Volumes/Vol26.html>).
- [190] D.S. Johnson, M. Yannakakis and C.H. Papadimitriou, On generating all maximal independent sets, *Inform. Proc. Lett.*, Vol. 27: 119–123, 1988.
- [191] L.F. Johnson, Determining cliques of a graph, *Proc. 5th Manitoba Conf. on Numer. Math.*: 429–437, 1975.
- [192] H.C. Johnston, Cliques of a graph: Variations on the Bron-Kerbosch algorithm, *Int. J. Comput. Inform. Sci.*, Vol. 5: 209–238, 1976.
- [193] N. Karmarkar, K.G. Ramakrishnan and M.G.C. Resende, An interior point approach to the maximum independent set problem in dense random graphs, *Proc. XV Latin American Conference on Informatics, Santiago, Chile*, I: 241–260, 1989.
- [194] R.M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, New York: 85–103, 1972.
- [195] R.M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, *J. ACM*, Vol. 32: 762–773, 1985.

- [196] M. Karpinski and A. Zelikovsky, Approximating dense cases of covering problems, in P.M. Pardalos et al. (eds), *Network Design: Connectivity and Facilities Location, DIMACS Workshop, April 28–30, 1997. AMS-DIMACS Ser. Discr. Math. Comput. Sci.* Vol. 40: 169–178, 1998.
- [197] T. Kashiwabara, S. Masuda, K. Nakajima and T. Fujisawa, Generation of maximum independent sets of a bipartite graph and maximum cliques of a circular-arc-graph, *J. Algorithms*, Vol. 13: 161–174, 1992.
- [198] O.H. Keller, Über die lückenlose Erfüllung des Raumes mit Würfeln (in German), *J. Reine Angew. Math.*, Vol. 163: 231–248, 1930.
- [199] P. Kikusts, Another algorithm determining the independence number of a graph, *Elektron. Inf. Verarb. Kybern. ELK* 22: 157–166, 1986.
- [200] D.S. Kim and D.-Z. Du, On conflict-free channel set assignments for optical cluster-based hypercube networks, In *DIMACS Series Vol. 46*: 109-116, American Mathematical Society (1999).
- [201] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, *Science*, Vol. 220: 671–680, 1983.
- [202] P.N. Klein, Efficient parallel algorithms for chordal graphs, *Proc. 29th Ann. Symp. Found. Computer Sci.*: 150–161, 1988.
- [203] J. Kleinberg and M.X. Goemans, The Lovász theta function and a semidefinite programming relaxation of vertex cover, *SIAM J. Discr. Math.* Vol. 11: 196–204, 1997.
- [204] D.E. Knuth, The sandwich theorem, *Electr. J. Combin.*, Vol. 1: A1, 1994
(http://www2.combinatorics.org/Volume_1/volume1.html#A1).
- [205] R. Kopf and G. Ruhe, A computational study of the weighted independent set problem for general graphs, *Found. Control Engin.*, Vol. 12: 167–180, 1987.
- [206] D. Kozen, A clique problem equivalent to graph isomorphism, *SIGACT News*: 50–52, Summer 1978.
- [207] J.C. Lagarias and P.W. Shor, Keller's cube-tiling conjecture is false in high dimensions, *Bull. Amer. Math. Soc. New Ser.*, Vol. 27: 279–283, 1992.

- [208] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Reidel, Dordrecht, 1987.
- [209] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, 1976.
- [210] L.J. Leifman, On construction of all maximal complete subgraphs (cliques) of a graph, Technical Report, Dept. of Mathematics., University of Haifa, Israel, 1976.
- [211] F. Lin and K. Lee, A parallel computation network for the maximum clique problem, in *Proc. 1st Int. Conf. Fuzzy Theory Tech.*, Baton Rouge, Louisiana, 1992.
- [212] R.J. Lipton, On proving that a graph has no large clique: A connection with Ramsey theory, *Inform. Proc. Lett.*, Vol. 58: 39–42, 1996.
- [213] C.-K. Looi, Neural network methods in combinatorial optimization, *Comput. Oper. Res.*, Vol. 19: 191–208, 1992.
- [214] E. Loukakis, A new backtracking algorithm for generating the family of maximal independent sets of a graph, *Comp. Math. Appl.*, Vol. 9: 583–589, 1983.
- [215] E. Loukakis and C. Tsouros, A depth first search algorithm to generate the family of maximal independent sets of a graph lexicographically, *Computing*, Vol. 27: 249–266, 1981.
- [216] E. Loukakis and C. Tsouros, Determining the number of internal stability of a graph, *Int. J. Comp. Math.*, Vol. 11: 207–220, 1982.
- [217] E. Loukakis and C. Tsouros, An algorithm for the maximum internally stable set in a weighted graph, *Int. J. Comput. Math.*, Vol. 13: 117–129, 1983.
- [218] L. Lovász, On the Shannon capacity of a graph, *IEEE Trans. Inform. Theory*, Vol. 25: 1–7, 1979.
- [219] L. Lovász and A. Schrijver, Cones of matrices and set-functions and 0-1 optimization, *SIAM J. Optim.*, Vol. 1: 166–190, 1991.
- [220] M. Luby, A simple parallel algorithm for the maximum independent set problem, *SIAM J. Comput.*, Vol. 15: 1036–1053, 1986.

- [221] J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1979.
- [222] K. Maghout, Sur la determination des nombres de stabilite et du nombre chromatique d'un graphe, *C.R. Acad. Sci., Paris*, Vol. 248: 2522–2523, 1959.
- [223] G.K. Manacher and T.A. Mankus, Finding a maximum clique in a set of proper circular arcs in $O(n)$ with applications, *Int. J. Found. Comput. Sci.*, Vol. 8: 443–467, 1997.
- [224] C. Mannino and A. Sassano, Edge projection and the maximum cardinality stable set problem, in [189]: 205–219, 1996.
- [225] E. Marchiori, A simple heuristic based genetic algorithm for the maximum clique problem, *Proc. ACM Symp. Appl. Comput.*: 366–373, 1998.
- [226] P.M. Marcus, Derivation of maximal compatibles using boolean algebra, *IBM J. Res. Develop.*, Vol. 8: 537–538, 1964.
- [227] S. Masuda and K. Nakajima, An optimal algorithm for finding a maximum independent set of a circular-arc graph, *SIAM J. Comput.*, Vol. 17: 41–52, 1988.
- [228] S. Masuda, K. Nakajima, T. Kashiwabara and T. Fujisawa, Efficient algorithms for finding maximum cliques of an overlap graph, *Networks*, Vol. 20: 157–171, 1990.
- [229] D.W. Matula, On the complete subgraphs of a random graph, *Proc. 2nd Conf. Combin. Theory Appl.*, Chapel Hill, NC: 356–369, 1970.
- [230] D.W. Matula, The largest clique size in a random graph, *Technical Report CS 7608*, Department of Computer Science, Southern Methodist University, 1976.
- [231] W. Meeusen and L. Cuyvers, Clique detection in directed graphs: A new algorithm, *J. Comput. Appl. Math.*, Vol. 1: 185–193, 1975.
- [232] W. Meeusen and L. Cuyvers, An annotated bibliography of clique-detection algorithms and related graph-theoretical problems, *CAM Discussion Paper 7909*, State University Center Antwerp, 1979.

- [233] H. Meyniel, On the perfect graph conjecture, *Discr. Math.*, Vol. 16: 339–342, 1976.
- [234] H. Meyniel, A new property of imperfect graphs and some consequences, *Europ. J. Combin.*, Vol. 8: 313–316, 1987.
- [235] G.J. Minty, On maximal independent sets of vertices in claw-free graphs, *J. Combin. Theory B*, Vol. 28: 284–304, 1980.
- [236] B. Mohar and S. Poljak, Eigenvalues in combinatorial optimization, In *Combinatorial and Graph-Theoretic Problems in Linear Algebra* R. Brualdi, S. Friedland and V. Klee, eds., IMA Volumes in Mathematics and Its Applications, Vol. 50, Springer, Berlin, 1993.
- [237] J.W. Moon and L. Moser, On cliques in graphs, *Israel Journal of Mathematics*, Vol. 3: 23–28, 1965.
- [238] T.S. Motzkin and E.G. Straus, Maxima for graphs and a new proof of a theorem of Turán, *Canad. J. Math.*, Vol. 17: 533–540, 1965.
- [239] G.D. Mulligan and D.G. Corneil, Corrections to Bierstone's algorithm for generating cliques, *J. ACM*, Vol. 19: 244–247, 1972.
- [240] W.J. Murphy, Computing independent sets in graphs with large girth, *Discr. Appl. Math.*, Vol. 36: 167–170, 1992.
- [241] A.S. Murthy, G. Parthasarathy and V.U.K. Sastry, Clique finding—A genetic approach, *Proc. 1st IEEE Conf. Evolutionary Comput.*: 18–21, 1994.
- [242] J. Naor, M. Naor and A.A. Schaffer, Fast parallel algorithms for chordal graphs, *Proc. 19th Annual ACM Symp. Theory Comput.*: 355–364, 1987.
- [243] G.L. Nemhauser and G. Sigismondi, A strong cutting plane/branch-and bound algorithm for node packing, *J. Opl. Res. Soc.*, Vol. 43: 443–457, 1992.
- [244] G.L. Nemhauser and L.E. Trotter,, Properties of vertex packings and independence system polyhedra, *Math. Programming*, Vol. 6: 48–61, 1974.
- [245] G.L. Nemhauser and L.E. Trotter, Vertex packings: Structural properties and algorithms, *Math. Programming*, Vol. 8: 232–248, 1975.

- [246] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, J. Wiley & Sons, New York, 1988.
- [247] H. Ogawa, Labeled point pattern matching by Delaunay triangulation and maximal cliques, *Pattern Recognition*, Vol. 19: 35–40, 1986.
- [248] S. Olariu, Weak bipolarizable graphs, *Discr. Math.*, Vol. 74: 159–171, 1989.
- [249] R.E. Osteen, Clique detection algorithms based on line addition and line removal, *SIAM J. Appl. Math.*, Vol. 26: 126–135, 1974.
- [250] R.E. Osteen and J.T. Tou, A clique-detection algorithm based on neighborhoods in graphs, *Int. J. Comput. Inform. Sci.*, Vol. 2: 257–268, 1973.
- [251] Q. Ouyang, P.D. Kaplan, S. Liu and A. Libchaber, DNA solutions of the maximal clique problem, *Science*, Vol. 278: 446–449, 1997.
- [252] M.W. Padberg, Covering, packing and knapsack problems, *Ann. Discr. Math.*, Vol. 4: 265–287, 1979.
- [253] A. Panconesi and D. Ranjan, Quantifiers and approximation, *Proc. 22nd ACM Ann. Symp. Theory of Comput.*: 446–456, 1990.
- [254] A. Panconesi and D. Ranjan, Quantifiers and approximation, *Theor. Comput. Sci.*, Vol. 107: 145–163, 1993.
- [255] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Amsterdam, 1994.
- [256] C. Papadimitriou and M. Yannakakis, The clique problem for planar graphs, *Inform. Proc. Lett.*, Vol. 13: 131–133, 1981.
- [257] C. Papadimitriou and M. Yannakakis, Optimization, approximation and complexity classes, *Proc. 20th Ann. ACM STOC*: 229–234, 1988.
- [258] C. Papadimitriou and M. Yannakakis, Optimization, approximation and complexity classes, *J. Comput. Syst. Sci.*, Vol. 43: 425–440, 1991.
- [259] P.M. Pardalos, Continuous approaches to discrete optimization problems, in *Nonlinear Optimization and Applications*, G. Di Pillo and F. Giannessi, eds., Plenum Press, New York: 313–328, 1996.

- [260] P.M. Pardalos and N. Desai, An algorithm for finding a maximum weighted independent set in an arbitrary graph, *Int. J. Comput. Math.*, Vol. 38: 163–175, 1991.
- [261] P.M. Pardalos and A.T. Phillips, A global optimization approach for solving the maximum clique problem, *Int. J. Comput. Math.*, Vol. 33: 209–216, 1990.
- [262] P.M. Pardalos, J. Rappe and M.G.C. Resende, An Exact Parallel Algorithm for the Maximum Clique Problem, in *High Performance Algorithms and Software in Nonlinear Optimization*, R. De Leone, A. Murlí, P.M. Pardalos and G. Toraldo (eds.), Kluwer: 279–300, Dordrecht, 1998.
- [263] P. M. Pardalos and G. Rodgers, Parallel branch and bound algorithms for quadratic zero-one programming on a hypercube architecture, *Ann. Oper. Res.*, Vol. 22: 271–292, 1990.
- [264] P. M. Pardalos and G. Rodgers, Computational aspects of a branch and bound algorithm for quadratic zero-one programming, *Computing*, Vol. 45: 131–144, 1990.
- [265] P.M. Pardalos and G.P. Rodgers, A branch and bound algorithm for the maximum clique problem, *Comput. Oper. Res.*, Vol. 19: 363–375, 1992.
- [266] P.M. Pardalos and J. Xue, The maximum clique problem, *J. Global Optim.*, Vol. 4: 301–328, 1994.
- [267] K. Park and B. Carter, On the effectiveness of genetic search in combinatorial optimization, *Technical Report BU-CS-94-010*, Computer Science Dept., Boston University, 1994. (a shorter version appears in: *Proc. 10th ACM Symp. Appl. Comput.*, 1995.)
- [268] M.C. Paull and S.H. Unger, Minimizing the number of states in incompletely specified sequential switching functions, *IRE Trans. Electr. Comput.*, Vol. EC-8: 356–367, 1959.
- [269] E.R. Peay, Jr., An iterative clique detection procedure, Michigan Math. Psychol. Program: MMPP 70-4, 1970.
- [270] E.R. Peay, Jr., Hierarchical clique structures, *Sociometry*, Vol. 37: 54–65, 1974.

- [271] M. Pelillo, Relaxation labeling networks for the maximum clique problem, *J. Artif. Neural Networks*, Vol. 2: 313–328, 1995.
- [272] M. Pelillo, A unifying framework for relational structure matching, in: *Proc. 14th Int. Conf. Pattern Recognition*: 1316–1319, 1998.
- [273] M. Pelillo, Replicator equations, maximal cliques, and graph isomorphism, *Neural Computation*, to appear, 1999.
- [274] M. Pelillo and A. Jagota, Feasible and infeasible maxima in a quadratic program for maximum clique, *J. Artif. Neural Networks*, Vol. 2: 411–420, 1995.
- [275] M. Pelillo, K. Siddiqi and S.W. Zucker, Matching hierarchical structures using association graphs, in H. Burkhardt and B. Neumann (eds.), *Computer Vision—ECCV'98* (Lecture Notes in Computer Science, Vol. 1407), Springer, Berlin: 3–16, 1998.
- [276] O. Perron, Über lückenlose Ausfüllung des n -dimensionalen Raumes durch kongruente Würfel (in German), *Math. Z.*, Vol. 46: 1–26, 161–180. Zbl 22, 202, 1940.
- [277] C. Peterson and B. Söderberg, Artificial neural networks, in [2]: 173–213, 1997.
- [278] J.C. Picard and M. Queyranne, On the integer-valued variables in the linear vertex packing problem, *Math. Programming*, Vol. 12: 97–101, 1977.
- [279] F. Pla and J.A. Marchant, Matching feature points in image sequences through a region-based method, *Comput. Vision Image Understanding*, Vol. 66: 271–285, 1997.
- [280] F.P. Preparata, G. Metze and R.T. Chien, On the connection assignment problem of diagnosable systems, *IEEE Trans. Electr. Comput.*, Vol. 16: 848–854, 1967.
- [281] W.R. Pulleyblank, Minimum node covers and 2-bicritical graphs, *Math. Programming*, Vol. 17: 91–103, 1979.
- [282] B. Radig, Image sequence analysis using relational structures, *Pattern Recognition*, Vol. 17: 161–167, 1984.

- [283] P. Raghavan, Probabilistic construction of deterministic algorithms: Approximating packing integer programs, *J. Comput. Syst. Sci.*, Vol. 37: 130–143, 1988.
- [284] J. Ramanujam and P. Sadayappan, Optimization by neural networks, *Proc. IEEE Int. Conf. Neural Networks*: 325–332, 1988.
- [285] M.C.G. Resende, T.A. Feo, and S.H. Smith, Fortran subroutines for approximate solution of maximum independent set problems using GRASP, *ACM Transactions on Mathematical Software*, to appear 1999.
- [286] J.M. Robson, Algorithms for maximum independent sets, *J. Algorithms*, Vol. 7: 425–440, 1986.
- [287] D.J. Rose, Triangulated graphs and the elimination process, *J. Math. Anal. Appl.*, Vol. 32: 597–609, 1970.
- [288] D.J. Rose, R.E. Tarjan and G.S. Leuker, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Comput.*, Vol. 5: 266–283, 1976.
- [289] D. Rotem and J. Urrutia, Finding maximum cliques in circle graphs, *Networks*, Vol. 11: 269–278, 1981.
- [290] B.E. Sagan, A note on independent sets in trees, *SIAM J. Discr. Math.*, Vol. 1: 105–108, 1988.
- [291] L. Sanchis and A. Jagota, Some experimental and theoretical results on test case generators for the maximum clique problem, *INFORMS J. Comput.*, Vol. 8: 87–102, 1996.
- [292] C.E. Shannon, Certain results in coding theory for noisy channels, *Inform. and Control*, Vol. 1: 6–25, 1957.
- [293] N. Z. Shor, Dual quadratic estimates in polynomial and Boolean programming, in *Computational Methods in Global Optimization*, P. M. Pardalos and J. B. Rosen (eds.), *Ann. Oper. Res.*, Vol. 25: 163–168, 1990.
- [294] Y. Srivastava, S. Dasgupta, and S.M. Reddy, Neural network solutions to a graph theoretic problem, in *Proc. IEEE Int. Symp. Circuits Syst.*: 2528–2531, 1990.

- [295] Y. Shrivastava, S. Dasgupta, and S.M. Reddy, Guaranteed convergence in a class of Hopfield networks, *IEEE Trans. Neural Networks*, Vol. 3: 951–961, 1992.
- [296] N.J.A. Sloane, Unsolved problems in graph theory arising from the study of codes, *J. Graph Theory Notes of New York*, Vol. 18: 11–20, 1989.
- [297] P. Soriano and M. Gendreau, Tabu search algorithms for the maximum clique problem, in [189]: 221–242, 1996.
- [298] P. Soriano and M. Gendreau, Diversification strategies in tabu search algorithms for the maximum clique problem, *Ann. Oper. Res.*, Vol. 63: 189–207, 1996.
- [299] V.T. Sós and E.G. Straus, Extremal of functions on graphs with applications to graphs and hypergraphs, *J. Combin. Theory B*, Vol. 32: 246–257, 1982.
- [300] S.K. Stein, Algebraic tiling, *Amer. Math. Monthly*, Vol. 81: 445–462, 1974.
- [301] S. Szabó, A reduction of Keller’s conjecture, *Periodica Math. Hung.*, Vol. 17: 265–277, 1986.
- [302] Y. Takefuji, *Neural Network Parallel Computing*, Kluwer, Dordrecht, 1992.
- [303] Y. Takefuji, L. Chen, K. Lee and J. Huffman, Parallel algorithms for finding a near-maximum independent set of a circle graph, *IEEE Trans. Neural Networks*, Vol. 1: 263–267, 1990.
- [304] R.E. Tarjan, Finding a maximum clique, *Technical Report 72-123*, Computer Sci. Dept., Cornell University, Ithaca, NY, 1972.
- [305] R.E. Tarjan and A.E. Trojanowski, Finding a maximum independent set, *SIAM J. Comput.*, Vol. 6: 537–546, 1977.
- [306] R.E. Tarjan and M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM J. Comput.*, Vol. 13: 566–579, 1984.

- [307] E. Tomita, Y. Kohata and H. Takahashi, A simple algorithm for finding a maximum clique, *Technical Report UEC-TR-C5*, 1988.
- [308] E. Tomita, S. Mitsuma and H. Takahashi, Two algorithms for finding a near-maximum clique, *Technical Report UEC-TR-C1*, 1988.
- [309] E. Tomita, A. Tanaka and H. Takahashi, The worst-case time complexity for finding all the cliques, *Technical Report UEC-TR-C5*, 1988.
- [310] S. Tsukiyama, M. Ide, H. Aviyoshi and I. Shirakawa, A new algorithm for generating all the maximum independent sets, *SIAM J. Comput.*, Vol. 6: 505–517, 1977.
- [311] P. Turán, On an extremal problem in graph theory (in Hungarian), *Mat. és Fiz. Lapok*, Vol. 48: 436–452, 1941.
- [312] L.G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.*, Vol. 8: 410–421, 1979.
- [313] O. Verbitsky, On the hardness of approximating some optimization problems that are supposedly easier than MAX CLIQUE, *Comb. Probab. Comput.* Vol. 4: 167–180, 1995.
- [314] M. Vingron and P. Argos, Motif recognition and alignment for many sequences by comparison of dot-matrices, *J. Molecular Biol.*, Vol. 218: 33–43, 1991.
- [315] M. Vingron and P.A. Pevzner, Multiple sequence comparison and consistency on multipartite graphs. *Adv. Appl. Math* Vol.16: 1–22, 1995.
- [316] H.S. Wilf, The eigenvalues of a graph and its chromatic number, *J. London Math. Soc.*, Vol. 42: 330–332, 1967.
- [317] H.S. Wilf, *Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [318] H.S. Wilf, Spectral bounds for the clique and independence numbers of graphs, *J. Combin. Theory B*, Vol. 40: 113–117, 1986.
- [319] S. Wimer, R.Y. Pinter and J. Feldman, Optimal chaining of CMOS transistors in a functional cell, *IEEE Trans. Computer-Aided Design*, Vol. 6: 795–801, 1987.

- [320] J. Wu, T. Harada and T. Fukao, New method to the solution of maximum clique problem: Mean-field approximation algorithm and its experimentation. *Proc. IEEE Int. Conf. Syst., Man, Cybern.*: 2248–2253, 1994.
- [321] J. Xue, Fast Algorithms for Vertex Packing and Related Problems, *Ph.D Thesis*, GSIA, Carnegie Mellon University, 1991.
- [322] J. Xue, Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem, *Networks*, Vol. 24: 109–120, 1994.
- [323] M. Yannakakis, On the approximation of maximum satisfiability, *Proc. 3rd Annual ACM-SIAM Symp. Discr. Algorithms*, 1992.
- [324] M.-S. Yu, L. Yu and S.-J. Chang, Sequential and parallel algorithms for the maximum-weight independent set problem on permutation graphs, *Inform. Proc. Lett.*, Vol. 46: 7–11, 1993.
- [325] B.-T. Zhang and S.-Y. Shin, Code optimization for DNA computing of maximal cliques, in *Advances in Soft Computing—Engineering and Design*, Springer, Berlin, 1998.

Linear Assignment Problems and Extensions*

Rainer E. Burkard and Eranda Çela

Institute of Mathematics

Technical University Graz, Steyrergasse 30, A-8010 Graz, Austria

E-mail: {burkard,cela}@opt.math.tu-graz.ac.at

Contents

1 Assignments	76
2 Linear Sum Assignment Problems (LSAPs)	83
3 Algorithms for the LSAP	88
3.1 Primal-Dual Algorithms	89
3.2 Simplex-Based Algorithms	96
3.3 Other Algorithms	102
3.4 On Parallel Algorithms	105
4 Asymptotic Behavior and Probabilistic Analysis	108
5 Bottleneck Assignment Problems	113
6 Assignment Problems with Other Objective Functions	118
7 Available computer codes and test instances	122
8 Multidimensional Assignment Problems	123
8.1 General Remarks and Applications	123
8.2 Axial 3-Dimensional Assignment Problems	127
8.3 Planar 3-Dimensional Assignment Problems	132

*This research has been supported by the Spezialforschungsbereich F 003 "Optimierung und Kontrolle", Projektbereich Diskrete Optimierung.

References

1 Assignments

Assignment problems deal with the question how to assign n items (e.g. jobs) to n machines (or workers) in the best possible way. They consist of two components: the *assignment* as underlying combinatorial structure and an objective function modeling the "best way".

Mathematically an *assignment* is nothing else than a bijective mapping of a finite set into itself, i.e., a *permutation*. Assignments can be modeled and visualized in different ways: every permutation ϕ of the set $N = \{1, \dots, n\}$ corresponds in a unique way to a *permutation matrix* $X_\phi = (x_{ij})$ with $x_{ij} = 1$ for $j = \phi(i)$ and $x_{ij} = 0$ for $j \neq \phi(i)$. We can view this matrix X_ϕ as adjacency matrix of a bipartite graph $G_\phi = (V, W; E)$, where the vertex sets V and W have n vertices, i.e., $|V| = |W| = n$, and there is an edge $(i, j) \in E$ iff $j = \phi(i)$, cf. Fig. 1.

$$\begin{aligned}\phi &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix} \\ X_\phi &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}\end{aligned}$$

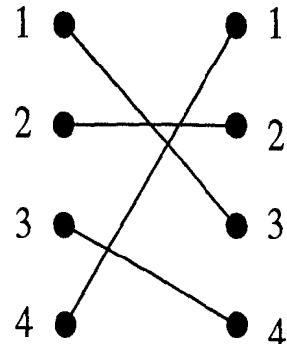


Figure 1: Different representations of assignments

The set of all assignments of n items will be denoted by S_n and has $n!$ elements. We can describe this set by the following equations called *assignment constraints*

$$\sum_{i=1}^n x_{ij} = 1 \quad \text{for all } j = 1, \dots, n$$

$$\begin{aligned} \sum_{j=1}^n x_{ij} &= 1 && \text{for all } i = 1, \dots, n \\ x_{ij} &\in \{0, 1\} && \text{for all } i, j = 1, \dots, n \end{aligned} \tag{1}$$

Let A be the coefficient matrix of the system of equations (1). Matrix A is *totally unimodular*, i.e., every square submatrix of A has a determinant of value $+1$, -1 or 0 . By replacing the conditions $x_{ij} \in \{0, 1\}$ by $x_{ij} \geq 0$ in (1), we get a *doubly stochastic matrix*. The set of all doubly stochastic matrices forms the *assignment polytope* P_A . Due to a famous result of Birkhoff [36], the assignment polytope P_A is the convex hull of all assignments:

Theorem 1.1 (Birkhoff [36], 1946)

The vertices of the assignment polytope correspond uniquely to permutation matrices.

Differently said every doubly stochastic matrix can be written as convex combination of permutation matrices. Further properties of the assignment polytope are summarized in the following theorem:

Theorem 1.2 (Balinski and Russakoff [22], 1974)

Let P_A be the assignment polytope for assignments on a set N with n elements. Then

1. $\dim P_A = (n - 1)^2$.
2. *Every of the $n!$ vertices of P_A coincides with $\sum_{k=0}^{n-1} \binom{n}{k} (n - k - 1)!$ different edges.*
3. *Any two vertices are joined by a path with at most two edges, i.e., the diameter of P_A is 2.*
4. *P_A is Hamiltonian.*

A polytope is *Hamiltonian*, if there exists a closed path along its edges which visits every vertex of the polytope just once.

Consider a bipartite graph $G = (V, W; E)$ with vertex sets $V = \{v_1, v_2, \dots, v_n\}$ and $W = \{w_1, w_2, \dots, w_n\}$, and edge set E . A subset M of E is called a *matching*, if every vertex of G is incident with at most one edge from M . The cardinality of M is called cardinality of the matching. The *maximum matching problem* asks for a matching with as many edges as possible. A matching M is called a *perfect matching*, if every vertex of G

is incident with exactly one edge from M , i.e., $|M| = n$. Evidently, every perfect matching is a maximum matching and corresponds to an assignment. The number of different perfect matchings in a bipartite graph G is given by the *permanent* of the corresponding $n \times n$ adjacency matrix $A = (a_{ij})$ defined by

$$a_{ij} := \begin{cases} 1 & \text{if } (v_i, w_j) \in E \\ 0 & \text{if } (v_i, w_j) \notin E \end{cases}$$

The permanent $\text{per}(A)$ is defined by

$$\text{per}(A) := \sum_{\phi \in S_n} a_{1\phi(1)} a_{2\phi(2)} \dots a_{n\phi(n)}.$$

The numerical evaluation of a permanent (and the determination of the number of different perfect matchings in a graph G) is $\#P$ -complete (Valiant [165]). This implies that the determination of the number of different perfect matchings in G is at least as hard as any NP-complete problem. The following theorem due to Hall [99] states a necessary and sufficient condition for the *existence* of a perfect matching in a bipartite graph. Halmos interpreted such a perfect matching as a marriage between ladies in the set V and gentlemen in set W , and due to this interpretation Hall's theorem is known as *Marriage theorem*. For a vertex $v \in V$ let $N(v)$ be the set of its neighbors, i.e., the set of all vertices $w \in W$ which are connected with v by an edge in E . Thus $N(v)$ contains the friends of v . Moreover, for any subset V' of V let $N(V') = \bigcup_{v \in V'} N(v)$.

Theorem 1.3 (Marriage theorem, Hall [99], 1935)

Let $G = (V, W; E)$ be a bipartite graph. G contains an assignment (perfect matching, marriage) if and only if $|V| = |W|$ and for all subsets V' of V

$$|V'| \leq |N(V')| \quad (\text{Hall condition})$$

This theorem says that by checking the exponentially many subsets V' of V we can decide, whether the graph G has a perfect matching or not. Hopcroft and Karp [103] gave a polynomial-time algorithm to decide this question. They construct a perfect matching, if it exists, in $O(|E|\sqrt{|V|})$ steps. Alt, Blum, Mehlhorn and Paul [9] improved the complexity for dense graphs by a fast adjacency matrix scanning technique and obtain an $O(|V|^{1.5}\sqrt{|E|/\log|V|})$ implementation for the Hopcroft-Karp algorithm. The decision can also be made faster by a randomized algorithm which is a Monte-Carlo approach. This algorithm may err with an arbitrarily small

probability in the case that its output does not confirm the existence of a matching. The randomized algorithm is based on the relationship between the determinant of the Tutte matrix of a graph and the existence of a perfect matching (Tutte [164]). The *Tutte matrix* $A(x) = (x_{ij})$ of an (undirected) graph $G = (V, E)$ is a skew symmetric matrix with variable entries x_{ij} , $x_{ij} = -x_{ji}$, where $x_{ij} = x_{ji} \equiv 0$, if (i, j) is not an edge of G .

Theorem 1.4 (Tutte [164], 1947)

Let $A(x)$ be the Tutte matrix of graph $G = (V, E)$. There exists a perfect matching in G , if and only if the determinant of $A(x)$ is not identically equal to 0.

A randomized algorithm to decide whether G has a perfect matching can be obtained by generating x_{ij} uniformly at random from $\{1, 2, \dots, 2|E|\}$ and by computing the determinant of the corresponding Tutte matrix $A(x)$, and eventually repeating this process a fixed number of times. The probability of an error can be estimated by applying a result of Schwarz [160] on the probability of hitting roots of a given polynomial. By applying Schwartz's results one gets that the error probability of the above randomized algorithm is equal to $(1/|E|)^r$, where r is the number of repetitions and $|E|$ is the number of edges of G . Since an $n \times n$ determinant can be evaluated in $O(n^{2.376})$ time (see Coppersmith and Vinograd [60]), and by assuming constant time for the generation of a random number, this randomized algorithm is faster than the best existing deterministic algorithm, at least in the case of dense graphs. This procedure does, however, not compute a perfect matching, it just decides - with an arbitrarily small probability of error - whether the given graph has one. To compute a perfect matching in G we may delete edges of G and test whether the remaining graph has a perfect matching. This procedure is repeated until no further edges can be deleted without destroying all perfect matchings in the remaining graph. Obviously, this remaining graph is then a perfect matching. Instead of this naive algorithm Mülmuley, Vazirani and Vazirani [129] propose a parallel algorithm which performs a matrix inversion only *once*. The basic idea relies on assigning random weights to the edges of the graph and on generating the random variables x_{ij} in the Tutte matrix so that the perfect matching with minimum weight is unique. Then, an easy-to-check condition is given for the membership of a certain edge to this minimum weighted perfect matching. This algorithm requires $O(|V|^{3.5}|E|)$ processors to find a perfect matching in $O(\log^2 |V|)$ time.

Cechlárová [57] gives a necessary and sufficient condition that a bipartite graph G contains a *unique* perfect matching. This is the case, if the rows and columns of the adjacency matrix can be permuted by ϕ such that in the permuted matrix $(a_{\phi(i)\phi(j)})$ all 1 entries lie on the main diagonal or below. Cechlárová shows that graphs with a unique perfect matching can be recognized in $O(|E|)$ time and the same time complexity is needed for constructing the unique perfect matching.

The maximum matching problem is particularly easy to solve in so-called *convex bipartite graphs* (Glover [89]). A bipartite graph $G = (V, W; E)$ with vertex sets V and W and edge set E is *convex*, if for all $i, k, i < k$, and j

$$(i, j), (k, j) \in E \Rightarrow \text{all } (i+1, j), \dots, (k-1, j) \in E.$$

Glover described the following simple algorithm for finding a maximum matching in convex bipartite graphs:

1. Compute for every $j \in W$ the label $\pi(j) := \max \{i | (i, j) \in E\}$.
2. Start with the empty matching M .
For $i = 1, 2, \dots, |V|$ do:
If there exists an edge (i, j) with unmatched vertex j , then enlarge the matching by the edge (i, k) with minimum value $\pi(k)$.

A straightforward implementation of this algorithm has complexity $O(n^2)$. Using a method of Gabow and Tarjan [86] the maximum matching can even be found in linear time. Matchings in convex graphs are used in the solution of so-called *terminal assignment problems*. Terminal assignment problems arise in connection with the design of integrated circuits. They can be described as follows: each of n entry terminals positioned on the upper layer of a channel has to be assigned to one of the m exit terminals positioned in the lower layer. The density $d(x)$ of a given assignment at position x is the number of terminals in the upper layer, left to x , which are connected to an exit in the lower layer right to position x , see Fig. 2. The density of an assignment is the maximum of $d(x)$ taken over all positions x . If there are two upper layers, the density is computed for each upper layer separately and the total density is the maximum of these two densities. In the terminal assignment problem the terminals situated in two upper layers are to be assigned such that the resulting channel routing problem has minimum total density. Rote and Rendl [158] solve the 2-layer terminal assignment problem by a recursive algorithm in linear time using matchings in convex bipartite graphs.

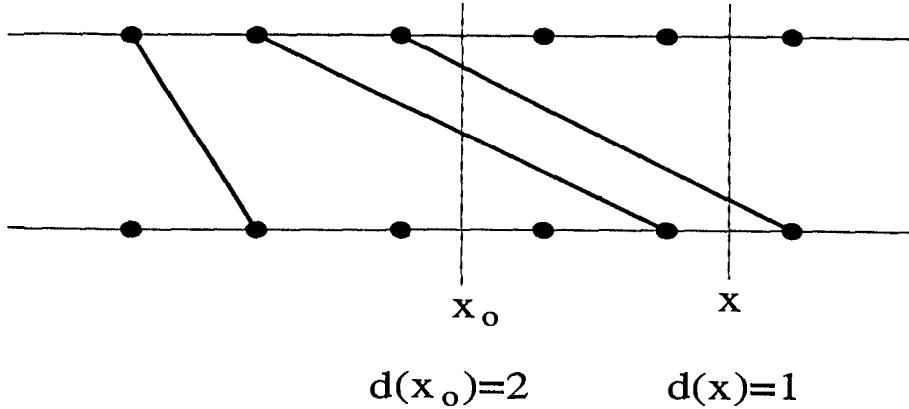


Figure 2: An instance of the terminal assignment problem. The density of this assignment is 2 and is attained at position x_0 , $d(x_0) = 2$.

Flows in networks offer another model to describe assignments. Let G be the bipartite graph introduced before. We embed G in the network $\mathcal{N} = (N, A, c)$ with *node set* N , *arc set* A and *arc capacities* c . The node set N consists of a *source* s , a *sink* t and the vertices of $V \cup W$. The source is connected to every node in V by an arc of capacity 1, every node in W is connected to the sink by an arc of capacity 1, and every edge in E is directed from V to W and supplied with an infinite capacity. A *flow* in network \mathcal{N} is a function $f : A \rightarrow \mathbb{R}$ with

$$\sum_{x \in V \cup W} f(x, y) = \sum_{x \in V \cup W} f(y, x), \quad \text{for all } y \in V \cup W \quad (2)$$

$$0 \leq f(x, y) \leq c(x, y), \quad \text{for all } (x, y) \in A \quad (3)$$

Equalities (2) and (3) are called *flow conservation constraints* and *capacity constraints*, respectively. The *value* $z(f)$ of the flow f is defined as

$$z(f) := \sum_{x \in N} f(s, x).$$

The *maximum network flow problem* asks for a flow with maximum value $z(f)$. Obviously, a maximum integral flow in the special network constructed above corresponds to a matching with maximum cardinality. A *cut* in the network \mathcal{N} is a subset C of the node set N with $s \in C$ and $t \notin C$. The value

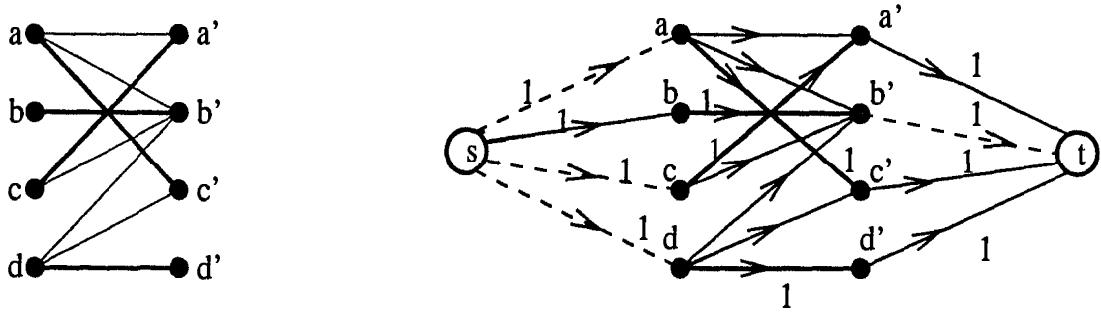


Figure 3: Perfect matching in a bipartite graph and corresponding network flow model. A minimum cut is given by $\{s, b, b'\}$. The dashed edges contribute to the value of the cut.

$u(C)$ of cut C is defined as

$$u(C) := \sum_{\substack{x \in C, y \notin C \\ (x, y) \in A}} c(x, y).$$

Ford and Fulkerson's famous *Max Flow-Min Cut Theorem* [85] states that the value of a maximum flow equals the minimum value of a cut.

This leads immediately to the matching theorem of König [116]. Given a bipartite graph G , a *vertex cover* (cut) in G is a subset of its vertices such that every edge is incident with at least one vertex in this set.

Theorem 1.5 (König's Matching Theorem, 1931)

In a bipartite graph the minimum number of vertices in a vertex cover equals the maximum cardinality of a matching.

Let us now translate this theorem in the language of 0-1 matrices. Given a bipartite graph $G = (V, W; E)$ with $|V| = |W| = n$, we define an $n \times n$ 0-1 matrix $B = (b_{ij})$ as follows

$$b_{ij} := \begin{cases} 0 & \text{if } (i, j) \in E \\ 1 & \text{if } (i, j) \notin E \end{cases}$$

A *zero-cover* is a subset of rows (columns) of matrix B which contains all 0 elements. A row (column) which is an element of a zero-cover is called a *covered row* (*covered column*). Now we get

Theorem 1.6 *There exists an assignment ϕ with $b_{i\phi(i)} = 0$ for all $i = 1, \dots, n$, if and only if the minimum zero cover has n elements.*



Figure 4: 0-1 matrix model of a bipartite graph and the corresponding minimum vertex cover which corresponds to the cut in Fig. 3. An assignment is given by the entries marked by *.

A maximum matching corresponds uniquely to a maximum flow in the corresponding network \mathcal{N} , and therefore to a minimum cut C in this network. From this minimum cut we can construct a zero-cover in the adjacency matrix: if node $i \in V$ of the network does not belong to cut C , then row i is an element of the zero-cover. Analogously, if node $j \in W$ belongs to cut C , then column j is an element of the zero-cover.

2 Linear Sum Assignment Problems (LSAPs)

In this section we deal with the classical *linear sum assignment problem* (LSAP). We first introduce the problem and some alternative interpretations, and then discuss some applications of the LSAP.

Let us return to the original model where n items (e.g. jobs) are to be assigned to n machines (or workers) in the best possible way. Let us assume that the assignment of job i to machine j incurs a cost c_{ij} . In order to complete all jobs in the cheapest possible way, we have to minimize the objective function

$$\sum_{i=1}^n c_{i\phi(i)}, \quad (4)$$

i.e., we have to find an assignment ϕ^* which minimizes (4). Thus, we get the *linear sum assignment problem* (LSAP) as

$$\min \sum_{i=1}^n c_{i\phi(i)}. \quad (5)$$

Based on the description (1) of the set of all assignments (see Section 1), the LSAP can also be formulated as a 0-1 linear program:

$$\begin{aligned} \min \quad & \sum_{i=1}^n c_{ij} x_{ij} \\ & \sum_{i=1}^n x_{ij} = 1 \quad j = 1, \dots, n \\ & \sum_{j=1}^n x_{ij} = 1 \quad i = 1, \dots, n \\ & x_{ij} \in \{0, 1\} \quad i, j = 1, \dots, n. \end{aligned} \tag{6}$$

Due to Theorem 1.1 we can relax the conditions $x_{ij} \in \{0, 1\}$ to $x_{ij} \geq 0$ and solve the corresponding linear program, denoted by LP: any basic solution of this linear program corresponds to a permutation matrix. We will see in Section 3 that LP-based algorithms belong to the most important solution techniques for the LSAP. Often they tackle the dual linear program: Let u_i , $1 \leq i \leq n$, and v_j , $1 \leq j \leq n$, be dual variables. Then the dual of the linear program LP is given by

$$\begin{aligned} \max \quad & \sum_{i=1}^n u_i + \sum_{j=1}^n v_j \\ & u_i + v_j \leq c_{ij} \quad i, j = 1, \dots, n \\ & u_i, v_j \in \mathbb{R} \quad i, j = 1, \dots, n. \end{aligned} \tag{7}$$

Keeping in mind the matching interpretation of assignments, the LSAP can be equivalently formulated as a matching problem: given a bipartite graph $G = (V, W; E)$ with edge weights c_{ij} for all edges $(i, j) \in E$, find a perfect matching with minimum weight:

$$\min \left\{ \sum_{(i,j) \in \mathcal{M}} c(i, j) : \mathcal{M} \text{ is a perfect matching} \right\}.$$

The LSAP can be solved efficiently, and the design of efficient solution methods for this problem has been an object of research for many years. There exists an amazing amount of algorithms for the LSAP, ranging from the simple Hungarian method to more recent developments involving sophisticated data structures and including parallel algorithms. (See Section 3 for

a more detailed discussion of these topics). There are, however, some simple cases where an optimal solution of the LSAP can be given before hand, even if the coefficients c_{ij} of the cost matrix are not known. These cases are related to *Monge properties* of the cost matrix C .

A matrix $C = (c_{ij})$ is said to be a *Monge matrix* (cf. Hoffman [102]), if it fulfills the following conditions:

$$c_{ij} + c_{kl} \leq c_{il} + c_{kj}, \quad \text{for } 1 \leq i < k \leq n, \quad 1 \leq j < l \leq n. \quad (8)$$

It is easy to show that the identical permutation is an optimal solution of an LSAP with a cost matrix which fulfills (8). This remains true, even if condition (8) is relaxed to the so-called *weak Monge property* (cf. Derigs, Goecke and Schrader [69])

$$c_{ii} + c_{kl} \leq c_{il} + c_{ki}, \quad \text{for } 1 \leq i < k \leq n, \quad 1 \leq i < l \leq n. \quad (9)$$

Analogously, it can be shown that the permutation ϕ defined by $\phi(i) = n - i + 1$ for all i , is an optimal solution of an LSAP with cost matrix C fulfilling the *inverse Monge property*:

$$c_{ij} + c_{kl} \geq c_{il} + c_{kj} \quad \text{for } 1 \leq i < k \leq n, \quad 1 \leq j < l \leq n. \quad (10)$$

Obviously, Monge properties depend on the proper numbering of the rows and columns of the matrix. A matrix C is called a *permuted Monge matrix*, if there exists a pair of permutations (ϕ, ψ) such that the matrix $C^{(\phi, \psi)} = (c_{\phi(i)\psi(j)})$ obtained from C by permuting its rows by ϕ and its columns by ψ , is a Monge matrix, see Burkard, Klinz and Rudolf [42]. Due to a result of Deĭneko and Filonenko [66] it can be recognized in $O(n^2)$ time, whether an $n \times n$ matrix C is a permuted Monge matrix. Moreover, the appropriate permutations ϕ, ψ for the rows and the columns can also be found within this time bound. As a consequence, the LSAP with a permuted Monge cost matrix can be solved in $O(n^2)$ time. The reader is referred to Burkard et al. [42] for a detailed discussion of Monge properties, and a description of the algorithm of Deĭneko and Filonenko.

An important special case of the LSAP with permuted Monge cost matrix arises if the cost coefficients have the form

$$c_{ij} = u_i v_j$$

with nonnegative numbers u_i and v_j . Such an LSAP can simply be solved in $O(n \log n)$ time by ordering the elements u_i and v_j .

Theorem 2.1 (Hardy, Littlewood, and Pólya [101], 1952)

Let $0 \leq u_1 \leq \dots \leq u_n$ and $0 \leq v_1 \leq \dots \leq v_n$. Then for any permutation ϕ

$$\sum_{i=1}^n u_i v_{n+1-i} \leq \sum_{i=1}^n u_i v_{\phi(i)} \leq \sum_{i=1}^n u_i v_i.$$

Let us now discuss some applications of the linear sum assignment problem. LSAPs occur quite frequently as subproblems in more involved applied combinatorial optimization problems like the quadratic assignment problem, traveling salesman problems or vehicle routing problems. Classical direct applications include the assignment of personnel (cf. e.g. Machol [122], Ewashko and Dudding [78]), and the assignment of jobs to parallel machines, but LSAPs have quite a number of further interesting applications. Neng [132] modeled the optimal engine scheduling in railway systems as an assignment problem. Further applications in the area of vehicle and crew scheduling are described by Carraresi and Gallo [49].

Another application concerns problems in telecommunication, in particular in earth - satellite systems. Usually the data to be remitted are buffered in the ground stations and are sent in very short data bursts to the satellite. There they are received by transponders and then, are transmitted again to the receiving earth station. A transponder just connects one sending station with a receiving station. For a fixed time interval of variable length λ_k the n sending stations are connected with the receiving stations via the n transponders onboard the satellite, i.e., a certain switch mode is applied. Mathematically, a *switch mode* P_k corresponds to a permutation matrix. Then the connections onboard the satellite are changed simultaneously and in the next time interval new pairs of sending and receiving stations are connected via transponders, i.e., a new switch mode is applied. Given an $(n \times n)$ traffic matrix $T = (t_{ij})$, where t_{ij} describes the amount of information to be remitted from the i -th sending station to the j -th receiving station, we have to determine the switch modes P_k , $k = 1, 2, \dots$, and the nonnegative lengths λ_k of the corresponding time slots during which the switch modes P_k are applied, such that all data are remitted in the shortest possible time. Summarizing, our problem consists of finding suited switch modes and corresponding lengths for the time slots during which the switch

modes are applied. This leads to the following model:

$$\begin{aligned} \min \quad & \sum_k \lambda_k \\ \text{s.t.} \quad & \sum_k \lambda_k p_{ij}^{(k)} \geq t_{ij}, \quad \text{for } 1 \leq i, j \leq n \\ & \lambda_k \geq 0, \quad \text{for all } k \end{aligned} \tag{11}$$

Since here the time is split among the participating stations, all of them having access to the satellite, such a traffic model is called time-division-multiple-access (TDMA) model. The problem can be transformed to an LSAP, if a system of $2n$ fixed switch modes $P_1, \dots, P_n, Q_1, \dots, Q_n$ is installed onboard the satellite, where

$$\sum_k P_k = \sum_l Q_l = 1$$

and $\mathbf{1}$ is the matrix with 1-entries only. For details see Burkard [38].

Brogan [37] describes assignment problems in connection with locating objects in space. Let us consider n objects which are detected by two sensors at geographically different sites. Each sensor measures the angle under which the object can be seen, i.e., it provides n lines, on which the objects lie. The location of every object is found by intersecting the appropriate lines. The pairing of the lines is modeled as follows: let c_{ij} be the smallest distance between the i -th line from sensor 1 and the j -th line from sensor 2. Due to small errors during the measurements, c_{ij} might even be greater than 0 if some object is determined by lines i and j . Solving an LSAP with cost matrix $C = (c_{ij})$ leads to very good results in practice. A similar technique can be used for tracking n moving objects (e.g. missiles) in space. If their locations at two different times t_1 and t_2 are known, we compute the (squared) Euclidean distances between any pair of old and new locations and solve the corresponding assignment problem in order to match the points in the right way.

For some further applications of assignment problems, including the optimal depletion of inventory, see Ahuja, Magnanti, Orlin and Reddy [2].

There are also other possibilities for the choice of the objective function in assignment problems. For example, if c_{ij} is the time needed for machine j to complete job i and all machines run in parallel, the shortest completion time for all jobs is given by $\max_{1 \leq i \leq n} \{c_{i\phi(i)}\}$. This leads to the *linear bottleneck assignment problem* (LBAP) which will be treated in Section 5. Other objective functions will be discussed in Section 6.

3 Algorithms for the LSAP

A large number of algorithms, sequential and parallel, have been developed for the linear sum assignment problem (LSAP). They range from primal-dual combinatorial algorithms, to simplex-like methods, cost operation algorithms, forest algorithms, and relaxation approaches. The worst-case complexity of the best sequential algorithms for the LSAP is $O(n^3)$, where n is the size of the problem. There is a number of survey papers and books on algorithms, e.g. Derigs [68], Martello and Toth [126], Bertsekas [30], Akgül [6], and the book on the first DIMACS challenge edited by Johnson and McGeoch [106]. Among papers reporting on computational experience we mention Derigs [68], Carpaneto, Martello and Toth [51], Carpaneto and Toth [56], Lee and Orlin [121], and some of the papers in [106].

Most sequential algorithms for the LSAP can be classified into primal dual algorithms and simplex-based algorithms. Primal-dual algorithms work with a pair consisting of an infeasible primal and a feasible dual solution which fulfill the complementarity slackness conditions. Such algorithms update the solutions iteratively until the primal solution becomes feasible, while keeping the complementary slackness conditions fulfilled. At this point the primal solution is also optimal, according to duality theory.

Simplex-based algorithms are special implementations of the primal and the dual simplex algorithm for linear programming. These algorithms can be applied to solve the LSAP due to Theorem 1.1. Although simplex algorithms are not polynomial in general, there are special polynomial implementations of them in the case of the LSAP.

More recently, an interior point algorithm has been applied to the LSAP, see Ramakrishnan, Karmarkar, and Kamath [155]. This algorithm produces promising results, in particular when applied to large size instances.

From the computational point of view very large scale dense assignment problems with about 10^6 nodes can be solved within a couple of minutes by sequential algorithms, see [121].

A number of parallel approaches developed for the LSAP show that the speed up achieved by such algorithms is limited by the sparsity of the cost matrices and/or the decreasing load across the iterations, see e.g. Balas, Miller, Pekny, and Toth [14], Bertsekas and Castañon [31, 32], Castañon, Smith, and Wilson [53], as well as Wein and Zenios [170, 171]. More information on these topics is given in Section 3.4.

3.1 Primal-Dual Algorithms

Primal-dual algorithms work with a pair of an unfeasible primal solution x_{ij} , $x_{ij} \in \{0, 1\}$, $1 \leq i, j \leq n$, and a feasible dual solution u_i, v_j , $1 \leq i, j \leq n$, which fulfill the complementary slackness conditions:

$$x_{ij}(c_{ij} - u_i - v_j) = 0, \quad \text{for } 1 \leq i, j \leq n \quad (12)$$

We denote $\bar{c}_{ij} := c_{ij} - u_i - v_j$ and call \bar{c}_{ij} *reduced costs* with respect to the dual solution u_i, v_j . This pair of solutions is updated iteratively by means of so-called *admissible transformations* of the costs matrix C . Also the process of determining the first pair of primal and dual solutions can be seen as an admissible transformation. A transformation T of a matrix $C = (c_{ij})$ to a matrix $\tilde{C} = (\tilde{c}_{ij})$ is called admissible if for each permutation ϕ of $1, 2, \dots, n$ the following equality holds:

$$\sum_{i=1}^n c_{i\phi(i)} = \sum_{i=1}^n \tilde{c}_{i\phi(i)} + z(T),$$

where $z(T)$ is a constant which depends only on the transformation T . Clearly, an admissible transformation does not change the order of the permutations (feasible solutions of the LSAP) with respect to the value of the objective function. Hence the LSAPs with cost matrices C and \tilde{C} are equivalent. We may assume without loss of generality that the entries of C are nonnegative. (Otherwise we could add a large enough constant to all entries of C . This results in just adding a constant to the objective function, and hence, does not change the optimal solution of the problem but only its optimal value.)

If all entries \tilde{c}_{ij} of matrix \tilde{C} are nonnegative and there exists a permutation ϕ^* such that $\tilde{c}_{i\phi^*(i)} = 0$, then ϕ^* is an optimal solution of the original LSAP with cost matrix C . The optimal value is equal to $z(T)$, where T is the transformation from C to \tilde{C} .

Primal-dual algorithms differ on 1) the way they obtain their first pair of a primal and a dual solution fulfilling the complementary slackness conditions, and 2) the implementation of the update of dual variables.

The Hungarian method

The first polynomial-time primal-dual algorithm for the LSAP is the Hungarian method due to Kuhn [118]. In the Hungarian method a starting dual

solution is obtained by so-called *row and column reductions*:

$$u_i = \min\{c_{ij}: 1 \leq j \leq n\}, \quad \text{for } 1 \leq i \leq n,$$

and then

$$v_j = \min\{c_{ij} - u_i: 1 \leq i \leq n\} \quad \text{for } 1 \leq j \leq n.$$

An infeasible primal start solution is obtained by finding a matching \mathcal{M} of maximal cardinality in the graph $\bar{G} = (V, W; \bar{E})$, where $V = W = \{1, 2, \dots, n\}$, and $\bar{E} = \{(i, j): \bar{c}_{ij} = c_{ij} - u_i - v_j = 0\}$. Then we set $x_{ij} = 1$ if (i, j) is an edge of the matching \mathcal{M} , and $x_{ij} = 0$, otherwise. Clearly, the primal solution x_{ij} , $1 \leq i, j \leq n$, and the dual solution u_i , v_j , $1 \leq i, j \leq n$, fulfill the complementarity slackness conditions.

According to Theorem 1.5, the matching \mathcal{M} corresponds to a minimal vertex cover of graph \bar{G} , or equivalently to a minimal zero-cover of the matrix. Let I be the set of row indices of uncovered rows and let J contain the indices of uncovered columns. Then the dual variables are updated by setting

$$u_i := \begin{cases} u_i - \delta & \text{if } i \in I \\ u_i & \text{otherwise} \end{cases} \quad v_j := \begin{cases} v_j & \text{if } j \in J \\ v_j + \delta & \text{otherwise} \end{cases},$$

where δ is defined as minimum of the currently uncovered reduced costs, i.e., $\delta = \min\{\bar{c}_{ij}: i \in I, j \in J\}$. This corresponds to an admissible transformation T given by

$$\tilde{c}_{ij} := \begin{cases} c_{ij} - \delta & \text{if } i \in I, j \in J \\ c_{ij} + \delta & \text{if } i \notin I, j \notin J \\ c_{ij} & \text{otherwise} \end{cases}. \quad (13)$$

Its constant $z(T)$ is equal to $\delta(|I| + |J| - n)$. This transformation preserves the nonnegativity of the reduced cost matrix (or equivalently, the feasibility of the dual solution). According to Theorem 1.6, the constant $z(T)$ equals zero ($|I| + |J| = n$), if and only if there is a permutation ϕ such that $c_{i\phi(i)} = 0$, for $1 \leq i \leq n$. In the latter case we have a feasible primal solution, and hence also an optimal one.

One possibility to implement the Hungarian method so that it achieves a worst case time complexity of $O(n^3)$ is by means of so-called *alternating trees*. Such a tree contains *matched edges* - which are edges from \mathcal{M} , and *free edges* which do not belong to \mathcal{M} . Analogously, *free nodes* are nodes which are adjacent only to free edges, and *matched nodes* are adjacent to one matched edge.

Definition 3.1 An alternating tree is a directed tree rooted at some free node $r \in V$, where all arcs are directed away from the root and

- all matched edges are directed from W to V and all free edges are directed from V to W ,
- the root r is the only free node,
- all leafs are nodes from V .

Starting from a maximal matching in graph \bar{G} the algorithm builds an alternating tree in \bar{G} . As soon as this tree reaches a leaf $v \in V$ which is adjacent to some free node in W , there is an *alternating path* between the free nodes v and w , i.e., there is a path joining v and w in G whose edges are alternatively free and matched edges. Such a path is called an *augmenting path*, because by exchanging the free and matched edges in it we get a new maximal matching in \bar{G} , whose cardinality is larger than that of the previous maximal matching. In this case we can increase the number of primal variables x_{ij} which are equal to 1, and thus *improve* the primal solution without changing the dual one. There may be at most n such augmentations because the cardinality of a matching cannot exceed n . After having performed all possible augmentation steps we get finally a *maximal alternating tree*, i.e., an alternating tree where all leafs have cardinality equal to 1 in \bar{G} . At this point, the dual variables - and with them the reduced costs and the graph \bar{G} - are updated by an admissible transformation. Thus some new edges are added to \bar{G} , i.e., additional reduced costs equal to 0 arise. Now the existing alternating tree can be updated with respect to the new graph \bar{G} . It is possible to implement the update of dual variables so that all due updates between two consecutive augmentations are performed by applying $O(n^2)$ elementary operations. Since there may be at most n augmentations, this leads to an $O(n^3)$ algorithm.

Shortest augmenting path algorithms

Although the Hungarian algorithm described above grows an alternating tree to augment the primal solution, in principle only alternating augmenting paths are used. This observation is the conceptual basis of numerous *shortest augmenting path algorithms* for the LSAP, which belong to the class of primal-dual methods and can also be implemented in $O(n^3)$ steps. Given a pair of a primal solution x_{ij} , $1 \leq i, j \leq n$, and a dual feasible solution u_i, v_j , $1 \leq i, j \leq n$, which fulfill the complementarity slackness conditions, together

with the reduced costs \bar{c}_{ij} , construct a weighted directed bipartite graph $\tilde{G} = (V, W; \tilde{E})$ with arc set $\tilde{E} = D \cup R$. Here $D = \{(i, j) : (i, j) \in E, x_{ij} = 0\}$ is the set of *forward arcs*, and $R = \{(j, i) : (i, j) \in \bar{E}, x_{ij} = 1\}$ is the set of *backward arcs*. The weights of the backward arcs are set equal to 0, whereas the weights of the forward arcs are set equal to the corresponding reduced costs. We select a free node r in V and solve the single-source shortest path problem, i.e., compute the shortest paths from r to all nodes of \tilde{G} . The shortest among all paths from r to some free node in W is used to augment the current primal solution by swapping the free and matched edges. The primal and dual solutions as well as the reduced costs are updated similarly as in the case of the Hungarian method. It can be shown that each (unfeasible) primal solution (or equivalently, each matching in \tilde{G}) constructed in the course of the algorithm has minimum weight among all solutions of the same cardinality, see e.g. Lawler [119], Derigs [68]. Hence, after n augmentations we obtain an optimal primal solution. The single-source shortest paths can be computed in $O(n^2)$ time in a graph with nonnegative weights, e.g. by applying the Dijkstra algorithm implemented with Fibonacci heaps, see Fredman and Tarjan [80]. This implies that the overall complexity of augmenting path algorithms amounts to $O(n^3)$.

The shortest path computation may stop as soon as the first path joining r with some free node in W has been computed.

Basically shortest augmenting path algorithms for the LSAP may differ in the way they determine a starting pair of primal and dual solutions, and by the subroutine they use for computing the shortest paths. (The reader is referred to Cherkassky, Goldberg, and Radzik [59] for a review on shortest paths algorithms. For developments on data structures used in shortest path algorithms the reader is referred to Fredman and Tarjan [80], and to Ahuja, Mehlhorn, Orlin, and Tarjan [3].) Most of the existing algorithms use the Dijkstra algorithm for the shortest path computations, e.g. Carpaneto and Toth [56], Jonker and Volgenant [107, 108], and Volgenant [167].

Several authors try to speed up the shortest path computations by *thinning out* the underlying bipartite graph, e.g. Carraresi and Sodini [52], Glover, Glover, and Klingman [90]. In these algorithms only “short” edges of the graph \tilde{G} are considered in the shortest path computations. A *threshold value* which is updated after each augmentation is used to determine the *short* edges. Therefore these algorithms are sometimes called *threshold algorithms*. Their worst case complexity is also $O(n^3)$.

Some shortest augmenting path algorithms apply specific implementations of shortest paths computations especially suited for sparse matrices, see [56],

and [167]. To apply this technique the cost matrix is thinned-out by some heuristic. If no feasible primal solution is found for the sparse problem, then the cost matrix is enlarged by re-adding some of the “canceled” entries.

The auction algorithm

Another primal-dual algorithm, called *the auction algorithm*, has been proposed by Bertsekas [28]. The auction algorithm is similar to the Hungarian method to the extent that the updates of primal and dual solutions can be modeled by admissible transformations. However, whereas in the Hungarian method the value of the dual objective function increases after each such an update, in auction algorithms it is only guaranteed that this value does not decrease. Another difference is that in the Hungarian algorithm a matched vertex of V or W remains matched for the rest of the algorithm, while this property only holds for vertices of V in the case of the auction algorithm. Some vertex of W matched in a certain iteration of the auction algorithm may become free in some further iteration. The original auction algorithm was developed for maximization problems, where we try to maximize the objective function of the LSAP under the assignment constraints. In the minimization case the auction algorithm associates each vertex of W with an item to be sold, and each vertex of V with a customer. c_{ij} is the cost of item j for customer i , the dual variable v_j is the profit on item j , and the difference $c_{ij} - v_j$ is customer's i cost margin relative to the profit on item j . The algorithm starts with a pair of an (unfeasible) primal solution x_{ij} (e.g. $x_{ij} = 0$ for all i, j) and a feasible dual solution u_i, v_j . This pair of solutions fulfills the complementarity slackness conditions (12), and is updated iteratively by maintaining (12). The update of the current x_{ij}, u_i, v_j , to obtain the subsequent pair of solutions x'_{ij}, u'_i, v'_j , is made by choosing a free customer \bar{i} ($\bar{i} \in V$), and by determining the smallest and second smallest cost margin \bar{u} and \tilde{u} , respectively:

$$\bar{u} = \min\{c_{\bar{i}j} - v_j : 1 \leq j \leq n\}, \quad \bar{j} = \operatorname{argmin}\{c_{\bar{i}j} - v_j : 1 \leq j \leq n\},$$

$$\tilde{u} = \min\{c_{\bar{i}j} - v_j : 1 \leq j \leq n, j \neq \bar{j}\}.$$

Then the algorithm distinguishes two cases: Case 1, where $\bar{u} < \tilde{u}$, or $\bar{u} = \tilde{u}$ and \bar{j} is free, and Case 2, where $\bar{u} = \tilde{u}$ and \bar{j} is matched. In the first case the algorithm assigns \bar{i} to \bar{j} ($x'_{\bar{i}\bar{j}} = 1$) and updates the dual variables $u_{\bar{i}}$ and $v_{\bar{j}}$ so as to preserve dual feasibility: $u'_{\bar{i}} = \tilde{u}$ and $v'_{\bar{j}} = v_{\bar{j}} + (\bar{u} - \tilde{u})$. If \bar{j} was not a free vertex, i.e., there was some \hat{i} such that $x'_{\hat{i}\bar{j}} = 1$, the new

primal solution sets $x'_{\tilde{i}\tilde{j}} = 0$. In the second case, where $\bar{u} = \tilde{u}$ and there is an index \hat{i} such that $x_{\hat{i}\tilde{j}} = 1$, the algorithm sets $u_{\hat{i}} := \bar{u}$ and performs a labeling procedure to either update the dual variables so as to assign \tilde{j} to \hat{i} or to get an augmenting path starting from \hat{i} and ending at some free vertex in W . In the latter case we would have an update of the pair of primal and dual solutions as in the case of the Hungarian method (see [28] for a complete description). This auction algorithm is pseudopolynomial and runs in $O(n^2(n+R))$ time, where $R = \max\{|c_{ij}| : 1 \leq i, j \leq n\}$. The similarity with the Hungarian method allows the combination of both algorithms as proposed in [28]. The combined algorithm starts by applying the auction algorithm, counts the iterations which do not lead to an increase of the number of items assigned to customers, and switches to the Hungarian method as soon as the counter exceeds an appropriately defined parameter. It can be shown that the worst case complexity of this combination is $O(n^3)$.

Bertsekas and Eckstein [35] modified the above described auction algorithm to obtain a polynomial-time algorithm with worst case complexity $O(n^3 \log(nR))$, where R is defined as above. The algorithm works with a pair of a primal and a dual solution which fulfills the ϵ -complementarity conditions

$$x_{ij}(c_{ij} - u_i - v_j) \geq -\epsilon, \text{ for } 1 \leq i, j \leq n.$$

In terms of the ϵ -complementarity conditions an ϵ -relaxed problem is defined: It consists of finding a pair of feasible primal and dual solutions of the given LSAP which fulfills the ϵ -complementarity conditions. It can be shown that an optimal solution of the $1/n$ -relaxed problem ($\epsilon = \frac{1}{n}$) is also an optimal solution of the initial problem. This fact suggests to apply so-called *scaling techniques*, i.e., to solve initially the ϵ -relaxed problem for a large ϵ , and then to decrease ϵ gradually until it reaches $1/n$.

When ϵ is decreased to ϵ' the optimal solution of the previous ϵ -relaxed problem is updated with “few” efforts to obtain the optimal solution of the ϵ' -relaxed problem. Bertsekas et al. [35] implement this ϵ -scaling in terms of a scaling of the costs. All costs c_{ij} are multiplied by $n+1$ and a 1 -relaxed problem ($\epsilon = 1$) is solved. If x_{ij} and u_i, v_j are optimal primal and dual solutions of the 1 -relaxed problem with costs $c'_{ij} = (n+1)c_{ij}$, then x_{ij} and $\frac{u_i}{n+1}, \frac{v_j}{n+1}$, is an optimal pair of primal and dual solutions for the $\frac{1}{n}$ -relaxed problem with costs c_{ij} . Consequently, x_{ij} is an optimal solution of the LSAP with costs c_{ij} . The 1 -relaxed subproblem with costs c'_{ij} is solved by solving $M = O(\log(nR))$ subproblems consecutively. The m -th subproblem is a

ϵ -relaxed LSAP with costs given by $\frac{c'_{ij}}{2M-m}$, and is solved by an algorithm similar to the original auction algorithm [28]. For a free node $\bar{i} \in V$, the values \bar{u} , \tilde{u} , and the index $\bar{j} \in W$ are determined as in the original auction algorithm, and two analogous cases are distinguished. The dual variable corresponding to \bar{i} is set to $u_{\bar{i}} := \tilde{u} + \frac{\epsilon}{2}$, and the dual variable corresponding to \bar{j} is set to $v_{\bar{j}} := v_{\bar{j}} + (\bar{u} - \tilde{u}) - \frac{\epsilon}{2}$.

Bertsekas et al. [35] call *bidding* the update of $u_{\bar{i}}$. The version where *one* node bids at a time is called Gauss–Seidel version of the auction algorithm and is better suited for sequential implementations (see [35] for more information). Another version, where *all* free customers bid simultaneously is called Jacobi version of the auction algorithm and is more appropriate for parallel implementations, see Bertsekas [29].

Pseudoflow algorithms

Other primal-dual algorithms for the LSAP are based on the formulation of the LSAP as a minimum cost flow problem (see Section 1). A *pseudoflow* is a flow which fulfills the capacity constraints (3), but does not necessarily fulfill the flow conservation constraints (2). Pseudoflow algorithms work with ϵ -relaxations of the minimum cost flow problem where the flow conservation constraints are violated by at most ϵ . They iteratively transform the starting solution, which is a pseudoflow, into an optimal solution for the ϵ -relaxed problem. Then, ϵ is decreased and the procedure is repeated until ϵ reaches $\frac{1}{n}$. At that point an optimal solution of the ϵ -relaxed flow problem is also an optimal solution for the original flow problem, and hence for the LSAP. The general idea is quite similar to the auction algorithm of Bertsekas et al. [35] which involves ϵ -complementarity conditions. One difference is that pseudoflow algorithms apply ϵ scaling, whereas the auction algorithm in [35] applies cost scaling. Another difference concerns the transformation of a pseudoflow into an optimal solution of the ϵ -relaxed flow problem, which is done by the push-relabel method described by different authors, e.g. Goldberg and Tarjan [96], Cherkassky and Goldberg [58]. If ϵ is divided by $\delta \geq 2$ in each iteration, it can be shown that the push relabel algorithm terminates after $O(\log_\delta(nR))$ iterations, where R is defined as in the previous section. Different pseudoflow algorithms for the LSAP have been developed by Orlin and Ahuja [135], Goldberg, Plotkin and Vaidya [95], and Goldberg and Kennedy [94]. These algorithms are particularly efficient for sparse problems: the worst-case complexity of the first two algorithms is $O(\sqrt{nm} \log(nR))$, where m is the number of finite entries of the cost ma-

trix C . (Infinite entries correspond to non-existing edges in the matching model.)

3.2 Simplex-Based Algorithms

Simplex-based algorithms for the LSAP are specific implementations of the network simplex algorithm. The latter is a specialization of the simplex method for linear programming to network problems and is due to Dantzig, see e.g. [65]. The specialization relies on exploiting the combinatorial structure of network problems to perform more efficient pivots acting on trees rather than on the coefficient matrix. A number of such algorithms for the LSAP, and more generally for the transportation problem of whom the LSAP is a special case, have been developed in the early 1970s, see e.g. Glover, Karney and Klingman [91], Glover and Klingman [92]. It is well known that there is a one-to-one correspondence between primal (integer) basic solutions of the LSAP and spanning trees of the bipartite graph G related to assignment problems as described in Section 1. Moreover, given the spanning tree, the values of the corresponding dual variables fulfilling the complementarity slackness conditions are uniquely determined, as soon as the value of one of those variables is fixed (arbitrarily). Every primal feasible basic solution is highly degenerate because it contains $2n - 1$ variables and $n - 1$ of them are equal to 0. Hence degeneracy poses a problem, and the first simplex-based algorithms for the LSAP were exponential. The first steps towards the design of polynomial-time simplex-based algorithms were made by introducing the concept of so-called *strongly feasible trees*, due to Cunningham [62, 63] and Barr, Glover, and Klingman [26].

Primal simplex-based algorithms

Primal simplex-based algorithms work with a primal feasible basic solution which corresponds to a spanning tree T in G . n of the x variables corresponding to edges of the tree are equal to 1, the other $n - 1$ edges of the tree correspond to x variables equal to 0. Given the primal variables x_{ij} , the dual variables u_i, v_j are uniquely determined by setting one of them arbitrarily and computing the others such that the complementarity conditions are fulfilled, i.e., $\bar{c}_{ij} = c_{ij} - u_i - v_j = 0$ for all i, j such that $(i, j) \in T$. If the reduced costs corresponding to the edges of the co-tree $T^c = E \setminus T$ are nonnegative, the current basis T is optimal. Otherwise, there is an edge $(i, j) = e \in T^c$ with $\bar{c}_{ij} < 0$. Adding this edge to T creates a unique cycle

$C(T, e)$. The subsequent basis T' is given by $T' := (T \setminus \{f\}) \cup \{e\}$, where $f = (i', j')$ is some edge in $C(T, e)$, with $x_{i'j'} = \min\{x_{ij} : (i, j) \in C(T, e)\}$.

Barr et al. [26] and Cunningham [62, 63] introduced a special type of spanning trees in G , called *strongly feasible trees*. They showed that the network simplex method for the LSAP can be implemented so as to visit only bases which correspond to strongly feasible trees. Strongly feasible trees are defined similarly as alternating trees in the Hungarian method.

Definition 3.2 A *strongly feasible tree* is a rooted directed tree in G with root in V and arcs directed away from the root, such that for all arcs (i, j) with $i \in V$ and $j \in W$, $x_{ij} = 1$, and for all arcs (i, j) with $i \in W$ and $j \in V$, $x_{ij} = 0$.

Strongly feasible trees have a number of useful properties with respect to the primal network simplex algorithm. Before we describe them, let us say that $(i, j) = f \in T^C$ is a *forward* (*backward*) arc if $i \in V$ ($j \in W$) lies on the directed path joining the root r of T with $j \in W$ ($i \in V$). An arc f which is neither backward nor forward is called a *cross* arc.

Proposition 3.3 (Barr et al. [26], 1977, Cunningham [62, 63], 1976, 1979) Let T be a strongly feasible tree for an LSAP instance, and let T^C be its corresponding co-tree. Then

1. The root has degree 1, every other node from V has degree 2.
2. If the edges $e = (i, j)$ and $f = (i', j')$, $i, i' \in V$, $j, j' \in W$, satisfy the properties $e \in T^C$, $f \in c(T, e)$, and $i \equiv i'$ then $(T \setminus \{f\}) \cup \{e\}$ is again a strongly feasible tree.
3. For an $e \in T^C$ the pivot by the pair (e, f) defined as above is non-degenerate iff e is a forward arc.

Cunningham showed that the primal network simplex algorithm for the LSAP cannot perform more than $O(n^2)$ degenerate pivots at some vertex of the assignment polytope, if strongly feasible trees and a specific pivot rule are employed. By combining these ideas with Dantzig's pivot rule (the new basic variable is that with the most negative reduced cost), Hung [104] obtained an algorithm which performs at most $O(n^3 \log \Delta)$ pivots, where Δ is the difference between the objective function value of the starting solution with the optimal one. Orlin [134] reduced the number of pivots to $O(n^3 \log n)$ for the primal network simplex algorithm with Dantzig's rule. He exploits

the fact that employing strongly feasible trees is equivalent to running the standard network simplex algorithm with some lexicographic pivoting rule for an appropriately perturbed problem. (This fact was also observed by Cunningham [62].) By scaling the coefficients of the perturbed problem, an upper bound of $O(n^2 \log \Delta)$ on the number of pivots is obtained. In the case of LSAPs can be shown that there exists an equivalent problem with cost coefficients bounded by $4^n (n!)^2$, and hence $\Delta = O(n \log n)$.

Another primal simplex algorithm which uses a pivot rule similar to Dantzig's was proposed by Ahuja and Orlin [4]. Assuming that the cost coefficients are integral, the algorithm performs at most $O(n^2 \log R)$ pivots and has a worst case complexity of $O(n^3 \log R)$, where R is defined as $R = \max\{|c_{ij}| : 1 \leq i, j \leq n\}$. The pivot rule involves scaling, and a pivot edge is chosen among those edges (i, j) for which $\bar{c}_{ij} \leq -0.5\epsilon$. If at any stage in the algorithm there does not exist a pair (i, j) which fulfills $\bar{c}_{ij} \leq -0.5\epsilon$, then ϵ is divided by 2. This process is continued until ϵ becomes less than 1. At this point an optimal solution is reached.

Based on strongly feasible trees Roohy-Laleh [157] gave a strongly polynomial primal network-simplex algorithm which performs at most $O(n^3)$ pivots and has a worst-case complexity of $O(n^5)$.

Akgül [7] designed a primal simplex algorithm which performs $O(n^2)$ pivots and has an overall worst case complexity of $O(n^3)$. Also this algorithm uses strongly feasible trees as basic solutions and Dantzig's pivot rule to choose the variable which enters the basis. A special feature of the algorithm is that it solves an increasing sequence of LSAPs which are subproblems of the original one, in the sense that the edge sets of the corresponding graphs are subsets of the edge set E for the original problem. The edge set associated with a certain subproblem is obtained by adding some edges incident to a single node to the edge set associated with the previous subproblem. The last subproblem in the sequence is the original LSAP. The author investigates the structure and properties of so-called *cut sets* which are sets of vertices whose dual variables are changed during the solution of a subproblem. It can be shown that the cut sets are disjoint, and that edges whose heads lie in the cut set are dual feasible for all subgraphs considered afterwards. By exploiting these properties it is shown that the optimal solution of the $(i + 1)$ -st subproblem can be found by performing at most $k + 2$ pivots starting from an optimal solution of the i -th subproblem.

Dual simplex-based algorithms

A spanning tree T of the graph G , besides determining uniquely an integer valued basic solution for the LSAP, determines also a basic solution of the dual (7) of the LSAP. This is done by setting the value of one of the variables arbitrarily, say $u_1 = 0$, and the other variables so as to fulfill $u_i + v_j = c_{ij}$ for all i, j such that $(i, j) \in T$. This dual solution is unique up to the arbitrary setting of one of the variables. Moreover, this basic dual solution is feasible iff $\bar{c}_{ij} = c_{ij} - u_i - v_j \geq 0$. Clearly, if both primal and dual solutions determined by T are feasible, then both of them are optimal.

Dual simplex methods for the LSAP work with a tree T which defines a dual feasible basic solution and keep transforming T iteratively until the corresponding primal solution is feasible, and hence, optimal. The transformation is done by pivoting on an edge $e = (k, l)$ of $E \setminus T$ for which the corresponding primal constraint is violated, i.e., $x_{kl} \leq -1$. This edge is added to the current tree T and one of the edges, say f , contained in the cycle $C(T, e)$ is removed from T to form a new tree $T' = (T \setminus \{f\}) \cup \{e\}$. In order to keep dual feasibility the chosen edge $f = (i, j)$ should fulfill $i \in T^{(k)}$, $j \in T^{(l)}$, where $T^{(k)}$, $T^{(l)}$ are the subtrees of T which arise when the edge $e = (k, l)$ is removed from T , and contain k and l , respectively.

Different rules for the choice of the edges e and f yield different dual simplex algorithms. Many of these algorithms make use of so-called *signatures*, introduced by Balinski [18]. Therefore, such algorithms are also called *signature methods*.

Definition 3.4 Consider an LSAP and a spanning tree T in the related graph $G = (V, W; E)$. The row (column) signature of T is the vector of the degrees in T of the vertices in V (W) for some arbitrary but fixed order of those vertices.

It can be shown that a spanning tree T which defines a dual feasible basis and whose row signature contains only one 1 and the other entries equal to 2 is an optimal basis. Consequently, the primal solution defined as follows is optimal. Start with $x_{kl}=1$ for $k \in V$ being the node of degree 1 and $(k, l) \in T$. Set the other variables x_{ij} such that the assignment constraints are fulfilled and $x_{ij} = 0$ for $(i, j) \notin T$. Thus the LSAP is reduced to finding a dual feasible tree with row (column) signature $(1, 2, 2, \dots, 2)$. Balinski [18, 19] proposes two algorithms to find such a tree T . Both algorithms start with a tree with row signature $(n, 1, \dots, 1)$ and dual variables $u_1 = 0$, $v_j = c_{1j}$ for $1 \leq j \leq n$, and $u_i = \min_j \{c_{ij} - v_j\}$, for $2 \leq i \leq n$. Such a tree is often called

a *Balinski tree*. In the first algorithm [18] the choice of the *pivoting edges* e and f is guided only by the signatures, trying to decrease the number of 1-s in the signature. This makes the algorithm a non-genuine simplex method because it may pivot on some edge $e = (k, l)$ with $x_{kl} = 0$. Moreover, the algorithm does not stop when an optimal solution with signature different from $(1, 2, \dots, 2)$ is found. The algorithm performs at most $O(n^2)$ operations to decrease the number of 1-s in the signature by 1. Hence its worst case complexity is $O(n^3)$.

Goldfarb [97] proposed a similar algorithm based on signatures which solves a sequence of subproblems of the given LSAP. As in the algorithm of Akgül [7] the subproblems are LSAPs with cost matrices being submatrices of $C = (c_{ij})$. The cost matrix of the k -th subproblem is defined by the first k rows and the first k columns of C . Balinski's algorithm is used to solve the k -th subproblem, i.e., to produce a tree corresponding to a dual feasible basis with signature $(2, \dots, 2, 1)$. After the transition to the $(k + 1)$ -st subproblem, i.e., after the introduction of the $(k + 1)$ -st row and column of matrix C , the signature either remains optimal or has the form $(3, 2, \dots, 2, 1, 1)$. In the latter case at most $k - 1$ pivots are needed to produce a tree with signature $(2, \dots, 2, 1)$ for the $(k + 1)$ -st problem. Clearly, this algorithm is not a genuine simplex dual algorithm for the same reasons as Balinski's algorithm.

The second algorithm of Balinski [19] employs another pivot rule which is a combination of the rule based on signatures used in [18] and the classical pivot on an edge $e = (i, j)$ with the most negative x_{ij} . This rule allows the algorithm to visit only so-called *strongly dual feasible trees*. These trees are defined in terms of so-called odd and even arcs. Consider a tree T corresponding to a dual feasible solution and root it at the first vertex in V (according to some fixed arbitrary order) and direct all edges away from the root. An arc $(i, j) \in T$ with $i \in V$, $j \in W$, is called *odd arc* and an arc (j, i) with $j \in W$ and $i \in V$ is called *even arc*. A dual feasible tree is strongly dual feasible if $x_{ij} \geq 1$ for all even arcs $(i, j) \in T$, and $x_{ij} \leq 0$ for all odd arcs $(i, j) \in T$ which do not join $1 = i \in V$ with a node j of degree 1 in W . The algorithm proposed in [19] pivots only on edges (i, j) for which $x_{ij} < 0$. Hence, this algorithm is a genuine dual simplex method, sometimes also called *competitive simplex algorithm*. The number of pivots is at most $(n - 1)(n - 2)/2$, thus yielding a worst case time complexity of $O(n^3)$.

Kleinschmidt, Lee, and Schannath [114] have shown that this algorithm of Balinski [19] is equivalent to an algorithm proposed earlier by Hung and Rom [105].

Akgül [5] proposed an algorithm similar to Goldfarb's. The two main differences are: 1) Akgül's algorithm works with *strongly* dual feasible trees, i.e., it finds a strongly dual feasible tree for the k -th problem and transforms it into a strongly dual feasible tree for the $(k + 1)$ -st problem, and 2) the subproblems considered by Akgül are transportation problems. The algorithm performs at most $O(n^2)$ pivots and has a worst case complexity of $O(n^4)$.

A similar algorithm was developed by Paparrizos [138]. The main difference to Akgül's algorithm is that here the subproblems are determined with the help of a Balinski tree. They are, however, solved in the same way as in [5].

Two other signature-guided algorithms were given by Paparrizos [137] and Akgül and Ekin [8]. Paparrizos' algorithm is similar to Balinski's competitive simplex algorithm [19]. The main difference concerns the dual feasibility which is not an invariant of Paparrizos' algorithm. The algorithm starts with a Balinski tree. Then it performs pivots which may produce infeasible dual solutions. The pivots are grouped in stages and each stage starts (and stops) with a strongly dual feasible tree. The last tree of the last stage is also primal feasible and hence optimal. The algorithm visits only so-called *strong trees* which are obtained from strongly feasible trees by dropping the feasibility requirement. The algorithm performs $O(n^2)$ pivots but spends as much time to determine the pair of edges involved in a pivot. Thus the overall complexity is $O(n^4)$. Akgül and Ekin [8] give an efficient implementation of Paparrizos' algorithm, which maintains and updates a forest rather than a single tree. In this way only dual feasible trees are visited. The number of pivots is again $O(n^2)$ but the overall complexity becomes $O(n^3)$.

Later, Paparrizos [139] designed a *purely infeasible* simplex algorithm which performs $O(n^2)$ pivots and has worst case complexity of $O(n^3)$. This algorithm is similar to the algorithm presented in [137] since it starts again with a Balinski tree and visits only strong trees. Also the edge inserted to a the current strong tree during a pivot operation is determined as in [137]. The edge which is removed from the tree is determined, however, by a different rule. In contrast with the previous algorithm proposed in [137], this algorithm is purely infeasible in the sense that it restores dual feasibility only when it stops with an optimal solution.

3.3 Other Algorithms

A primal algorithm of Balinski and Gomory [20]

The algorithm of Balinski and Gomory is probably the oldest primal algorithm for the LSAP. It works with a perfect matching \mathcal{M} in the graph G (i.e., a primal feasible solution x_{ij} , $1 \leq i, j \leq n$) and dual variables u_i, v_j , $1 \leq i, j \leq n$, which fulfill the complementary slackness conditions. The primal and dual solutions are updated iteratively by maintaining thereby the complementarity slackness conditions. Let $\tilde{E} = \{(i, j) : \bar{c}_{ij} = c_{ij} - u_i - v_j \geq 0\}$. If $\tilde{E} = E$ than the primal solution \mathcal{M} and the dual solution u_i, v_j , $1 \leq i, j \leq n$, are optimal. If there is an edge $e = (i, j) \notin \tilde{E}$, then there exists an edge $(r, j) \in \mathcal{M}$. The algorithm builds an alternating tree T in the subgraph $(V, W; \tilde{E})$ of G and grows it by updating the dual variables similarly as in the primal-dual methods. Then, the matching is changed if there is a cycle in $T \cup \{e\}$. This procedure is repeated until $E = \tilde{E}$. A characteristic feature of the algorithm is that \tilde{E} grows monotonically in the course of the algorithm, i.e., if an edge becomes dual feasible at a certain iteration it remains feasible in the subsequent iterations until the algorithm stops. An efficient version of this algorithm with time complexity $O(n^3)$ has been described by Cunningham and Marsh [64].

Cost Operator Algorithms

Two other primal algorithms for the transportation problem, and hence also for the LSAP, are the cost operator algorithms developed by Srinivasan and Thompson [161]. In contrast to the algorithm of Balinski and Gomory these algorithms work with primal basic solutions. Both algorithms start with a dual feasible solution u_i, v_j , $1 \leq i, j \leq n$, found by the usual row/column reductions: compute the column minima, subtract them from the columns, then compute the row minima and subtract them from the rows. Then both algorithms construct a primal feasible basic solution x_{ij} (x_{ij} are reals between 0 and 1), and introduce new costs: $\tilde{c}_{ij} = u_i + v_j$ for all i, j such that x_{ij} is in the basic solution, and $\tilde{c}_{ij} = c_{ij}$, otherwise. Thus, x_{ij} and u_i, v_j form a pair of feasible primal and dual solutions for the LSAP with cost matrix (\tilde{c}_{ij}) . This pair of solutions fulfills the complementarity slackness conditions. Hence both solutions are optimal. Next both algorithms change step by step the costs \tilde{c}_{ij} back to c_{ij} . In each step, the primal and dual solutions are updated so as to remain optimal for the problem considered at that step. One algorithm employs the *cell cost operator* and changes

c_{ij} for one pair (i, j) at a step, whereas the other algorithm employs the *area cost operator* and changes the costs c_{ij} for a set of pairs (i, j) at a step. Both algorithms maintain strongly feasible trees for all intermediate problems, as pointed out by Akgül [6]. The cell operator algorithm performs one dual pivot (changing the dual variables) and one primal pivot (changing the basic solution) at each step. Nevertheless this is not a purely pivotal algorithm because the primal and the dual solution do not always fulfill the complementary slackness conditions. Each step performs $O(n)$ pivots. The total number of pivots is $O(n^2)$.

The area cost operator uses similar dual and primal pivots and tries to restore as many c_{ij} as possible, at a step. This algorithm performs also $O(n^2)$ pivots. Akgül [6] conjectures that both algorithms can be implemented so as to achieve a time complexity of $O(n^3)$.

A recursive algorithm

Thompson [163] proposed an $O(n^3)$ algorithm which solves a sequence of transportation problems the last of which is the given LSAP. Given an LSAP of size n with cost matrix $C = (c_{ij})$, the algorithm introduces an $(n + 1)$ -st dummy vertex in V (dummy row in C) and an $(n + 1)$ -st dummy vertex in W (dummy columns in C). All edges adjacent to dummy nodes have costs equal to 0, $c_{n+1,j} = c_{i,n+1} = 0$, $1 \leq i, j \leq n + 1$. Obviously, the problem of minimizing $\sum_{i,j=1}^{n+1} c_{ij}x_{ij}$ subject to $\sum_j x_{ij} = 1$, if $i \leq n$, and $\sum_j x_{ij} = 0$, if $i = n + 1$, $\sum_i x_{ij} = 1$, if $j \leq n$, and $\sum_i x_{ij} = 0$, if $j = n + 1$, is equivalent to the original LSAP.

The first subproblem solved is the trivial transportation problem defined by the last row of the extended matrix C with supply equal to n at vertex $i = n + 1$ in V , demand equal to 1 for all vertices $j \leq n$ in W , and demand equal to 0 for $j = n + 1$. The k -th problem is a transportation problem defined by the $k - 1$ first rows and the $(n + 1)$ -st row of the extended matrix C , where the supply of the first $k - 1$ vertices in V is equal to 1 and that of the $(n + 1)$ -st vertex is equal to $n - k$. The demands of vertices in W are as for the first subproblem, and they remain the same for all subproblems. The relaxation algorithm proceeds as follows: it produces a primal optimal solution for the current problem and uses it to find an optimal dual solution for the current problem. This is used to produce a primal optimal solution of the next problem in sequence, and so on. A specific feature of the algorithm is that the dual variables change monotonically: the u variables do not increase, whereas the v variables do not decrease. Another feature is that

every primal or dual feasible solution found for some subproblem is also optimal. The algorithm can be implemented to achieve an $O(n^3)$ worst case complexity. For more information we refer to [163].

Forest algorithms

We saw above that primal (dual) simplex-based algorithms for the LSAP work with primal (dual) feasible bases which correspond to spanning trees in bipartite graphs.

Forest algorithms work with forests in G instead of trees. Feasible (primal or dual) forests are defined analogously as feasible (primal or dual) trees. A forest is primal feasible, if there exists a primal feasible solution with $x_{ij} = 0$ for all i, j such that (i, j) is not an edge of the forest. Analogously, a forest is dual feasible, if there exists a dual feasible solution u_i, v_j , such that $\bar{c}_{ij} = c_{ij} - u_i - v_j = 0$, whenever (i, j) is an edge of the forest. Unlike trees, forests do not determine (primal or dual) bases of the problem.

A primal-dual forest algorithm, or more precisely a forest implementation of the Hungarian method, has been described by Akgül in [6]. The algorithm iteratively solves a shortest path problem in the graph G associated with the assignment problem, and equipped with the reduced costs \bar{c}_{ij} as edge lengths. This is done by growing a forest of alternating trees rooted at a free node in V each and by exploiting the relationship between alternating trees and strongly feasible trees. As soon as the leafs of an alternating tree are joined by edges with free nodes in W , a strongly feasible tree arises. The algorithm maintains a set of alternating trees called *planted forest*, and a set of strongly feasible trees called *matched forest*. Each augmentation transforms an alternating tree in a strongly feasible tree, until the planted forest becomes empty. The augmentations employ labeling and update of dual variables like all primal-dual algorithms in general, and the Hungarian method in particular.

An $O(n^3)$ forest algorithm is proposed by Achatz, Kleinschmidt, and Paparrizos [1] and is a more efficient variant of the algorithm proposed by Paparrizos in [139]. The algorithm starts with some basic dual feasible tree rooted at some arbitrary node in V , and constructs a forest by deleting all edges (i, j) where $i \in V$, $j \in W$, and i is the father of j with respect to the usual orientation of the tree which directs all edges away from the root. This yields a spanning forest which is dual feasible. All nodes in W are considered as roots of the connected components to which they belong. Further, the algorithm works with *superforests* which are forests where every connected

component contains one root, and nodes from W , which are not roots, have degree equal to 2. The starting forest constructed as above is a *superforest*. Each superforest is divided into a *surplus forest* and a *deficit forest*. The surplus forest contains all connected components whose roots have degree larger than 1. The deficit forest is the complement of the surplus forest with respect to the considered superforest. The algorithm constructs a sequence of dual feasible superforests by applying a special pivoting strategy. It stops when a superforest with an empty surplus forest is found. It can be shown that this yields a dual optimal solution of the considered LSAP.

3.4 On Parallel Algorithms

Since the late 1980s a number of parallel algorithms for the LSAP has been proposed. They are parallelized versions of primal-dual algorithms based on shortest path computations, due to Kennington and Wang [113], Zaki [173], Balas, Miller, Pekny, and Toth [14], Bertsekas and Castañon [32, 33], parallelized versions of the auction algorithm, due to Zaki [173], Wein and Zenios [170, 171], Bertsekas and Castañon [31], and parallelizations of primal simplex-based methods, due to Miller, Pekny, and Thompson [128], Peters [141], Barr and Hickman [27]. For a good review on parallel algorithms for the LSAP and network flow problems in general the reader is referred to Bertsekas, Castañon, Eckstein, and Zenios [34].

Parallel implementations of primal-dual shortest path algorithms

The primal-dual parallel algorithms based on shortest path computations can be classified into *single node algorithms* and *multi-node algorithms*. Single node algorithms compute a shortest path tree from a *single* free source node in parallel, i.e., by parallelly scanning the arcs connected to a single node. Most of these algorithms are obtained as implementations of the algorithm of Jonker and Volgenant [107]. As shown in [113, 173], considerable speedups can be obtained in the case of dense assignment problems. Kennington et al. [113] obtain a speedup of 3–4 for a number of processors between 3 and 8. For sparse problems the speedups decrease, since the amount of work which can be performed in parallel decreases. Thus, the effectiveness of these algorithms is limited by the density of the input network. Computational results, e.g. Castañon, Smith, and Wilson [53], show that *single instruction stream multiple data stream* (SIMD) shared memory massively parallel architectures are well suited for implementing single node

parallel algorithms to solve dense problems.

Multi-node approaches compute multiple augmenting paths starting from different free nodes, and perform multiple augmentations and updates of variables in parallel. Balas et al. [14] propose a synchronous algorithm of this type which employs a coordination protocol to combine all augmentations and changes of the dual variables so as to maintain feasibility and keep the complementarity slackness conditions fulfilled. Bertsekas et al. [32] extend this protocol to obtain asynchronous algorithms. The coordination in the asynchronous algorithm is realized by maintaining a master copy of a pair of primal and dual solutions which can be accessed by all processors. The latter compute augmenting paths and variable updates based on the copy of the primal and dual solution they have, and then check whether the update is feasible (the master copy may have been updated in the meanwhile). If the update is feasible, the master copy is updated. The master copy is locked during the updates. Computational experience with these algorithms shows that their principal limitation is the decreasing amount of parallel work with the number of iterations. Initially many augmenting paths can be found so that many processors can be used effectively. Then the load decreases as the number of iterations increases. When comparing synchronous and asynchronous implementations Bertsekas et al. [32] show that for sparse problems asynchronous algorithms may be outperformed by synchronous algorithms. This is basically due to the considerable coordination overhead of the asynchronous algorithms.

Parallel implementations of the auction algorithm

Recall that the auction algorithm computes in each iteration the smallest and the second smallest cost margin with respect to the profit for each free customer. Different implementations may compute these differences (also called *bids* in the maximization version of the LSAP) for all free customers - resulting in the so-called *Jacobi methods*, for a single free customer - resulting in so-called *Gauss-Seidel methods*, or for a subset of free customers - resulting in so-called *block Gauss-Seidel methods*. Here again, we have single node and multi-node algorithms. The multi-node algorithms compute the bids for different customers in parallel, and this is appropriate for Gauss-Seidel and block Gauss-Seidel methods. Single node algorithms compute only one bid at a time but perform this computation in parallel. There are also hybrid approaches where multiple bids are computed in parallel and the computation of each bid is done in parallel by several processors. Hybrid ap-

proaches yield the maximum speedup, if the number of processors assigned to different tasks is chosen properly. Bertsekas and Casta n  [31] have proposed a hybrid implementation of the auction algorithm which converges to an optimal solution even in the case of a fully asynchronous implementation. Computational results of Phillips and Zenios [144] and Wein and Zenios [170] show that massively parallel architectures are appropriate for the implementation of the above mentioned hybrid approach. The obtained speedup is often larger than that achieved by primal-dual shortest path algorithms. Bertsekas and Casta n  [31] show that their hybrid approach can be implemented efficiently on SIMD architectures. These authors also compare synchronous versus asynchronous algorithms and show that in the case of auction algorithms, asynchronous implementations really pay off.

Parallel implementations of primal simplex-based algorithms

Most of the computation load of primal simplex-based algorithms concerns the computation of the new edge to be included in the basic solution. It would be desirable to include an edge which violates dual feasibility most, but this requires the computation of the reduced costs for all edges. Many algorithms compute instead a list of candidate edges to enter the basis and choose among them the edge which violates dual feasibility at most.

Most of parallel primal simplex-based algorithms compute this list of candidates in parallel. One possibility is to assign to each processor a set of rows to investigate. Each processor chooses a pivot row. These rows form a list of candidate edges for pivoting. Then, another processor performs pivot steps sequentially. This yields a synchronous algorithm. For the transportation problem such an algorithm has been proposed by Miller et al. [128]. The computational results of Miller et al. show that such procedures yield a good speedup (3–7.5, if 14 processors are employed), when applied to dense problems and implemented on a small-scale coarse-grain parallel system with distributed memory.

Another approach would be to search for edges to enter the basis in parallel, while pivot operations are taking place, resulting in an asynchronous algorithm. Such an algorithm was proposed by Peters [141]. The convergence of the algorithm is guaranteed by the so-called *pivot processor* which performs a pivot only after having checked whether the pivot is feasible. It is necessary to check feasibility because the processors searching for pivot edges do their job while pivots are taking place, and hence, their computations may be based on out-of-date information. Computational experience on a

shared memory machine shows again a good speedup which increases with the size of the problem. Peter's results also show that it pays off to increase the number of processors when the size of the problems increases. The time needed to perform a pivot operation induces, however, limitations to the number of processors which may be used efficiently.

A refinement of the algorithm of Peters was proposed by Barr and Hickman [27]. These authors give an implementation of the algorithm of Barr, Glover, and Klingman [26]. This implementation does not distinguish between search and pivot processors, all processors may perform both tasks. The algorithm maintains a queue of tasks and each processor refers to the queue to receive its task. Also the pivot operations are divided into so-called *basis update* and *flow update* operations, and different groups of operations are performed by different processors in parallel. Computational results show that this algorithm is faster than that of Peters. This is mainly due to the efficient use of the processors through the task assignment approach, but also to the splitting of a long pivot task among two different processors.

4 Asymptotic Behavior and Probabilistic Analysis

When dealing with the asymptotic behavior of the LSAP, it is always assumed that the cost coefficients c_{ij} are independent random variables with a common prespecified distribution. The main question concerns the behavior of the expected optimal value of the problem as its size tends to infinity.

Let us first consider the question whether a random bipartite graph admits a perfect matching. A class of bipartite random graphs $(V, W; E)$ can be described by specifying the number of vertices and the probabilities for the existence of each edge. The existence of a perfect matching in such a graph is intuitively related to the expected number of edges. Erdős and Rényi [74] have shown that $O(n \log n)$ edges do not suffice for the existence of a perfect matching. More precisely, for each $n \in \mathbb{N}$ the authors introduce a class of random bipartite graphs $\mathcal{G}(n) = (V(n), W(n); E(n))$ with $|V(n)| = |W(n)| = n$ and $|E(n)| = O(n \log n)$, and show that the probability that a graph chosen uniformly at random from $\mathcal{G}(n)$ contains a perfect matching tends to 0 as n tends to infinity. The reason for this behavior is the growing probability of the existence of isolated vertices in such graphs as n tends to infinity.

Another scenario is obtained, if a probabilistic model is chosen which avoids isolated vertices. Consider the class $\mathcal{G}(n, d)$ of *directed* bipartite

graphs with n vertices in each part, where each vertex has out-degree d , and denote by $P(n, d)$ the probability that a graph chosen uniformly at random from $\mathcal{G}(n, d)$ contains a perfect matching. (Matchings and perfect matchings in directed bipartite graphs are defined as matchings and perfect matchings of the graphs obtained from the digraphs by removing the orientation of the edges, respectively.) Walkup [168] shows that $P(n, 1)$ tends to 0, but for all $d \geq 2$, $P(n, d)$ tends to 1 as n approaches infinity. More precisely

$$P(n, 1) \leq 3n^{1/2} \left(\frac{2}{e}\right)^n, \quad P(n, 2) \geq 1 - \frac{1}{5n}, \quad \text{and}$$

$$P(n, d) \geq 1 - \frac{1}{122} \left(\frac{d}{n}\right)^{(d+1)(d-2)} \text{ for all } d \geq 3$$

Notice that due to the marriage theorem the existence of a perfect matching in an undirected bipartite graph regular of degree d is trivial.

The above lower bounds on $P(n, d)$ are used by Walkup [169] to show that 3 is an upper bound on the expected optimal value of the LSAP in the case that the cost coefficients c_{ij} are independent random variables uniformly distributed on $[0, 1]$. The idea is to consider sparse subgraphs of the graph G related to the LSAP as described in Section 1. The considered subgraphs are regular of degree d and have edges with small weights. Choosing d edges with the smallest weights could create problems since the chosen edges would not be independent. To avoid this difficulty Walkup replaces each edge by two directed edges with opposite directions, and chooses their weights from distributions such that the minimum of the two weights still has uniform distribution on $[0, 1]$. More precisely, c_{ij} can be obtained as $\min\{a_{ij}, b_{ij}\}$, where a_{ij}, b_{ij} are independent random variables with a common distribution function F given by $F(\lambda) = 1 - (1 - \lambda)^{0.5}$. Then, with the help of a_{ij} and b_{ij} graphs from the classes $\mathcal{G}(n, d)$, $1 \leq d \leq n$, are associated with the $n \times n$ cost matrix $C = (c_{ij})$, and the existence results described above are applied to evaluate the expected value of $\min_{\phi} \sum_i c_{i\phi(i)}$ by means of conditional probabilities.

Four years later the upper bound of 3 was improved to 2 by Karp [111]. The proofs for both bounds are non-constructive and do not lead to heuristics which produce assignments with expected optimal value within the given bounds.

Lower bounds for the expected optimal value of the LSAP with independent and uniformly distributed costs c_{ij} on $[0, 1]$ are given by Lazarus [120]. The author exploits the weak duality and evaluates the expected value of the

dual objective function $\sum_i u_i + \sum_j v_j$ achieved after the first iteration of the Hungarian method (row and column reductions, see also Section 3.1). By computations involving first order statistics it is shown that the expected value of $\sum_i u_i + \sum_j v_j$ - which is a lower bound for the expected optimal value of the LSAP - is of order $1 + \frac{1}{e} + \frac{\log n}{n}$. This yields a bound of 1.368. Moreover, Lazarus evaluates the maximum number of 0-s lying in different rows and columns of the reduced cost matrix after row and column reductions (or, equivalently, after the first iteration of the Hungarian method). This evaluation implies that the probability of finding an optimal assignment after row/column reductions tends to 0 as n tends to infinity.

The lower bound was improved by Goemans and Kodilian [93] to 1.44, and then to 1.51 by Olin [133]. 1.51 is currently the best known value. Goemans and Kodilian propose a dual heuristic for the LSAP and show that the expected objective function value yielded by this heuristic is larger than 1.44. The heuristic starts with the dual solution obtained by applying row/column reductions as in the first iteration of the Hungarian method. Then, this solution is improved by elementary operations in the fashion of the Hungarian method. The analysis of the heuristic builds upon the analysis of Lazarus and essentially exploits the fact that the solution obtained after the first iteration of the Hungarian method is with high probability highly structured. Also the best known bound given by Olin [133] is obtained from a feasible solution of the dual. Again, Olin starts with the solution of the dual obtained by row and column reductions, after the first iteration of the Hungarian method. Then, this solution is improved by adding the second smallest element of each row to all elements in that row and by subtracting to each column the largest among the terms added to the elements of that column. This transformation leads to a new feasible dual solution which yields an expected value of the objective function equal to 1.47. By applying then an analogous transformation starting from the columns of the reduced cost matrix, the bound of 1.51 is obtained.

Summarizing, the gap between the lower and the upper bound on the expected optimal value of the LSAP is still large. In the case of independent costs c_{ij} uniformly distributed in $[0, 1]$, it is believed that the expected value is close to 1.6 as observed by Donath [72] in his experiments, or more precisely, equal to $\frac{\pi^2}{6} - o(1)(\frac{\pi^2}{6} - 1.645)$ as suggested by Mézard and Parisi [127]. Other experiments concerning the expected value of large random LAPs have been done by Pardalos and Ramakrishnan [140].

Frenk, Houweninge, and Rinnooy Kan [81] as well as Olin [133] have

studied the more general situation of an arbitrary distribution function for the cost elements c_{ij} . Frenk et al. analyze the asymptotic behavior of the first order statistics in the case of distribution functions F defined on $(-\infty, +\infty)$ ($\lim_{n \rightarrow \infty} F^{-1}(1/n) = -\infty$, F^{-1} being the inverse of function F) which fulfill the conditions

$$\int_{-\infty}^{+\infty} |x|F(x)dx < \infty \quad \text{and} \quad \liminf_{x \rightarrow +\infty} \frac{F(-x)}{F(-ax)} > 1 \quad \text{for some } a > 1,$$

as well as for functions F defined on $(0, \infty)$ ($\lim_{n \rightarrow \infty} F^{-1}(1/n) = 0$). The estimates on the first order statistics are then used to evaluate the expected optimal value of the LSAP along the ideas of Walkup [169]. In the case of functions F defined on $(-\infty, +\infty)$ the authors show the following relation for the expectation EZ_n of the optimal value Z_n of the LSAP of size n

$$\left(1 - \frac{3}{2\sqrt{e}}\right)^2 \leq \liminf_{n \rightarrow \infty} \frac{EZ_n}{nF^{-1}(1/n)} \leq \limsup_{n \rightarrow \infty} \frac{EZ_n}{nF^{-1}(1/n)} < \infty. \quad (14)$$

In the case that F is defined on $(0, \infty)$ an analogous result holds with a lower bound equal to 0 on the left hand-side of (14). In this second case, Olin [133] imposes further conditions on F and derives specific bounds which generalize those of Walkup and Lazarus. More precisely, if (i) F admits a continuous density function which is strictly positive in a neighborhood of the origin, (ii) F has finite expected value, and (iii) $F^{-1}(0^+) := \lim_{y \rightarrow 0^+} F^{-1}(y)$ exists, then

$$(1 + e^{-1})F^{-1}(0^+) \leq \liminf_{n \rightarrow \infty} EZ_n \leq \limsup_{n \rightarrow \infty} EZ_n \leq 3F^{-1}(0^+).$$

It seems difficult to derive analogous results which hold almost surely under the above mild conditions on F . However, for special distributions e.g. the uniform distribution, almost sure results can be derived. Olin shows that in this case almost surely the lower bound equals $1 + e^{-1}$ and the upper bound equals 4. This result generalizes to distribution functions F defined on $(0, +\infty)$ and fulfilling the stronger set of conditions posed by Olin. In this case the following inequalities hold with probability 1:

$$\frac{1 + e^{-1}}{f(0^+)} \leq \liminf_{n \rightarrow \infty} Z_n \leq \limsup_{n \rightarrow \infty} Z_n \leq \frac{4}{f(0^+)}.$$

Let us now turn to the probabilistic analysis of heuristics for the LSAP. One of the first results in this area was obtained by Karp [110] and provides

an exact algorithm for the LSAP with expected running time $O(n^2 \log n)$ in the case of independent costs c_{ij} uniformly distributed on $[0, 1]$. The proposed algorithm - which is faster than the best known sequential algorithm - is a special implementation of the standard shortest augmenting path algorithm for the assignment problem (see Section 3). It uses priority queues to compute a shortest augmenting path in $O(n^2 \log n)$ time and has thus a worst case time complexity of $O(n^3 \log n)$. The algorithm tries to reduce the number of insertions in the priority queue at the cost of a slight increase of the number of deletions. This is done by so-called *surrogate items*. Each surrogate item replaces a large number of regular items of the priority queue, and is used instead of them. In the case of the specific distribution of the costs c_{ij} the surrogate items do indeed represent and replace many regular items, reducing the expected overall number of operations associated with the queue to $O(n^2)$. Considering that each of the queue-operations takes $O(\log n)$ times and that the time for all other operations is $O(n^2)$, we obtain the above mentioned time complexity.

Other results in this area concern *heuristics* and provide worst case bounds and/or average case bounds in the case that the costs c_{ij} are independently and uniformly distributed on $[0, 1]$. Avis and Devroye [11] analyze a heuristic for the LSAP proposed by Kurzberg [117]. This heuristic decomposes a large problem of size $n = mk$, into k^2 problems of size $m \times m$ each, solves the smaller problems, and combines their solutions to obtain a solution for the original problem. In its space-optimal version ($k = \sqrt{n}$) the heuristic takes $O(n)$ space and $O(n^{2.5})$ time. In its time-optimal version ($k = n^{3/4}$) it takes $O(n^{2.25})$ time and $O(n^{1.5})$ space. It is not difficult to show that the worst case ratio of this heuristic is ∞ . (By applying this heuristic to maximize the objective function of the LSAP, one can obtain a worst case ratio of $\min\{m, k\}$.) However, in the case of cost coefficients c_{ij} uniformly distributed on $[0, 1]$, the expected value of the solution produced by the heuristic gets smaller than or equal to $k/2$ times the expected optimal value, as n tends to infinity.

The first heuristic which produced a solution with expected value within a constant factor from the optimum value, was proposed by Avis and Lai [12]. This is an $O(n^2)$ algorithm which provides a solution within a factor of 4 from the optimal one (for $n \geq 11$), in the case of the specific distribution of the costs c_{ij} . The heuristic elaborates the idea of Walkup [169] to work with a sparse subgraph of the given graph. The sparse subgraph considered by the authors has $10n$ “cheap” edges and contains a perfect matching with

high probability. If necessary, the largest matching contained in this graph is completed to a perfect matching of the original graph in a greedy way. The good behavior of the heuristic relies on the fact that the sparse subgraph contains with high probability a cheap perfect matching, i.e., a good solution of the LSAP instance.

Karp, Rinnooy Kan, and Vohra [112] derive a better heuristic which runs in $O(n \log n)$ time (expected time $O(n)$) and provides a solution of the LSAP whose value is smaller than $3 + O(n^{-a})$, for some $a > 0$, with probability $1 - O(n^{-a})$. The basic idea is similar to that of Avis and Lai: Construct a “cheap” sparse subgraph of the given graph and show that it contains a perfect matching with high probability. Again, if the subgraph does not contain a perfect matching a solution for the original LSAP instance is determined in a greedy way. The subgraph is introduced in terms of so-called *random 2-out* bipartite graphs. Such a graph contains a perfect matching with probability $1 - O(n^{-a})$ and the matching can be found in $O(n \log n)$ time. Further, it is shown how to construct a random 2-out bipartite subgraph with cheap edges for the given LSAP instance. The expected value of a solution obtained either as a perfect matching in this subgraph, or by applying a greedy approach, if the former does not exist, equals $3 + O(n^{-a})$.

5 Bottleneck Assignment Problems

As mentioned in Section 2 linear bottleneck assignment problems (LBAP)

$$\min_{\phi} \max_{1 \leq i \leq n} c_{i\phi(i)} \quad (15)$$

occur in connection with assigning jobs to parallel machines so as to minimize the latest completion time. The first authors who considered bottleneck assignment problems were Fulkerson, Glicksberg and Gross [84]. Gross [98] proved the following duality result for LBAPs which gave rise to the theory of blocking systems, see Edmonds and Fulkerson [73].

Theorem 5.1 (Gross [98], 1959)

Let $N = \{1, 2, \dots, n\}$ and let \mathcal{S}_n be the set of all permutations ϕ of N . Then, for an arbitrary $n \times n$ matrix $C = (c_{ij})$ of real numbers

$$\min_{\phi \in \mathcal{S}_n} \max_{i \in N} c_{i\phi(i)} = \max_{\substack{A, B \subseteq N \\ |A| + |B| = n+1}} \min_{i \in A, j \in B} c_{ij}. \quad (16)$$

So-called *threshold algorithms* play an important role to solve LBAPs. A threshold algorithm alternates between two phases. In the first phase a cost element c_{ij}^* - the *threshold value* - is chosen and a matrix \bar{C} is defined by

$$\bar{c}_{ij} := \begin{cases} 1 & \text{if } c_{ij} > c_{ij}^* \\ 0 & \text{if } c_{ij} \leq c_{ij}^* \end{cases}$$

In the second phase it is checked, whether the bipartite graph with adjacency matrix \bar{C} contains a perfect matching or not. The smallest value c_{ij}^* for which the corresponding bipartite graph contains a perfect matching, is the optimum value of the LBAP (15).

There are several ways to implement such a threshold algorithm. One possibility is to order the cost elements increasingly and to apply a binary search, in the first phase. This leads to an $O(T(n) \log n)$ algorithm, where $T(n)$ is the time complexity for checking the existence of a perfect matching. Concerning $T(n)$ see the remarks after Theorem 1.3 in Section 1.

Another possibility is to mimic the Hungarian method. We start with

$$c^* := \max_{i,j} (\min_i c_{ij}, \min_j c_{ij}) \quad (17)$$

and grow bottleneck augmenting paths as long as the matrix \bar{C} does not contain an assignment with objective function value equal to 0, see Page [136], Garfinkel [88], Derigs and Zimmermann [70], Carpaneto and Toth [55] and Derigs [67]. FORTRAN codes for this method can be found in the book by Burkard and Derigs [39] and in Carpaneto and Toth [55]. The implementations differ in the determination of a starting solution and in the applied data structures. One of the most efficient implementations is described in Derigs [67]. A similar approach using the framework of strong spanning trees (see also Section 3 and Balinski and Gonzalez [21]) is given by Armstrong and Jin [10]. Their algorithm can be implemented within a running time of $O(mn + n^2 \log n)$, where m is the number of finite entries in matrix C , i.e., the number of edges of the bipartite graph G associated with the assignment problem.

An algorithm with the currently best time complexity is obtained by combining the threshold approach with augmenting paths. This idea goes back to Gabow and Tarjan [87], who gave an algorithm for LBAP with worst case complexity $O(m\sqrt{n \log n})$. For dense graphs this bound has been further improved by Punnen and Nair [153]. According to these authors we

first solve

$$\min_{M \in F^*} \max_{(i,j) \in M} c_{ij}, \quad (18)$$

where F^* is the set of all matchings in G which differ from a maximum matching by at most $n\sqrt{n/m}$ edges. Combining the maximum matching algorithm of Alt et al. [9] (see Section 1) with binary search, this problem can be solved in $O(n^{1.5}\sqrt{m})$ time. Then this solution is extended to a optimal solution of the bottleneck assignment problem by growing at most $n\sqrt{n/m}$ augmenting paths. Every augmenting path can be completed in $O(m)$ time, see e.g. Tarjan [162]. Thus the overall complexity of the algorithm becomes $O(n\sqrt{nm})$.

Pferschy [142] describes an algorithm with expected running time $O(n^2)$, i.e., linear on m in the case that the graph G is dense. In a first step the given graph is thinned out by considering only the $2n \log n$ cheapest edges. This can be done in $O(n^2)$ time by using a linear selection algorithm for finding the $2n \log n$ -smallest edge. In the second step the LBAP on the sparse subgraph is solved by using the method of Gabow and Tarjan. This yields $O((n \log n)^{3/2})$ additional elementary operations. Finally the solution is completed to a perfect matching of the full graph by applying again the algorithm of Gabow and Tarjan. But this completion step is only necessary with a low probability, namely with a probability less than $O(1/\sqrt{n \log n})$. Thus, the expected time needed by the completion is $O(n^2)$, and we get an overall expected running time of $O(n^2)$. This algorithm provides not only a good bound in terms of complexity, but is also very efficient and simple to use.

In a detailed study Pferschy [143] compared different (deterministic) solution procedures for LBAPs up to $n = 2000$. He observed that the fastest algorithm with respect to the overall running time consists of computing a lower bound for the optimal solution value, e.g. by (17), then finding a maximum matching (using the Hopcroft-Karp procedure) in the subgraph consisting of the edges with weight less or equal to this bound, and finally augmenting this matching. Moreover, Pferschy noticed that the selection of a sparse subgraph takes a considerable amount of time, whereas not much computational efforts are needed to solve the LBAP in sparse subgraphs.

There are again some simple cases when an optimal solution of the LBAP can be given before hand. As in Section 2 these cases are related to Monge properties of the cost matrix C .

Matrix $C = (c_{ij})$ is said to be a *bottleneck Monge matrix* (cf. Burkard et al. [42]), if it fulfills the following conditions:

$$\max\{c_{ij}, c_{kl}\} \leq \max\{c_{il}, c_{kj}\}, \quad \text{for } 1 \leq i < k \leq n, \quad 1 \leq j < l \leq n. \quad (19)$$

It is easy to show that the identical permutation is an optimal solution of an LBAP with a bottleneck Monge cost matrix. An analogous result holds for inverse bottleneck Monge matrices defined by

$$\max\{c_{ij}, c_{kl}\} \geq \max\{c_{il}, c_{kj}\}, \quad \text{for } 1 \leq i < k \leq n, \quad 1 \leq j < l \leq n. \quad (20)$$

LBAPs with an inverse bottleneck Monge cost matrix are solved by the permutation $\phi(i) = n - i + 1$, for all i . The recognition of permuted bottleneck Monge matrices, however, is more difficult than solving the bottleneck assignment problem directly, see Klinz, Rudolf, and Woeginger [115], and Burkard et al. [42].

In Pferschy [142] the asymptotic behavior of the LBAP is studied. Pferschy shows that the expected value of the optimal solution of an LBAP with independently and identically distributed costs tends towards the infimum of the cost range as the size of the problem tends to infinity. He derives explicit lower and upper bounds for the optimum value of an LBAP of size n .

Besides the classical application in scheduling theory bottleneck assignment problems have also been used in connection with time slot assignment problems (see the description of the telecommunication problem in the earth-satellite systems, Section 2). One variant of the time slot assignment problem is to decompose the given traffic matrix in a weighted sum of at most n permutation matrices such that $T \leq \sum_{k=1}^n \lambda_k P_k$, $\lambda_k \geq 0$ holds and $\sum_{k=1}^n \lambda_k$ is minimum. Balas and Landweer [13] propose a heuristic for this NP-hard problem: solve a series of bottleneck assignment problems in order to determine the weights λ_k .

A multi-level bottleneck assignment problem has been considered by Carraresi and Gallo [50] in connection with the bus drivers' rostering problem: Daily shifts are to be assigned to bus drivers. To each shift a weight is assigned. This weight represents the cost of the shift for the drivers, e.g., length of the shift, lateness, time to reach the starting point etc. The problem of finding an equal balance of the shifts leads to minimizing the total weight of the shifts assigned to a single driver. For solving this NP-hard optimization problem the authors propose to solve a sequence of bottleneck assignment problems.

Lexicographic bottleneck assignment problems (LexBAP) have been considered by Burkard and Rendl [43]. Let $C = (c_{ij})$ be a real $n \times n$ matrix and let ϕ be a permutation of the set $\{1, 2, \dots, n\}$. Now let c_ϕ be the vector where the n cost coefficients $c_{i,\phi(i)}$, $1 \leq i \leq n$, are ordered decreasingly. The lexicographic bottleneck assignment problem asks for a solution ϕ^* with

$$c_{\phi^*} = \text{lexmin}_{\phi \in S_n} c_\phi.$$

The authors propose two solution approaches. The first one scales the cost coefficients and transforms the LexBAP in a sum assignment problem. (A similar transformation of bottleneck assignment problems in sum assignment problems had already been proposed by Page [136]). In a second approach the LexBAP is iteratively solved by a series of bottleneck and sum assignment problems. If the LexBAP has k distinct cost coefficients and we denote the computational complexity of the sum assignment problem by $T(n)$, then LexBAP can be solved in $O(T(n)n^2 \log k)$ time. Baradadym [25] suggested the following improved algorithm. Order the k distinct cost elements decreasingly and represent any cost coefficient c_{ij} by a vector $d_{ij} = (d_{ij}^1, d_{ij}^2, \dots, d_{ij}^k)$ with k elements, where $d_{ij}^r = 1$, if c_{ij} is the r -largest cost coefficient, and $d_{ij}^r = 0$, otherwise. In this model the lexicographic bottleneck assignment problem becomes a lexicographic vector assignment problem which fits in the framework of algebraic assignment problems (see Section 6). The solution of this problem involves $O(kT(n))$ elementary operations.

Balanced assignment problems have been considered by Martello, Pulleyblank, Toth, and de Werra [125]. Given a real $n \times n$ matrix $C = (c_{ij})$, the balanced assignment problem can be formulated as

$$\min_{\phi} \left[\max_i c_{i\phi i} - \min_i c_{i\phi i} \right].$$

For solving this problem the authors sort the entries c_{ij} non-decreasingly and propose the following $O(n^4)$ procedure:

1. Solve the corresponding bottleneck assignment problem. Let ϕ^* be its optimal solution.
2. Define

$$l := \min_i c_{i\phi^*(i)}, \quad u := \max_i c_{i\phi^*(i)}.$$

If $l = u$, stop. A balanced solution has been found. Otherwise go to Step 3.

3. Delete in C all elements $\leq l$ and $> u$ and grow augmenting paths.
If there exists a perfect matching, set ϕ^* equal to that solution and go to Step 2. If no solution exists, then go to Step 4.
4. If u is already the maximum element of matrix C , stop. The present solution is optimal. Otherwise increase u to the next larger element and return to Step 3.

6 Assignment Problems with Other Objective Functions

As pointed out by Burkard, Hahn, and Zimmermann [41], sum and bottleneck assignment problems are just special cases of a more general problem, the so-called *algebraic assignment problem* which can also be solved efficiently.

Let $(H, *, \prec)$ be a totally ordered commutative semigroup with composition $*$ and order relation \prec . The algebraic assignment problem (AAP) with cost coefficients $c_{ij} \in H$ can be formulated as

$$\min_{\phi \in S_n} c_{1\phi(1)} * c_{2\phi(2)} * \cdots * c_{n\phi(n)}. \quad (21)$$

For deriving efficient solution procedures we require some additional properties for $(H, *, \prec)$. We assume that the order relation is compatible with the composition, i.e.,

$$a \prec b \Rightarrow a * c \prec b * c \quad \text{for } a, b, c \in H,$$

and that H is a so-called *d-monoid*, i.e.,

$$a \prec b \Rightarrow \text{there exists } c \in H \text{ such that } a * c = b.$$

As we will see below, under these conditions the AAP can be solved in $O(n^4)$ time. If we additionally require that the *weak cancellation rule*

$$\text{if } a * c = b * c, \text{ then either } a = b \text{ or } a * c = b$$

applies, the complexity can be improved to $O(n^3)$, see Burkard and Zimmermann [46]. For further results in this direction, consult the survey on algebraic optimization by Burkard and Zimmermann [47].

Special models for d-monoids are:

- $H = \mathbb{R}$ with addition as composition and the usual (or inverse) order relation. This model leads to sum assignment problems (with a minimization or a maximization objective function).
- H is the set of extended real numbers $\bar{\mathbb{R}}$ (including $-\infty$) with the usual order relation. The composition is defined by $a * b := \max(a, b)$. This model leads to bottleneck assignment problems.
- $H = \mathbb{R}^n$, the composition is the vector addition and the order relation is the lexicographical order. This leads to lexicographical sum assignment problems.
- $H = \mathbb{R} \times \bar{\mathbb{R}}$, \prec is the lexicographical order and the composition is defined by

$$(a, b) * (c, d) := \begin{cases} (a, b) & \text{if } a \geq c \\ (a, b + d) & \text{if } a = c \end{cases}$$

This leads to the so-called *time-cost assignment problem*, where a duration measure t_{ij} and a cost c_{ij} are assigned under these conditions to each pair (i, j) . Among all assignments which minimize the overall duration we seek for one which minimizes the overall cost.

- $H = [0, 1]$, the order relation is the usual “greater than or equal to” order \geq , and the composition is the multiplication of real numbers. This model leads to reliability assignment problems: $\max \prod_{i=1}^n a_{i\phi(i)}$ over all permutations ϕ of $1, 2, \dots, n$.

Algebraic assignment problems can be solved by performing *admissible transformations*. Let us denote

$$z[C, \phi] := c_{1,\phi(1)} * c_{2,\phi(2)} * \cdots * c_{n,\phi(n)}.$$

Definition 6.1 (*Admissible transformations*)

A transformation T of the $n \times n$ matrix $C = (c_{ij})$ to the matrix $\bar{C} = (\bar{c}_{ij})$ is called admissible with index $z(T)$, if

$$z[C, \phi] = z(T) * z[\bar{C}, \phi]$$

for all $\phi \in S_n$.

A composition of admissible transformations S and T with indices $z(S)$ and $z(T)$, respectively, is again an admissible transformation with index $z(S) * z(T)$. Now we get the following optimality criterion:

Theorem 6.2 *Let $T : C \mapsto \bar{C}$ be an admissible transformation such that there exists a permutation $\hat{\phi}$ with the following properties:*

1. $z(T) * \bar{c}_{ij} \geq z(T)$
2. $z[\bar{C}, \hat{\phi}] * z(T) = z(T).$

Then $\hat{\phi}$ is an optimal assignment with value $z(T)$.

Proof. Let ϕ be an arbitrary permutation. According to Definition 6.1 and properties (1) and (2) of the proposition above we get:

$$z[C, \phi] = z(T) * z[\bar{C}, \phi] \geq z(T) = z(T) * z[\bar{C}, \hat{\phi}] = z[C, \hat{\phi}]$$

Therefore $\hat{\phi}$ is optimal.

Admissible transformations for assignment problems are described by the following theorem:

Theorem 6.3 (Admissible transformations for assignment problems [41], 1977)

Let $I, J \subseteq \{1, 2, \dots, n\}$, $m := |I| + |J| - n \geq 1$, and $c := \min\{c_{ij} : i \in I, j \in J\}$. Then the transformation $C \mapsto \bar{C}$ defined by

$$\begin{aligned} \bar{c}_{ij} * c &= c_{ij}, && \text{for } i \in I, j \in J \\ \bar{c}_{ij} &= c_{ij} * c, && \text{for } i \notin I, j \notin J \\ \bar{c}_{ij} &= c_{ij}, && \text{otherwise} \end{aligned}$$

*is admissible with $z(T) = c * c * \dots * c$, where the expression on the right hand-side contains m factors.*

For solving an algebraic assignment problem we can use now the following procedure:

1. Perform *row reductions*, i.e., perform admissible transformations with $I = \{i\}, J = \{1, 2, \dots, n\}$. Start with $i = 1$ and let $z := z(T)$ be the corresponding index. Continue with $i = 2, \dots, n$ and update $z := z * z(T)$. Afterwards all elements are “nonnegative”, namely $\bar{c}_{ij} * z \geq z$.

2. Perform *column reductions*, i.e., perform admissible transformations with $I = \{1, 2, \dots, n\}$, $J = \{j\}$, for $j = 1, 2, \dots, n$. Afterwards every row and column contains at least one *generalized 0-element*, i.e., an element which fulfills

$$z * \bar{c}_{ij} = z.$$

3. Determine a maximum matching in subgraph of the bipartite graph associated with the assignment problem, which contains only the edges corresponding to the generalized 0-elements, (cf. Theorem 1.6).
4. If the maximum matching is perfect, then stop: the optimal solution is given by this matching and z is the optimal value of the objective function.
Otherwise, go to Step 5.
5. Determine a minimum cover of the “0-elements”. This cover yields the new index sets I and J . I contains the indices of the uncovered rows, J contains the indices of uncovered columns.
6. Perform an admissible transformation determined by the new index sets I and J as in Theorem 6.3, update $z := z * z(T)$, and go to Step 3.

It is rather straightforward to show that this algorithm yields after at most $n^2 - 2n + 3$ admissible transformations an optimal solution of the algebraic assignment problem. If the composition $*$ is specialized to “+”, it becomes a variant of the Hungarian method. If the composition is specialized to the max operation, then we obtain the bottleneck assignment problem and the above algorithm is a variant of the threshold method.

Analogously to (sum) Monge and bottleneck Monge matrices, we can also define Monge matrices in the case that c_{ij} are taken from the ordered semigroup H , cf. Burkard et al. [42]. The AAP with an algebraic Monge cost matrix is again solved to optimality by the identity permutation. The problem of recognizing permuted algebraic Monge matrices is rather subtle. Woeginger [172] has shown that this problem is NP-hard, if $n \geq 3$ and the ordered semigroup fulfills no additional property. It becomes polynomially solvable, if e.g. a weak cancellation rule of the form

$$a * c = b * c \Rightarrow a = b \quad \text{or} \quad a * c = c, \quad \text{for } a, b, c \in H$$

is fulfilled. For details see Burkard et al. [42].

7 Available computer codes and test instances

FORTRAN listings of a primal-dual algorithm for the LSAP, based on shortest path computations done by a version of Dijkstra's algorithm, can be found in the book by Burkard and Derigs [39]. In that book there is also a code of an algorithm for the linear bottleneck assignment problem (LBAP). This algorithm is obtained as an adaptation of the above mentioned LSAP algorithm to the LBAP, and is described in detail by Derigs and Zimmerman [70]. The differences between the two algorithms are discussed by Burkard and Zimmermann [46].

Source codes of another primal-dual algorithm for the LSAP can be downloaded from <http://207.158.230.188/assignment.html>. One can choose among a C++, a PASCAL, and a FORTRAN implementation of the algorithm of Jonker and Volgenant [108] for dense LSAPs, and among a FORTRAN and a PASCAL implementation for sparse LSAPs. There are also PASCAL and FORTRAN codes of an adaptation of the above mentioned algorithm to the LBAP. All codes are available as zip files.

A (compressed) FORTRAN source file - called 548.Z - of an implementation of the Hungarian algorithm, due to Carpaneto and Toth [54], can be downloaded from <ftp://netlib.att.com> in /netlib/toms. Other listings of FORTRAN codes for the LSAP can be found in Carpaneto, Martello, and Toth [51]. These codes are also available from the floppy disc included in the book [51].

The C code (compressed tar) of an efficient implementation of the scaling push-relabel algorithm of Goldberg and Kennedy [94] for the LSAP can be downloaded from Goldberg's network optimization library at

<http://www.neci.nj.nec.com/homepages/avg/soft.html>.

Finally, listings of 5 FORTRAN codes of auction algorithms for the LSAP can be found in Bertsekas' homepage at

<http://web.mit.edu/dimitrib/www/auction.txt>.

The codes are: AUCTION - a forward auction algorithm for symmetric assignment problems, i.e., problems where the cost matrix is symmetric, AUCTION_FLP - a version of AUCTION which employs floating point arithmetic, AUCTION_AS - an auction algorithm for the asymmetric LSAP, AUCTION_FR - a forward-reverse auction algorithm for symmetric LSAPs, and NAUCTION_SP - a combination of the naive auction algorithm with sequential shortest paths methods.

Test instances of the LSAP can be downloaded as ascii files from the

homepage of the OR-Library maintained by J. Beasley at
 $\text{http://mscmga.ms.ic.ac.uk/pub}.$

Other test instances can be obtained as ascii or uuencoded files from the ELIB library at

$\text{ftp://ftp.zib.de/pub/Packages/mp-testdata/assign/index.html}.$

Finally, let us notice that due to the equivalent formulation of the LSAP as a minimum cost flow problem, algorithms developed for the later can also be applied to the LSAP. However, such algorithms are not supposed to exploit the specific features of the LSAP, and hence may not be competitive with algorithms developed especially for the LSAP. Besides Goldberg's network optimization library and Bertsekas' homepage, other codes for network optimization can be found in Netlib at

$\text{http://www.OpsResearch.com/OR-Links/index.html}.$

Furthermore, C codes (compressed tar files) of implementations of the primal and the dual network simplex algorithm, due to Löbel, can be obtained through $\text{http://www.zib.de/Optimization/index.de.html}.$

8 Multidimensional Assignment Problems

8.1 General Remarks and Applications

Multi-dimensional (sometimes referred as *multi-index*) assignment problems (MAP) are natural extensions of the linear assignment problem. They have been considered for the first time by Pierskalla [146]. The most prominent representatives of this class are axial and planar 3-dimensional assignment problems, which are treated in the subsections below.

The general formulation of the MAP is

$$\begin{aligned} \min \quad & \sum_{i_1=1}^n \cdots \sum_{i_d=1}^n c_{i_1 \dots i_d} x_{i_1 \dots i_d} \\ \text{s.t.} \quad & \sum_{i_2=1}^n \cdots \sum_{i_d=1}^n x_{i_1 \dots i_d} = 1, \quad i_1 = 1, \dots, n, \\ & \sum_{i_1=1}^n \cdots \sum_{i_{k-1}=1}^n \sum_{i_{k+1}=1}^n \cdots \sum_{i_d=1}^n x_{i_1 \dots i_d} = 1, \\ & \text{for } k = 2, \dots, d-1, \text{ and } i_k = 1, 2, \dots, n, \end{aligned} \tag{22}$$

$$\sum_{i_1=1}^n \cdots \sum_{i_{d-1}=1}^n x_{i_1 \dots i_d} = 1, \quad i_d = 1, \dots, n,$$

$$x_{i_1 \dots i_d} \in \{0, 1\} \text{ for } 1 \leq i_1, i_2, \dots, i_d \leq n,$$

with n^d cost coefficients $c_{i_1 \dots i_d}$.

Differently stated the MAP asks for $d-1$ permutations $\phi_1, \phi_2, \dots, \phi_{d-1}$ such that the objective function becomes minimum:

$$\min_{\phi_1, \phi_2, \dots, \phi_{d-1}} \sum_{i=1}^n c_{i\phi_1(i)\phi_2(i)\dots\phi_{d-1}(i)}.$$

The multidimensional assignment problem is NP-hard in general, but in the case that the array of the cost coefficients is a Monge array (see [42]), it is solved by the identical permutations $\phi_i = \text{id}$, for $i = 1, 2, \dots, d-1$. A slightly weaker condition is considered in Fortin and Rudolf [79]. Burkard, Rudolf and Woeginger [45] have shown, however, that the MAP remains NP-hard for $d \geq 3$, if the cost array is inverse Monge. This implies that *maximizing* the objective function with cost elements drawn from a Monge array is NP-hard.

In terms of graphs a multidimensional assignment problem can be described as follows: Let a complete d -partite graph $G = (V_1, V_2, \dots, V_d; E)$ with vertex sets V_i , $|V_i| = n$, $i = 1, 2, \dots, d$, and edge set E be given. A subset X of $V = \bigcup_{i=1}^d V_i$ is a *clique*, if it meets every set V_i in exactly one vertex. A d -dimensional assignment is a partition of V into n pairwise disjoint cliques. If c is a real valued cost function defined on the set of cliques of $G = (V_1, V_2, \dots, V_d; E)$, the d -dimensional assignment problem asks for a d -dimensional assignment of minimum cost.

Bandelt, Crama, and Spieksma [24] consider cases where the costs c of a clique are not arbitrary, but given as a function of elementary costs attached to the edges of the complete d -partite graph. The costs of a clique considered in [24] are sum costs (sum of the lengths of edges induced by the clique), star costs (minimum length of a spanning star), tour costs (minimum cost of a traveling salesman tour in the clique), and tree costs (minimum cost of a spanning tree in the clique). The authors derive worst case bounds on the ratio between the costs of solutions obtained by applying simple heuristics and the costs of optimal solutions for these special MAPs.

Multidimensional assignment problems in their general form have recently found some applications as a means to solve data association problems. More specifically, the central problem in any multi-target tracking

and multi-sensor surveillance is the data association problem of partitioning the observations into tracks and false alarms in real time. General classes of these problems can be formulated as multidimensional assignment problems. In the following we briefly describe such a formulation for a multi-target tracking problem. The reader is referred to Poore, Rijavec, Liggins, and Vannicola [150] for more details and an alternative derivation of the multidimensional assignment formulation. In a target tracking process we collect reports from different measurements performed at discrete moments in time denoted by $1, 2, \dots, n$. Let us denote by $Z(k)$ a data set of M_k reports (dummy or missing reports are included) obtained by measurements at time k , and let Z^n denote the cumulative set of $Z(k)$, $k = 1, 2, \dots, n$,

$$Z(k) = \{z_{i_k}^k\}_{i_k=0}^{M_k}, \quad Z^n = \{Z(1), Z(2), \dots, Z(n)\}.$$

The index $i_k = 0$ is reserved for missing reports. We are looking for a feasible partition γ of the data set Z^n into data tracks corresponding to different objects and false alarms, which leads to a best possible reproduction of the real life situation. A feasible partition $\gamma = \{\gamma_1, \gamma_2, \dots, \gamma_{n(\gamma)}\}$ consists of $n(\gamma)$ tracks γ_j , $1 \leq j \leq n(\gamma)$ of reports. Each track γ_j of a partition γ is given as $\gamma_j = \{z_{i_1}^1, z_{i_2}^2, \dots, z_{i_n}^n\}$ and is defined in terms of the properties it fulfills. First the different tracks may have only missing or dummy reports in common, and no track consists of only dummy reports:

$$\gamma_i \cap \gamma_j \subset \{z_0^1, z_0^2, \dots, z_0^n\}, \quad \text{for } i \neq j, \text{ and } \gamma_j \neq \{z_0^1, z_0^2, \dots, z_0^n\}, \quad \text{for all } j$$

Further, the union of all tracks together with the track consisting of only dummy reports covers the cumulative set of data Z^n , and each track contains at least one report from the set $Z(k)$, $1 \leq k \leq n$:

$$Z^n = \left[\bigcup_{j=1}^{n(\gamma)} \gamma_j \right] \cup \{z_0^1, z_0^2, \dots, z_0^n\}, \quad \gamma_j \cap Z(k) \neq \emptyset, \quad \text{for all } j, k.$$

Now we can use 0-1 variables $x_{i_1 i_2 \dots i_n}$ to characterize a feasible partition γ , where

$$x_{i_1 i_2 \dots i_n} = \begin{cases} 1 & \text{if } \{z_{i_1}^1, z_{i_2}^2, \dots, z_{i_n}^n\} \in \gamma \\ 0 & \text{otherwise} \end{cases}$$

and the following equalities are fulfilled

$$\sum_{i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n}^{M_1, M_2, \dots, M_{k-1}, M_{k+1}, \dots, M_n} x_{i_1 i_2 \dots i_n} = 1, \quad \text{for } k = 1, 2, \dots, n.$$

It is clear that these equalities are similar to those in the formulation (22) of the MAP given earlier in this section. In the special case where all M_i have a common value we obtain the constraints of (22).

Let us turn now to the modeling of the objective function. Let γ_0 be a reference partition, e.g. the partition consisting of all false reports, and let $P(\gamma|Z^n)$ be the probability of a partition γ given the data set Z^n . Without going into details - for which the user is referred to [150] - notice that the maximization of $P(\gamma|Z^n)/P(\gamma_0|Z^n)$ over all feasible partitions γ would be a reasonable objective function from the physical point of view. Under natural probabilistic assumptions on the space of feasible partitions it can be assumed that

$$\frac{P(\gamma|Z^n)}{P(\gamma_0|Z^n)} = L_\gamma = \prod_{\{z_{i_1}^1, z_{i_2}^2, \dots, z_{i_n}^n\} \in \gamma} L_{i_1 i_2 \dots i_n},$$

where each *likelihood ratio* $L_{i_1 i_2 \dots i_n} \equiv L(z_{i_1}^1, z_{i_2}^2, \dots, z_{i_n}^n)$ depends only on the sequence of reports $\{z_{i_1}^1, z_{i_2}^2, \dots, z_{i_n}^n\}$ and not on the partition in which this sequence occurs. (The reader is referred to Poore and Rijavec [149] for a complete list of assumptions under which this decomposition is allowed.) By introducing $c_{i_1 i_2 \dots i_n} := -\ln L_{i_1 i_2 \dots i_n}$ for all i_1, i_2, \dots, i_n we get

$$-\ln \left[\frac{P(\gamma|Z^n)}{P(\gamma_0|Z^n)} \right] = \sum_{\{z_{i_1}^1, z_{i_2}^2, \dots, z_{i_n}^n\} \in \gamma} c_{i_1 i_2 \dots i_n}. \quad (23)$$

Then the maximization of $P(\gamma|Z^n)/P(\gamma_0|Z^n)$ is equivalent to the minimization of the sum in the right hand-side of (23).

For other applications of MAPs in target tracking problems, like track initiation and track maintenance, or multi-sensor tracking, the reader is referred to Poore [147] and the references therein. Some Lagrangian relaxation algorithms for the MAP have been developed by Poore and Rijavec [149] and Poore and Robertson [151]. These algorithms relax one or more constraints and derive a set of good Lagrangean multipliers, e.g. by standard nonsmooth optimization techniques, in a first phase. In a second phase they involve a so-called *recovery* procedure to construct a good feasible solution of the original problem starting from an optimal (or good) solution of the relaxed problem with values of Lagrangean multipliers determined in the first phase. A numerical study of data association problems arising in multi-target and multi-sensor tracking is given in Poore [148]. Greedy randomized adaptive search (GRASP) heuristics for multidimensional assignment problems

arising in multitarget tracking and data association have been proposed by Murphrey, Pardalos, and Pitsoulis [130, 131].

Pusztaszeri, Rensing, and Liebling [152] describe another interesting MAP which arises in the context of tracking elementary particles. By solving a five-dimensional assignment problem, they reconstruct tracks generated by charged elementary particles produced by the Large Electron-Positron Collider (LEP) at CERN institute.

8.2 Axial 3-Dimensional Assignment Problems

Consider n^3 cost coefficients c_{ijk} . The *axial 3-dimensional assignment problem* (3-DAP) can then be stated as

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n c_{ijk} x_{ijk} \\ \text{s.t.} \quad & \sum_{j=1}^n \sum_{k=1}^n x_{ijk} = 1, \quad i = 1, 2, \dots, n, \\ & \sum_{i=1}^n \sum_{k=1}^n x_{ijk} = 1, \quad i = 1, 2, \dots, n, \\ & \sum_{i=1}^n \sum_{j=1}^n x_{ijk} = 1, \quad i = 1, 2, \dots, n, \\ & x_{ijk} \in \{0, 1\} \quad \forall \quad 1 \leq i, j, k \leq n. \end{aligned} \tag{24}$$

The name stems from the model which assigns the 1-s on the right hand-side to distinct positions at the axes of a 3-dimensional array. The sum over the corresponding “flat” in the array must equal the amount assigned to the position on the axis, see Fig. 5. We can think of c_{ijk} as the cost of assigning job j to be performed by worker i in machine k . It follows that $x_{ijk} = 1$, if job j is assigned to worker i in machine k , and $x_{ijk} = 0$, otherwise. A feasible solution of the above problem will be a three dimensional *permutation array*, i.e., a 0-1 array that satisfies the above assignment constraints.

Equivalently, a 3-DAP can be described with the help of two permutations ϕ and ψ

$$\min_{\phi, \psi \in S_n} \sum_{i=1}^n c_{i\phi(i)\psi(i)}. \tag{25}$$

Thus this problem has $(n!)^2$ feasible solutions. Karp [109] showed that the 3-DAP is NP-hard in contrast to the sum linear assignment problem (LSAP).

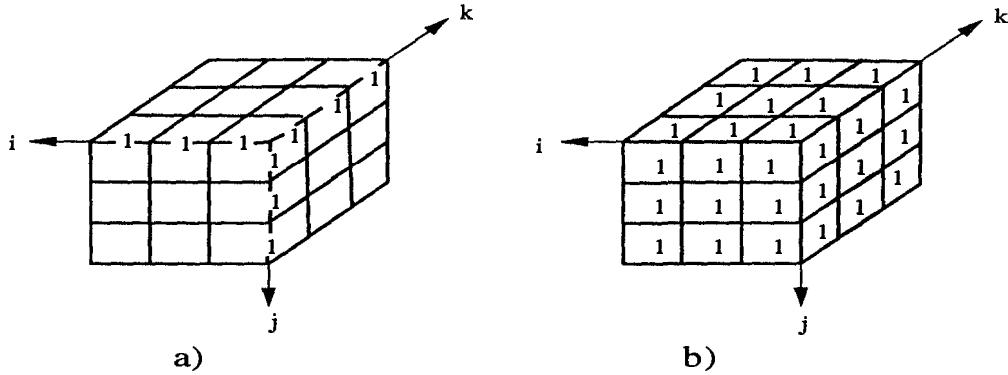


Figure 5: a) A geometric representation of the axial 3-dimensional assignment problem for $n = 3$. b) A geometric representation of the planar 3-dimensional assignment problem for $n = 3$.

In terms of graph theory, the 3-DAP can be stated as follows: Given a complete 3-partite graph $K_{n,n,n}$ with disjoint vertex sets U, V , and W , find a subset A of n triangles, $A \subseteq U \times V \times W$, such that every element of $U \cup V \cup W$ occurs in exactly one triangle of A and the total cost $\sum_{(i,j,k) \in A} c_{ijk}$ is minimized.

Euler [75] started the investigation of the axial 3-index assignment polytope (i.e., the convex hull of feasible solutions to problem (24)) by considering the role of odd cycles for a class of facets of this polytope. Independently Balas and Saltzman [16] investigate in detail the polyhedral structure of the three-index assignment polytope, and show that this polytope has dimension $n^3 - 3n + 2$. They describe an $O(n^4)$ separation algorithm for facets induced by certain cliques. Balas and Qi [15] continue the above work and present $O(n^3)$ separation algorithms for earlier identified classes of facets as well as for a new class of facets. (Note that since there are n^3 variables these are linear-time algorithms). Qi, Balas, and Gwan [154] base their work on linear-time separation algorithms for different facets and construct a polyhedral procedure for solving the 3-DAP.

Pierskalla [145] was the first, who proposed a heuristic for solving 3-DAP. As for algorithms to solve the 3-DAP to optimality, branch and bound is a classical one. Most of the algorithms split the current problem into two subproblems by fixing one variable x_{ijk} to 1 and to 0, respectively. In this way the size of the first subproblem decreases. Balas and Saltzman [17] have

introduced a branching strategy that exploits the structure of the problem and allows to fix several variables at each branching node. Burkard and Rudolf [44] experiment with different branch and bound schemes for the 3-DAP. In particular, they investigate branching rules stemming from the polyhedral description of the underlying polytope, see Balas and Saltzman [16].

Hansen and Kaufman [100] describe a primal-dual method similar to the Hungarian method for linear assignment problems. In the case of 3-DAPs, hypergraphs have to be considered instead of bipartite graphs. First as many 0-elements as possible are generated among the cost coefficients by generalized row- and column reductions. Then a covering problem is solved for these 0-elements. If the covering number is smaller than n , further 0-elements are generated by means of an admissible transformation similar to those in Section 6. If the covering number equals n , the corresponding stability problem is solved. Notice that for hypergraphs the minimum number of (hyper-)edges in a cover is in general strictly larger than the cardinality of a stable set, and this is an inherent difficulty for this problem class. If the stability number equals n , an optimal solution has been found, otherwise a branching is performed. Note the similarity of this method with the approach in Section 6 for solving algebraic assignment problems. Due to this similarity the same algorithm solves the “algebraic” version of the 3-DAP with cost coefficients drawn from a d-monoid. The subproblems of finding a minimum cover and a maximum stable set are in this case, however, NP-hard.

Admissible transformations for 3-DAP are described by the following theorem due to Burkard and Fröhlich [40]. Let us first introduce some notation. Let $N := \{1, 2, \dots, n\}$, $I, J, K \subseteq N$ and denote $\bar{I} := N \setminus I$, $\bar{J} = N \setminus J$, $\bar{K} = N \setminus K$. With these definitions we get

Theorem 8.1 (Admissible transformations for the 3-DAP [40], 1980)
Let a 3-DAP with cost coefficients c_{ijk} , $i, j, k = 1, 2, \dots, n$, be given. For $I, J, K \subseteq N$ with $m := n - (|I| + |J| + |K|) \geq 1$ and

$$c := \min\{c_{ijk} : (i, j, k) \in \bar{I} \times \bar{J} \times \bar{K}\}$$

we define

$$\begin{aligned}\bar{c}_{ijk} &:= c_{ijk} - c, & (i, j, k) \in \bar{I} \times \bar{J} \times \bar{K} \\ \bar{c}_{ijk} &:= c_{ijk} + c, & (i, j, k) \in (\bar{I} \times J \times K) \cup (I \times \bar{J} \times K) \cup (I \times J \times \bar{K}) \\ \bar{c}_{ijk} &:= c_{ijk} + 2c, & (i, j, k) \in I \times J \times K \\ \bar{c}_{ijk} &:= c_{ijk}, & \text{otherwise}\end{aligned}$$

Then, for any feasible solution ϕ, ψ of the 3-DAP we have

$$\sum_{i=1}^n c_{i\phi(i)\psi(i)} = \sum_{i=1}^n \bar{c}_{i\phi(i)\psi(i)} + mc.$$

Clearly, row and column reductions are special cases of the admissible transformations described by the above theorem.

Strong lower bounds are essential for a branch and bound procedure. One of the approaches used to compute lower bounds for the 3-DAP is Lagrangean relaxation. By taking two blocks of the constraints of a 3-DAP into the objective function via Lagrangean multipliers, we get a Lagrangian relaxation of (24) (cf. [40]):

$$L(\pi, \epsilon) := \min \left\{ \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n (c_{ijk} + \pi_j + \epsilon_i) x_{ijk} - \sum_{j=1}^n \pi_j - \sum_{i=1}^n \epsilon_i \right\}$$

such that

$$\begin{aligned}\sum_{i=1}^n \sum_{j=1}^n x_{ijk} &= 1, \quad k = 1, 2, \dots, n \\ x_{ijk} &\in \{0, 1\}, \quad 1 \leq i, j, k \leq n \\ \pi &\in \mathbb{R}^n, \epsilon \in \mathbb{R}^n.\end{aligned}$$

$L(\pi, \epsilon)$ is a concave function, and a subgradient method can be used to find its maximum. Let r be a counter of iterations. Start with $\pi^r := \epsilon^r := 0$ for $r = 0$. In each iteration r use a greedy algorithm to minimize $L(\pi^r, \epsilon^r)$. Let x_{ijk}^r be the corresponding optimal solution. Define $v_{i_0}^r := |\{x_{i_0,j,k}^r : x_{i_0,j,k} = 1\}| - 1$ for $i_0 = 1, 2, \dots, n$, and $w_{j_0}^r := |\{x_{i,j_0,k}^r : x_{i,j_0,k} = 1\}| - 1$ for $j_0 = 1, 2, \dots, n$. If $v^r = w^r = (0, 0, \dots, 0)$, then the maximum is reached. Otherwise π and ϵ are updated by setting

$$\pi^{r+1} := \pi^r + \lambda_r w^r, \quad \epsilon^{r+1} := \epsilon^r + \lambda_r v^r,$$

where λ is a suitable step length λ . This procedure is repeated a prespecified number of iterations.

Another subgradient procedure for solving a Lagrangean relaxation of the 3-DAP together with computational considerations has been described by Frieze and Yadegar [83]. Burkard and Rudolf [44] report on satisfactory computational results obtained by an algorithm which uses the classical branching rule combined with a reduction step in every node of the search tree. The lower bound computation is done by applying the above described subgradient optimization procedure. Another branch and bound algorithm proposed by Balas and Saltzman [17] solves another Lagrangean relaxation to compute the lower bounds. This relaxation incorporates a class of facet inequalities and is solved by a modified subgradient procedure. The upper bounds are computed by a primal heuristic based on the principle of minimizing maximum regret, plus a variable depth interchange phase. A novel branching strategy fixes several variables at each node and reduces the size of the branching tree. The authors report on satisfactory numerical results.

There exists a number of polynomially solvable special cases of the 3-DAP. As mentioned in Section 8.1, the 3-DAP becomes polynomially solvable, if the cost coefficients are taken from a 3-dimensional Monge array (see [42]). Burkard, Rudolf, and Woeginger [45] investigate 3-DAPs with decomposable cost coefficients, where $c_{ijk} = u_i v_j w_k$ and u_i, v_j , and w_k are nonnegative. It is shown that the maximization version of this problem is polynomially solvable, whereas the minimization is in general NP-hard. Moreover, several polynomially solvable special cases of the minimization problem are identified.

Crama and Spieksma [61] consider special cases of the graph theoretic formulation of the 3-DAP. They assume that the lengths of the edges of the underlying 3-partite graph fulfill the triangle inequality, and that the cost of a triangle is defined either as sum of the lengths of its sides (problem $T\Delta$) or as sum of the lengths of the two shorter sides (problem $S\Delta$). The authors prove that the corresponding 3-DAPs are NP-hard. They design, however, heuristics which always deliver feasible solutions whose corresponding value of the objective function is within $3/2$ ($4/3$) from the optimal value, in the case of $T\Delta$ ($S\Delta$). of the optimal cost. Computational experiments show that the performance of these heuristics is very good on randomly generated instances.

8.3 Planar 3-Dimensional Assignment Problems

Let n^3 cost coefficients c_{ijk} be given. *Planar 3-dimensional assignment problems* (3-PAP) have the following form.

$$\begin{aligned}
 \min \quad & \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n c_{ijk} x_{ijk} \\
 \text{s.t.} \quad & \sum_{i=1}^n x_{ijk} = 1, \quad j, k = 1, 2, \dots, n, \\
 & \sum_{j=1}^n x_{ijk} = 1, \quad i, k = 1, 2, \dots, n, \\
 & \sum_{k=1}^n x_{ijk} = 1, \quad i, j = 1, 2, \dots, n, \\
 & x_{ijk} \in \{0, 1\} \quad i, j, k = 1, 2, \dots, n.
 \end{aligned} \tag{26}$$

For a geometric interpretation of the 3-PAP see Fig. 5. Every “flat” in the three-dimensional array x_{ijk} must contain a (2-dimensional) assignment. Thus the feasible solutions of the 3-PAP correspond to *Latin squares*. Fig. 6 shows a feasible solution for a 3-PAP with $n = 3$: number 1 represents the assignment in the lowest horizontal flat, number 2 shows the assignment in the medium flat, and 3 represents the assignment in the upper flat. Due to this interpretation, the number of feasible solutions of a 3-PAP of size n equals the number of Latin squares of order n , and hence increases very fast. The number of different Latin squares of order d equals $d!(d-1)!T(d)$, where $T(d)$ is the number of reduced Latin squares. There is not much known about $T(d)$, the largest value known so far seems to be for $d = 9$. $T(9) = 377,597,570,964,258,816$ due to Bammel and Rothstein [23]. An explicit formula for the number of Latin squares of order d can be found in Shao and Wei [159], but this formula is difficult to evaluate.

Frieze [82] has shown that the 3-PAP is NP-hard. A description of the polyhedral structure of the 3-PAP polytope has been given by Euler, Burkard, and Grommes [76].

3-PAPs have interesting applications in time tabling problems. A recent study on time tables and related polyhedra can be found in Euler and Verge [77].

There are not many algorithms known for the 3-PAP. The first branch and bound algorithm goes back to Vlach [166]. Vlach computes the lower

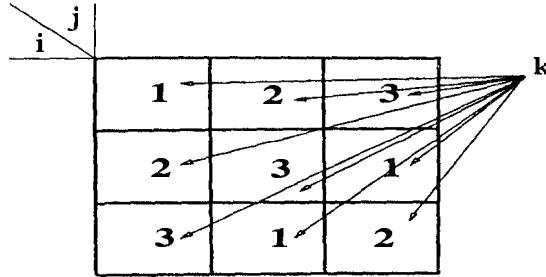


Figure 6: A latin square representing a feasible solution of the planar 3-dimensional assignment problem of size $n = 3$.

bounds by applying (generalized) row and column reductions similar to those used in the case of 3-DAPs. Another branch and bound procedure for the planar 3-index assignment problem has been described by Magos and Miliotis [124]. To our knowledge this is the only paper reporting on computational experience with an algorithm which solves the 3-PAP to optimality. The authors use a so-called relaxation heuristic to compute upper bounds by relaxing the third group of constraints in (26). The remaining problem can be decomposed into n LSAPs, one for each fixed value of the index k . These LSAPs are solved consecutively, and after solving each of them a number of variables are fixed so as to guarantee feasibility for the original problem. The so-found solutions are further improved by a local improvement procedure based on the relationship between 3-PAPs and Latin squares. More precisely, the current solution is presented as a Latin square and a cell of the Latin square is selected. The contents of this cell is changed according to some prespecified rule. Then, the contents of other cells of the Latin square is changed so as to obtain a new Latin square corresponding to a better feasible solution of the 3-PAP.

The lower bounds are computed by a heuristic to solve the dual and - in a small number of branch and bound nodes - also by solving a Lagrangean relaxation of the 3-PAP. The heuristic for the dual is based on row and column reductions in the fashion of the Hungarian method. The Lagrangean relaxation is obtained by relaxing the third group of constraints in (26). Good values for the Lagrangean multipliers are found by standard subgradient optimization techniques (see also Camerini, Fratta, and Maffioli [48]). For each set of fixed multipliers the relaxed problem can again be decomposed in n LSAPs, and is solved similarly as in the case of the relaxation heuristic.

The algorithm uses depth first search and the following branching rule. Fix the indices i and j and let Q be a set of indices k such that $\{(i, j, k) : k \in Q\}$ is the set of free variables. The set Q_1 contains about half as many elements as Q , namely the indices k for those free variables with smaller reduced costs. Then the current node is branched into two new nodes obtained by imposing $\sum_{k \in Q_1} x_{ijk} = 0$ and $\sum_{k \in Q \setminus Q_1} x_{ijk} = 0$.

A nice feature of this branch and bound scheme is that the lower bound is additive, and the lower bounding procedure works with the reduced costs instead of the original costs of the problem. Numerical experiments with instances of size up to 9 show that this really helps. As reported in [124], all instances of size 9 are solved in less than 4 minutes of CPU time.

The neighborhood structure involved in the improvement procedure mentioned above is also used by Magos [123] in a tabu search algorithm. A move in the neighborhood of some Latin square is completely determined by changing the contents of a certain cell (i, j) . This affects at least 4 other cells and at most $2n$ of them (n is the size of the problem). These cells have to be adapted accordingly. There are n^2 cells in total and there are $n - 1$ possibilities to change the contents of each of them. Consequently, the neighborhood size is between $\frac{n(n-1)}{2}$ and $\frac{n^2(n-1)}{4}$. There are two nice properties of this neighborhood structure. First, for each move the change of the objective function can be computed in linear time ($O(n)$). Secondly, not all moves have to be evaluated in each iteration: moves which put a certain element in a certain cell imply the same change in the objective function, independently from the solution to which they are applied. The other tabu search elements involved in this algorithm, e.g. tabu size, tabu and aspiration criteria, stopping criterion etc., are designed and implemented in a standard way. The numerical results show that the algorithm provides a good trade-off between computation time and solution quality for 3-PAP instances of size up to 14.

References

- [1] H. Achatz, P. Kleinschmidt, and K. Paparrizos, A dual forest algorithm for the assignment problem, in *Applied Geometry and Discrete Mathematics*, P. Gritzmann and B. Sturmfels, eds., *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 4, AMS, Providence, RI, 1991, pp. 1–11.

- [2] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, and M. R. Reddy, Applications of network optimization, in *Network Models - Handbooks of Operations Research and Management Science 7*, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., Elsevier, Amsterdam, 1995, pp.1–83.
- [3] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, Faster algorithms for the shortest path problem, *Journal of the ACM* **37**, 1990, 213–223.
- [4] R. K. Ahuja and J. B. Orlin, The scaling network simplex algorithm, *Operations Research* **40**, Suppl. No. 1, 1992, S5–S13.
- [5] M. Akgül, A sequential dual simplex algorithm for the linear assignment problem, *Operations Research Letters* **7**, 1988, 155–158.
- [6] M. Akgül, The linear assignment problem, in *Combinatorial Optimization*, M. Akgül and S. Tufocki, eds., Springer Verlag, Berlin, 1992, pp. 85–122.
- [7] M. Akgül, A genuinely polynomial primal simplex algorithm for the assignment problem, *Discrete Applied Mathematics* **45**, 1993, 93–115.
- [8] M. Akgül and O. Ekin, A dual feasible forest algorithm for the assignment problem, *RAIRO Operations Research* **25**, 1991, 403–411.
- [9] H. Alt, N. Blum, K. Mehlhorn, and M. Paul, Computing maximum cardinality matching in time $O(n^{1.5}\sqrt{m/\log n})$, *Information Process. Letters* **37**, 1991, 237–240.
- [10] R. D. Armstrong and J. Zhiying, Solving linear bottleneck assignment problems via strong spanning trees, *Operations Research Letters* **12**, 1992, 179–180.
- [11] D. Avis and L. Devroye, An analysis of a decomposition heuristic for the assignment problem, *Operations Research Letters* **3**, 1985, 279–283.
- [12] D. Avis and C. W. Lai, The probabilistic analysis of a heuristic for the assignment problem, *SIAM Journal on Computing* **17**, 1988, 732–741.
- [13] E. Balas and P. R. Landwehr, Traffic assignment in communications satellites, *Operations Research Letters* **2**, 1983, 141–147.

- [14] E. Balas, D. Miller, J. Pekny, and P. Toth, A parallel shortest path algorithm for the assignment problem, *Journal of the ACM* **38**, 1991, 985–1004.
- [15] E. Balas and L. Qi, Linear-time separation algorithms for the three-index assignment polytope, *Discrete Applied Mathematics* **43**, 1993, 1–12.
- [16] E. Balas and M. J. Saltzman, Facets of the three-index assignment polytope, *Discrete Applied Mathematics* **23**, 1989, 201–229.
- [17] E. Balas and M. J. Saltzman, An algorithm for the three-index assignment problem, *Operations Research* **39**, 1991, 150–161.
- [18] M. L. Balinski, Signature methods for the assignment problem, *Operations Research* **33**, 1985, 527–537.
- [19] M. L. Balinski, A competitive (dual) simplex method for the assignment problem, *Mathematical Programming* **34**, 1986, 125–141.
- [20] M. L. Balinski and R. E. Gomory, A primal method for the assignment and transportation problems, *Management Science* **10**, 1964, 578–593.
- [21] M. L. Balinski and J. Gonzalez, Maximum matchings in bipartite graphs via strong spanning trees, *Networks* **21**, 1991, 165–179.
- [22] M. L. Balinski and A. Russakoff, On the assignment polytope, *SIAM Review* **16**, 1974, 516–525.
- [23] S. E. Bammel and J. Rothstein, The number of 9×9 Latin squares, *Discrete Mathematics* **11**, 1975, 93–95.
- [24] H.-J. Bandelt, Y. Crama, and F. C. R. Spieksma, Approximation algorithms for multi-dimensional assignment problems with decomposable costs, *Discrete Applied Mathematics* **49**, 1994, 25–50.
- [25] V. A. Bardadym, Modifications of general lexicographic bottleneck optimization problems, *Proceedings of the 5-th Twente Workshop on Graphs and Combinatorial Optimization*, U. Faigle and C. Hoede, eds., 1997, University of Twente, Enschede, The Netherlands, 27–30.
- [26] R. S. Barr, F. Glover, and D. Klingman, The alternating basis algorithm for assignment problems, *Mathematical Programming* **13**, 1977, 1–13.

- [27] R. S. Barr and B. L. Hickman, A new parallel network simplex algorithm and implementation for large time-critical problems, Technical Report, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1990.
- [28] D. P. Bertsekas, A new algorithm for the assignment problem, *Mathematical Programming* **21**, 1981, 152–171.
- [29] D. P. Bertsekas, The auction algorithm: A distributed relaxation method for the assignment problem, *Annals of Operations Research* **14**, 1988, 105–123.
- [30] D. P. Bertsekas, *Linear Network Optimization: Algorithms and codes*, MIT Press, Cambridge, MA, 1991.
- [31] D. P. Bertsekas and D. A. Castaño, Parallel synchronous and asynchronous implementations of the auction algorithm, *Parallel Computing* **17**, 1991, 707–732.
- [32] D. P. Bertsekas and D. A. Castaño, Parallel asynchronous Hungarian methods for the assignment problem, *ORSA Journal on Computing* **5**, 1993, 661–674.
- [33] D. P. Bertsekas and D. A. Castaño, Parallel primal-dual methods to the minimum cost network flow problem, *Computational Optimization and Applications* **2**, 1993, 319–338.
- [34] D. P. Bertsekas, D. A. Castaño, J. Eckstein, and S. Zenios, Parallel computing in network optimization, in *Network Models - Handbooks in Operations Research and Management Science*, Vol. **7**, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., Elsevier, Amsterdam, The Netherlands, 1995, pp. 330–399.
- [35] D. P. Bertsekas and J. Eckstein, Dual coordinate step methods for linear network flow problems, *Mathematical Programming* **42**, 1988, 203–243.
- [36] G. Birkhoff, Tres observaciones sobre el algebra lineal, *Rev. univ. nac. Tucumán (A)* **5**, 1946, 147–151.
- [37] W. L. Brogan, Algorithm for ranked assignments with applications to multiobject tracking, *Journal of Guidance* **12**, 1989, 357–364.

- [38] R. E. Burkard, Time-slot assignment for TDMA-systems, *Computing* **35**, 1985, 99–112.
- [39] R. E. Burkard and U. Derigs, *Assignment and Matching Problems: Solution Methods with FORTRAN Programs*, Springer, Berlin, 1980.
- [40] R. E. Burkard and K. Fröhlich, Some remarks on 3-dimensional assignment problems, *Methods of Operations Research* **36**, 1980, 31–36.
- [41] R. E. Burkard, W. Hahn, and U. Zimmermann, An algebraic approach to assignment problems, *Mathematical Programming* **12**, 1977, 318–327.
- [42] R. E. Burkard, B. Klinz, and R. Rudolf, Perspectives of Monge properties in optimization, *Discrete Applied Mathematics* **70**, 1996, 95–161.
- [43] R. E. Burkard and F. Rendl, Lexicographic bottleneck problems, *Operations Research Letters* **10**, 1991, 303–308.
- [44] R. E. Burkard and R. Rudolf, Computational investigations on 3-dimensional axial assignment problems, *Belgian J. of Operations Research* **32**, 1993, 85–98.
- [45] R. E. Burkard, R. Rudolf, and G. J. Woeginger, Three dimensional axial assignment problems with decomposable cost coefficients, *Discrete Applied Mathematics* **65**, 1996, 123–169.
- [46] R. E. Burkard and U. Zimmermann, Weakly admissible transformations for solving algebraic assignment and transportation problems, *Mathematical Programming Study* **12**, 1980, 1–18.
- [47] R. E. Burkard and U. Zimmermann, Combinatorial optimization in linearly ordered semimodules: a survey, in *Modern Applied Mathematics*, B. Korte ed., North Holland, Amsterdam, 1982, pp. 392–436.
- [48] P. Camerini, L. Fratta, and F. Maffioli, On improving relaxation methods by modified gradient techniques, *Mathematical Programming Study* **3**, 1975, 26–34.
- [49] P. Carraresi and G. Gallo, Network models for vehicle and crew scheduling, *European Journal of Operational Research* **16**, 1984, 139–151.

- [50] P. Carraresi and G. Gallo, A multi-level bottleneck assignment approach to the bus drivers' rostering problem, *European Journal of Operational Research* **16**, 1984, 163–173.
- [51] G. Carpaneto, S. Martello, and P. Toth, Algorithms and codes for the assignment problem, *Annals of Operations Research* **13**, 1988, 193–223.
- [52] P. Carraresi and C. Sodini, An efficient algorithm for the bipartite matching problem, *European Journal of Operational Research* **23**, 1986, 86–93.
- [53] D. A. Castañon, B. Smith, and A. Wilson, Performance of parallel assignment algorithms on different multiprocessor architectures, Technical Report TP-1245, ALPHATECH, Inc., Burlington, Mass.
- [54] G. Carpaneto and P. Toth, Solution of the assignment problem, *ACM Transactions on Mathematical Software* **6**, 1980, 104–11.
- [55] G. Carpaneto and P. Toth, Algorithm for the solution of the bottleneck assignment problem, *Computing* **27**, 1981, 179–187.
- [56] G. Carpaneto and P. Toth, Primal-dual algorithms for the assignment problem, *Discrete Applied Mathematics* **18**, 1987, 137–153.
- [57] K. Cechlárová, The uniquely solvable bipartite matching problem, *Operations Research Letters* **10**, 1991, 221–224.
- [58] B. V. Cherkassky and A. V. Goldberg, On implementing push-relabel methods for the maximum flow problem, *Algorithmica* **19**, 1997, 390–410.
- [59] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, Shortest paths algorithms: theory and experimental evaluation, *Mathematical Programming* **73**, 1996, 129–174.
- [60] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, *Journal of Symbolic Computing* **9**, 1990, 251–280.
- [61] Y. Crama and F. C. R. Spieksma, Approximation algorithms for three-dimensional assignment problems with triangle inequalities, *European Journal of Operational Research* **60**, 1992, 273–279.
- [62] W. H. Cunningham, A network simplex method, *Mathematical Programming* **11**, 1976, 105–116.

- [63] W. H. Cunningham, Theoretical properties of the network simplex method, *Mathematics of Operations Research* **4**, 1979, 196–208.
- [64] W. H. Cunningham and A. B. Marsh, A primal algorithm for optimum matching, *Mathematical Programming Study* **8**, 1978, 50–72.
- [65] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.
- [66] V. G. Deineko and V. L. Filonenko, On the reconstruction of specially structured matrices, *Aktualnyje Problemy EVM i Programmirovaniye*, Dnepropetrovsk, DGU, 1979, (in Russian).
- [67] U. Derigs, Alternate strategies for solving bottleneck assignment problems - analysis and computational results, *Computing* **33**, 1984, 95–106.
- [68] The shortest augmenting path for solving assignment problems - motivation and computational experience, in *Algorithms and Software for Optimization- Part I*, *Annals of Operations Research* **4**, C. L. Monma ed., Baltzer, Basel, 1985, 57–102.
- [69] U. Derigs, O. Goecke, and R. Schrader, Monge sequences and a simple assignment algorithm, *Discrete Applied Mathematics* **15**, 1986, 241–248.
- [70] U. Derigs and U. Zimmermann, An augmenting path method for solving linear bottleneck assignment problems, *Computing* **19**, 1978, 285–295.
- [71] E. W. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik* **1**, 1959, 269–271.
- [72] W. E. Donath, Algorithms and average-value bounds for assignment problems, *IBM Journal on Research Development* **13**, 1969, 380–386.
- [73] J. Edmonds and D. R. Fulkerson, Bottleneck extrema, *Journal of Combinatorial Theory* **8**, 1970, 299–306.
- [74] P. Erdős and A. Rényi, On random matrices, *Pub. Math. Inst. Hung. Acad. of Sciences* **8A**, 1963, 455–461.
- [75] R. Euler, Odd cycles and a class of facets of the axial 3-index assignment polytope, *Applicationes mathematicae (Zastowania Matematyki)* **19**, 1987, 375–386.

- [76] R. Euler, R. E. Burkard, and R. Grommes, On Latin squares and the facial structure of related polytopes, *Discrete Mathematics* **62**, 1986, 155–181.
- [77] R. Euler and H. Le Verge, Time-tables, polyhedra and the greedy algorithm, *Discrete Applied Mathematics* **65**, 1996, 207–221.
- [78] T. A. Ewashko and R. C. Dudding, Application of Kuhn's Hungarian assignment algorithm to posting servicemen, *Operations Research* **19**, 1971, 991.
- [79] D. Fortin and R. Rudolf, Weak algebraic Monge arrays, to appear in *Discrete Mathematics*, 1998.
- [80] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM* **34**, 1987, 596–615.
- [81] J. B. G. Frenk, M. van Houweninge, and A. H. G. Rinnooy Kan, Order statistics and the linear assignment problem, *Computing* **39**, 1987, 165–174.
- [82] A. M. Frieze, Complexity of a 3-dimensional assignment problem, *European Journal of Operational Research* **13**, 1983, 161–164.
- [83] A. M. Frieze and L. Yadegar, An algorithm for solving 3-dimensional assignment problems with application to scheduling in a teaching practice, *Journal of the Operational Research Society* **32**, 1981, 989–995.
- [84] R. Fulkerson, I. Glicksberg, and O. Gross, A production line assignment problem, Technical Report RM-1102, The Rand Corporation, Santa Monica, CA, 1953.
- [85] L. R. Ford and D. R. Fulkerson, Maximal flow through a network, *Canadian Journal of Mathematics* **8**, 1956, 399–404.
- [86] H. N. Gabow and R. E. Tarjan, A linear time algorithm for a special case of set union, *Journal of Computer and System Sciences* **30**, 1985, 209–221.
- [87] H. N. Gabow and R. E. Tarjan, Algorithms for two bottleneck optimization problems, *Journal of Algorithms* **9**, 1988, 411–417.

- [88] R. Garfinkel, An improved algorithm for the bottleneck assignment problem, *Operations Research* **19**, 1971, 1747–1751.
- [89] F. Glover, Maximum matching in a convex bipartite graph, *Naval Research Logistics Quarterly* **14**, 1967, 313–316.
- [90] F. Glover, R. Glover, and D. Klingman, Threshold assignment algorithm, *Mathematical Programming Study* **26**, 1986, 12–37.
- [91] F. Glover, D. Karney, and D. Klingman, Implementation and computational study on start procedures and basis change criteria for a primal network code, *Networks* **4**, 1974, 191–212.
- [92] F. Glover and D. Klingman, Improved labeling of L.P. bases in networks, Research report CS 218, Center for Cybernetic Studies, University of Texas, Austin, TX, 1974.
- [93] M. X. Goemans and M. Kodilian, A lower bound on the expected value of an optimal assignment, *Mathematics of Operations Research* **18**, 1993, 267–274.
- [94] A. V. Goldberg and R. Kennedy, An efficient cost scaling algorithm for the assignment problem, *Mathematical Programming* **75**, 1995, 153–177.
- [95] A. V. Goldberg, S. A. Plotkin, and P. Vaidya, Sublinear-time parallel algorithms for matching and related problems, *Journal of Algorithms* **14**, 1993, 180–213.
- [96] A. V. Goldberg and R. E. Tarjan, Finding minimum-cost circulations by successive approximation, *Mathematics of Operations Research* **15**, 1990, 430–466.
- [97] D. Goldfarb, Efficient dual simplex methods for the assignment problem, *Mathematical Programming* **33**, 1985, 187–203.
- [98] O. Gross, The bottleneck assignment problem, Technical Report P-1630, The Rand Corporation, Sta. Monica, CA, 1959.
- [99] Ph. Hall, On representatives of subsets, *Journal of the London Mathematical Society* **10**, 1935, 26–30.
- [100] P. Hansen and L. Kaufman, A primal-dual algorithm for the three-dimensional assignment problem, *Cahiers du CERO* **15**, 1973, 327–336.

- [101] G. H. Hardy, J. E. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, London and New York, 1952.
- [102] A. J. Hoffman, On simple linear programming problems, in *Convexity, Proceedings of Symposia in Pure Mathematics* **7**, V. Klee ed., AMS, Providence, RI, 1963, 317–327.
- [103] J. E. Hopcroft and R. M. Karp, An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs, *SIAM Journal on Computing* **2**, 1973, 225–231.
- [104] M. S. Hung, A polynomial simplex method for the assignment problem, *Operations Research* **31**, 1983, 595–600.
- [105] M. S. Hung and W. D. Rom, Solving the assignment problem by relaxation, *Operations Research* **28**, 1980, 969–982.
- [106] D. S. Johnson and C. C. McGeoch, eds., *Network Flows and Matching - First DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **12**, AMS, Providence, RI, 1993.
- [107] R. Jonker and A. Volgenant, Improving the Hungarian assignment algorithm, *Operations Research Letters* **5**, 1986, 171–175.
- [108] R. Jonker and A. Volgenant, A shortest augmenting path algorithm for dense and sparse linear assignment problems, *Computing* **38**, 1987, 325–340.
- [109] R. M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, 85–103.
- [110] R. M. Karp, An algorithm to solve the $m \times n$ assignment problem in expected time $O(mn \log n)$, *Networks* **10**, 1980, 143–152.
- [111] R. M. Karp, An upper bound on the expected cost of an optimal assignment, in *Discrete Algorithms and Complexity*, Academic Press, Boston, 1987, 1–4.
- [112] R. M. Karp, A. H. G. Rinnooy Kan, and R. V. Vohra, Average case analysis of a heuristic for the assignment problem, *Mathematics of Operations Research* **19**, 1994, 513–522.

- [113] J. Kennington and Z. Wang, Solving dense assignment problems on a shared memory multiprocessor, Report 88-OR-16, Department of Operations Research and Applied Science, Southern Methodist University, Dallas, TX, 1998.
- [114] P. Kleinschmidt, C. W. Lee, and H. Schannath, Transportation problem which can be solved by the use of Hirsch paths for the dual problems, *Mathematical Programming* **37**, 1987, 153–168.
- [115] B. Klinz, R. Rudolf and G. J. Woeginger, On the recognition of permuted bottleneck Monge matrices, *Discrete Applied Mathematics* **60**, 1995, 223–248.
- [116] D. König, Graphok és matrixok, *Mat. Fiz. Lapok* **38**, 1931, 116–119.
- [117] J. M. Kurzberg, On approximation methods for the assignment problem, *Journal of the ACM* **9**, 1962, 419–439.
- [118] H. W. Kuhn, The Hungarian method for the assignment and transportation problems, *Naval Research Logistics Quarterly* **2**, 1955, 83–97.
- [119] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.
- [120] A. J. Lazarus, Certain expected values in the random assignment problem, *Operations Research Letters* **14**, 1993, 207–214.
- [121] Y. Lee and J. B. Orlin, On very large scale assignment problems, in *Large Scale Optimization: State of the Art*, W. W. Hager, D. W. Hearn, and P. M. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994, pp. 206–244.
- [122] R. E. Machol, An application of the assignment problem, *Operations Research* **18**, 1970, 745–746.
- [123] D. Magos, Tabu search for the planar three-index assignment problem, *Journal of Global Optimization* **8**, 1996, 35–48.
- [124] D. Magos and P. Miliotis, An algorithm for the planar three-index assignment problem, *European Journal of Operational Research* **77**, 1994, 141–153.
- [125] S. Martello, W. R. Pulleyblank, P. Toth, and D. de Werra, Balanced optimization problems, *Operations Research Letters* **3**, 1984, 275–278.

- [126] S. Martello and P. Toth, Linear assignment problems, in *Surveys in Combinatorial Optimization, Annals of Discrete Mathematics* **31**, S. Martello, G. Laporte, M. Minoux, and C. Ribeiro, eds., North-Holland, Amsterdam, 1987, pp. 259–282.
- [127] M. Mézard and G. Parisi, On the solution of the random link matching problems, *Journal de Physique* **48**, 1987, 1451–1459.
- [128] D. Miller, J. Pekny, and G. L. Thompson, Solution of large dense transportation problems using a parallel primal algorithm, *Operations Research Letters* **9**, 1990, 319–324.
- [129] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani, Matching is as easy as matrix inversion, *Combinatorica* **7**, 1987, 105–113.
- [130] R. Murphey, P. M. Pardalos, and L. S. Pitsoulis, A GRASP for the Multitarget Multisensor Tracking Problem, in *Network Design: Connectivity and Facilities Location*, P. M. Pardalos and D.-Z. Du, eds., *DIMACS Series on Discrete Mathematics and Theoretical Computer Science* **40**, AMS, Providence, RI, 1998, pp. 277–302.
- [131] R. Murphey, P. M. Pardalos, and L. S. Pitsoulis, A Parallel GRASP for the Data Association Multidimensional Assignment Problem, in *Parallel Processing of Discrete Problems, The IMA Volumes in Mathematics and its Applications* **106**, Springer Verlag, 1998, pp. 159–180.
- [132] B. Neng, Zur Erstellung von optimalen Triebfahrzeugplänen, *Zeitschrift für Operations Research* **25**, 1981, B159–B185.
- [133] B. Olin, Asymptotic Properties of Random Assignment Problems, Ph.D. Thesis, Division of Optimization and Systems Theory, Department of Mathematics, Royal Institute of Technology, Stockholm, 1992.
- [134] J. B. Orlin, On the simplex algorithm for networks and generalized networks, *Mathematical Programming Studies* **24**, 1985, 166–178.
- [135] J. B. Orlin and R. K. Ahuja, New scaling algorithms for the assignment and minimum cycle mean problems, *Mathematical Programming* **54**, 1992, 41–56.
- [136] E. S. Page, A note on assignment problems, *Computer Journal* **6**, 1963, 241–243.

- [137] K. Paparrizos, A non-dual signature method for the assignment problem and a generalization of the dual simplex method for the transportation problem, *RAIRO Operations Research* **22**, 1988, 269–289.
- [138] K. Paparrizos, A relaxation column signature method for assignment problems, *European Journal of Operational Research* **50**, 1991, 211–219.
- [139] K. Paparrizos, An infeasible (exterior point) simplex algorithm for assignment problems, *Mathematical Programming* **51**, 1991, 45–54.
- [140] P. M. Pardalos and K. G. Ramakrishnan, On the expected value of random assignment problems: Experimental results and open questions, *Computational Optimization and Applications* **2**, 1993, 261–271.
- [141] J. Peters, The network simplex method on a multiprocessor, *Networks* **20**, 1990, 845–859.
- [142] U. Pferschy, The random linear bottleneck assignment problem, *RAIRO Operations Research* **30**, 1996, 127–142.
- [143] U. Pferschy, Solution methods and computational investigations for the linear bottleneck assignment problem, *Computing* **59**, 1997, 237–258.
- [144] C. Phillips and S. Zenios, Experiences with large scale network optimization on the connection machine, in *The Impact of Recent Computing Advances on Operations Research*, *Operations Research Series* **9**, Elsevier, 1989, pp. 169–180.
- [145] W. P. Pierskalla, The tri-substitution method for the three-dimensional assignment problem, *Canadian ORS Journal* **5**, 1967, 71–81.
- [146] W. P. Pierskalla, The multidimensional assignment problem. *Operations Research* **16**, 1968, 422–431.
- [147] A. B. Poore, Multidimensional assignment formulation of data association problems arising from multitarget and multisensor tracking, *Computation Optimization and Application* **3**, 1994, 27–54.
- [148] A. B. Poore, A numerical study of some data association problems arising in multitarget tracking, in *Large Scale Optimization: State of*

- the Art*, W. W. Hager, D. W. Hearn, and P. M. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994, pp. 339–361.
- [149] A. B. Poore and N. Rijavec, Partitioning multiple data sets: multidimensional assignments and Lagrangian relaxation, in *Quadratic assignment and related problems*, P. M. Pardalos and H. Wolkowicz, eds., *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **16**, AMS, Providence, RI, 1994, pp. 317–342.
- [150] A. B. Poore, N. Rijavec, M. Liggins, and V. Vannicola, Data association problems posed as multidimensional assignment problems: problem formulation, in *Signal and Data Processing of Small Targets*, O. E. Drummond ed., SPIE, Bellingham, WA, 1993, pp. 552–561.
- [151] A. B. Poore and A. J. Robertson III, A new Lagrangean relaxation based algorithm for a class of multidimensional assignment problems, *Computational Optimization and Applications* **8**, 1997, 129–150.
- [152] J. Pusztaszeri, P. E. Rensing, and T. M. Liebling, Tracking elementary particles near their primary vertex: a combinatorial approach, *Journal of Global Optimization* **16**, 1995, 422–431.
- [153] A. P. Punnen and K. P. K. Nair, Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem, *Discrete Applied Mathematics* **55**, 1994, 91–93.
- [154] L. Qi, E. Balas, and G. Gwan, A new facet class and a polyhedral method for the three-index assignment problem, in *Advances in Optimization*, D.-Z. Du ed., Kluwer Academic Publishers, 1994, pp. 256–274.
- [155] K. G. Ramakrishnan, N. K. Karmarkar, and A. P. Kamath, An approximate dual projective algorithm for solving assignment problems, in *Network flows and matching—First DIMACS Implementation Challenge*, D. S. Johnson and C. C. McGeoch, eds., *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **12**, AMS, Providence, RI, 1993, pp. 431–449.
- [156] F. Rendl, On the complexity of decomposing matrices arising in satellite communication, *Operations Research Letters* **4**, 1985, 5–8.
- [157] E. Roohy-Laleh, Improvements to the theoretical efficiency of the network simplex method, Ph.D. Thesis, Carleton University, Ottawa, 1980.

- [158] G. Rote and F. Rendl, Minimizing the density in lay-out design, *Operations Research Letters* **5**, 1986, 111–118.
- [159] J. Shao and W. Wei, A formula for the number of Latin squares, *Discrete Mathematics* **110**, 1992, 293–296.
- [160] J. T. Schwarz, Fast probabilistic algorithms for verification of polynomial identities, *Journal of the ACM* **27**, 1980, 701–717.
- [161] V. Srinivasan and G. L. Thompson, Cost operator algorithms for the transportation problem, *Mathematical Programming* **12**, 1977, 372–391.
- [162] R. E. Tarjan, Data Structures and Network Algorithms, SIAM, Philadelphia, PA, 1983.
- [163] G. L. Thompson, A recursive method for solving assignment problems, in *Studies on Graphs and Discrete Programming*, *Annals of Discrete Mathematics* **11**, P. Hansen ed., North Holland, Amsterdam, 1981, 319–343.
- [164] W. T. Tutte, The factorization of linear graphs, *Journal of the London Mathematical Society* **22**, 1947, 107–111.
- [165] L. G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science* **8**, 1979, 189–201.
- [166] M. Vlach, Branch and bound method for the three-index assignment problem, *Ekonomicko-Matematicky Obzor* **12**, 1967, 181–191.
- [167] A. Volgenant, Linear and semi-assignment problems: A core oriented approach, *Computers of Operations Research* **23**, 1996, 917–932.
- [168] D. W. Walkup, On the expected value of a random assignment problem, *SIAM Journal on Computing* **8**, 1979, 440–442.
- [169] D. W. Walkup, Matching in random regular bipartite digraphs, *Discrete Mathematics* **31**, 1980, 59–64.
- [170] J. Wein and S. Zenios, Massively parallel auction algorithms for the assignment problem, *Proceedings of the 3-nd Symposium on the Frontiers of Massively Parallel Computations*, 1990, pp. 90–99.

- [171] J. Wein and S. Zenios, On the massively parallel solution of the assignment problem, *Journal of the Parallel and Distributed Computing* **13**, 1991, 221–236.
- [172] G. J. Woeginger, private communication.
- [173] H. Zaki, A comparison of two algorithms for the assignment problem, Technical Report ORL 90-002, Department of Mechanical and industrial Engineering, University of Illinois, Champaign-Urbana, IL, 1990.

Bin Packing Approximation Algorithms: Combinatorial Analysis

Edward G. Coffman, Jr.
Bell Labs, Lucent Technologies
E-mail: egc@bell-labs.com

Gabor Galambos
Computer Science Department, Teacher's Training College, Szeged
E-mail: galambos@jgytf.u-szeged.hu

Silvano Martello
DEIS, University of Bologna
E-mail: smartello@deis.unibo.it

Daniele Vigo
DEIS, University of Bologna
E-mail: dvigo@deis.unibo.it

Contents

1	Introduction	152
2	Definitions	154
3	On-line Algorithms	157
3.1	Classical algorithms	158
3.2	Weighting functions	159
3.3	Any-Fit and Almost Any-Fit algorithms	161
3.4	Bounded-space algorithms	162
3.5	Variations and the best-in-class	166
3.6	APR lower bounds	169

4 Off-Line Algorithms	171
4.1 Algorithms with presorting	171
4.2 Linear time, randomization, and comparisons	174
4.3 Asymptotic approximation schemes	175
4.4 Anomalies	178
5 Variations and Special Lists	179
5.1 Semi-on-line algorithms	180
5.2 Item-size restrictions	182
5.3 Variable size bins	184
5.4 Dynamic bin packing	187
5.5 Bounds on the number of items per bin	189
5.6 Minimizing the bin capacity	190
5.7 Maximizing the number of items packed	193
5.8 Maximizing the number of full bins	195
5.9 Further directions	196
5.9.1 Partial orders	196
5.9.2 Clustered Items	197
5.9.3 Item types	198
5.9.4 Packing sets and graphs	198

References

1 Introduction

In the classical version of the bin packing problem one is given a list $L = (a_1, \dots, a_n)$ of items (or elements) and an infinite supply of bins with capacity C . A function $s(a_i)$ gives the size of item a_i , and satisfies $0 < s(a_i) \leq C$, $1 \leq i \leq n$. The problem is to pack the items into a minimum number of bins under the constraint that the sum of the sizes of the items in each bin is no greater than C . In simpler terms, a set of numbers is to be partitioned into a minimum number of blocks subject to a sum constraint common to each block. We use the bin packing terminology, as it eases considerably the problem of describing and analyzing algorithms.

The mathematical foundations of bin packing began at Bell Laboratories with the work of M. R. Garey and R. L. Graham in the early 70's. They were soon joined by J. D. Ullman, then at Princeton; these three published the first [49] of many papers to appear in the conferences of the computer science theory community over the next 25 years. The second such paper

appeared within months; in that paper, D. S. Johnson [68] extended the results in [49] and studied general classes of approximation algorithms. In collaboration with A. Demers, researchers Johnson, Garey, Graham, and Ullman published the first definitive analysis of bin packing approximation algorithms [72]. In parallel with the research producing this landmark paper, Johnson completed his 1973 Ph.D. thesis at MIT which gave a more comprehensive treatment and more detailed versions of compact proofs in [72].

The pioneering work in [72] opened an extremely rich research area; it soon turned out that this simple model could be used for a variety of different practical problems, ranging from a large number of cutting stock applications to packing trucks with a given weight limit, assigning commercials to station breaks in television programming, or allocating memory in computers. The problem is well-known to be NP-hard (see Garey and Johnson [50]); hence, it is unlikely that efficient (i.e., polynomial time) optimization algorithms can be found for its solution. Researchers have thus turned to the study of approximation algorithms, which do not guarantee an optimal solution for every instance, but attempt to find a near-optimal solution within polynomial time. Together with closely related partitioning problems, bin packing has played an important role in applications of complexity theory and in the theory of approximation algorithms (see, e.g., the recent book of Hochbaum [62]).

Starting with the seminal papers mentioned above, most of the early research focused on combinatorial analysis of algorithms leading to performance guarantees and bounds on worst-case behavior. In particular, letting $A(L)$ be the number of bins used by an algorithm A and letting $OPT(L)$ be the minimum number of bins needed to pack the elements of L , one tries to find a least upper bound on $A(L)/OPT(L)$ over all lists L (for a more formal definition, see Section 2 below). Much of the research can be divided along the boundary between *on-line* and *off-line* algorithms. In the case of on-line algorithms, items are packed in the order they are encountered in the scan of the given list L ; the bin in which an item is packed is chosen without knowledge of items not yet encountered in L . These algorithms are the only ones that can be used in certain situations, where the items to be packed arrive in a sequence according to some physical process and have to be assigned to a bin as soon as they arrive. Off-line algorithms have complete information about the entire list throughout the packing process.

In the last two decades progressively more attention has been devoted to the probabilistic analysis of packing algorithms; a recent book by Coffman

and Lueker [26] covers the methodology in some detail (see also the book by Hofri [64, Ch. 10]). Nevertheless, combinatorial analysis remains a central research area and, from time to time, numerous new results need to be collected into survey papers. The first comprehensive surveys of bin packing algorithms were by Garey and Johnson [51] in 1981, and by Coffman, Garey, and Johnson [22] in 1984. The next such survey was written some ten years later by Galambos and Woeginger [46] who gave an overview restricted to on-line algorithms. Very recently two new surveys appeared. Csirik and Woeginger [33] concentrate on on-line algorithms, while Coffman, Garey and Johnson [23] extend the coverage to include off-line algorithms as well; both worst-case and average-case analysis are surveyed in [33] and [23].

In this paper, we give a broad summary of results in the one-dimensional bin-packing arena, concentrating on combinatorial analysis; but in contrast to other recent surveys, we devote considerable space to variants of the classical problem. Important new variants continue to arise in many different settings and help account for the thriving interest in bin packing research. We have not attempted to give a self-contained work, e.g., we do not present all algorithms in detail, nor do we give formal proofs; but we refer to certain proof techniques which have been used frequently to establish the most important results. Also, in our coverage of variants, we continue our restriction to partitioning problems in one dimension. Packing in higher dimensions, e.g., strip packing and two-dimensional bin packing, is itself a big subject, one deserving its own survey.

In Section 2, we cover the main definitions and notation. For the classical bin packing problem, we discuss on-line algorithms in Section 3, and off-line algorithms in Section 4. Section 4.4 is a slight abuse of this organization; that section deals with anomalous behavior as it applies to on-line as well as off-line algorithms. Section 5 concentrates on special cases and variants of the classical problem.

2 Definitions

In the classical problem, the bin capacity C is just a scale factor, so we can, without loss of generality, assume the normalization $C = 1$. Unless stated otherwise, this convention is in force throughout. One of the exceptions will be in the section treating a variant of the classical problem in which bins have varying sizes; in that section, bin sizes will be denoted by $s(B)$.

In the algorithmic context, we classify nonempty bins as either *open*

or *closed*. Open bins are available for packing additional items. Closed bins are unavailable, and must remain so, i.e., they are never re-opened. It is convenient to regard a completely full bin as open under any given algorithm until it is specifically closed, even though such bins can receive no further items. We denote the bins by B_1, B_2, \dots . When no confusion arises, we may also use B_j to denote the set of items packed in the j -th bin, and $|B_j|$ to denote the number of such items.

We always pack items organized into a list L (or a list L_j in some indexed set of lists). The notation for list *concatenation* is as usual; $L = L_1 L_2 \dots \dots L_k$ means that the items of list L_i are followed by the items of list L_{i+1} for each $i = 1, 2, \dots, k-1$. When the number of items in a list is needed as part of the notation, we use an index in parentheses; $L_{(n)}$ denotes a list of n items.

If B_j is nonempty, we define its current *content* or *level* as

$$c(B_j) = \sum_{a_i \in B_j} s(a_i)$$

A bin (necessarily empty) is *opened* in order to receive its first item. Of course, it may be closed immediately thereafter, depending on the algorithm and the item size. We assume without loss of generality that all algorithms open bins in order of increasing index. Within any collection of nonempty bins, the *earliest opened*, the *leftmost*, and the *lowest indexed* all refer to the same bin.

In general, we consider lists in which the item sizes are taken from the interval $(0, \alpha]$, with $\alpha \in (0, 1]$. Usually, we assume that $\alpha = \frac{1}{r}$, for some integer $r \geq 1$, and many of our results apply only to $r = \alpha = 1$. We can consider two possible measures of the worst-case behavior of an algorithm. Recall that $A(L)$ denotes the number of bins used by algorithm A to pack the elements of L ; OPT denotes an optimal algorithm, one that uses a minimum number of bins. Define the set V_α of all lists L for which the maximum size of the items is bounded from above by α . For every $k \geq 1$, let

$$R_A(k, \alpha) = \sup_{L \in V_\alpha} \left\{ \frac{A(L)}{k} : OPT(L) = k \right\}$$

Then the *asymptotic worst-case ratio* (or *asymptotic performance ratio*, APR) as a function of α is given by

$$R_A^\infty(\alpha) = \overline{\lim}_{k \rightarrow \infty} R_A(k, \alpha)$$

Clearly $R_A^\infty(\alpha) \geq 1$, and this number measures the quality of the packings produced by algorithm A compared to optimal packings in the worst case. In an equivalent definition, $R_A^\infty(\alpha)$ is a smallest number such that there exists a constant $K \geq 0$ for which

$$A(L) \leq R_A^\infty(\alpha)OPT(L) + K$$

for every list $L \in V_\alpha$. Hereafter, if α is left unspecified, the APR of algorithm A refers to $R_A^\infty \equiv R_A^\infty(1)$.

The second way to measure the worst-case behavior of an algorithm A , is the *absolute worst-case ratio*

$$R_A(\alpha) = \sup_{L \in V_\alpha} \left\{ \frac{A(L)}{OPT(L)} \right\}$$

In modern parlance, this is often called a *competitive ratio*, particularly when A is on-line; generally speaking, competitive analysis is the process of proving bounds on worst-case ratios. For recent work in deriving absolute worst-case ratios for bin packing algorithms, see [15, 95]. Both in theory and in practice the asymptotic worst-case ratio seems to be the more reasonable measure of performance for bin packing algorithms. The absolute worst-case ratio is often achieved for only small numbers of bins and/or items in the optimal packing, and can give an overly pessimistic value for even moderately large instances in practice. Thus, in our treatment of classical bin packing, we will keep with asymptotic worst-case ratios R_A^∞ , often dropping the term “asymptotic”. In our discussion of variants in Section 5, performance ratios other than R_A^∞ will be appropriate. The reader is referred to Section 2.2.10.1 of [23] for further remarks on the comparison between absolute and asymptotic worst-case ratios in the analysis of classical bin-packing algorithms.

We conclude this section by mentioning a special sequence t_i of integers which was investigated by Salzer [94] in connection with a number theoretic problem, and later generalized by Golomb [55]. In describing the performance of various algorithms in later sections, we shall have occasion to refer back to this sequence. For an integer $r \geq 1$, define

$$\begin{aligned} t_1(r) &= r + 1 \\ t_2(r) &= r + 2 \\ t_{i+1}(r) &= t_i(r)(t_i(r) - 1) + 1, \quad \text{for } i \geq 2 \end{aligned}$$

(The Salzer sequence was defined for $r = 1$.) It was conjectured by Golomb that for $r = 1$ this sequence gives the closest approximation to 1 from below among the approximations by sums of reciprocals of k integers, the basis of the conjecture being

$$\sum_{i=1}^k \frac{1}{t_i(1)} + \frac{1}{t_{k+1}(1) - 1} = 1$$

Furthermore, it is easily proved that the following expression is valid for the Golomb sequences:

$$\frac{(r-1)}{t_1(r)} + \sum_{i=2}^k \frac{1}{t_i(r)} + \frac{1}{t_{k+1}(r) - 1} = 1 \quad \text{for any } r \geq 1$$

On the other hand, the following value appears in several results:

$$h_\infty(r) = 1 + \sum_{i=2}^{\infty} \frac{1}{t_i(r) - 1}$$

The first few values of $h_\infty(r)$ are: $h_\infty(1) \approx 1.69103$, $h_\infty(2) \approx 1.42312$, $h_\infty(3) \approx 1.30238$.

3 On-line Algorithms

Recall that a bin-packing algorithm is called *on-line* if it packs each element as soon as it is inspected, without any knowledge of the elements not yet encountered (either the number of them or their sizes). We start with results for some classical algorithms, and then generalize to the *Any-Fit* and the *Almost Any-Fit* classes of on-line algorithms. We will also consider the subclass of *bounded-space* on-line algorithms: An algorithm is bounded-space if the number of open bins at any time in the packing process is bounded by a constant. (The practical significance of this condition is clear; for example, we may have to load trucks at a depot where only a limited number of trucks can be at the loading dock). We will conclude this section with a detailed discussion of lower bounds on the APR's of certain classes of on-line algorithms, but before doing so, we present the most recent variations of old algorithms and the current best-in-class algorithms. We remind the reader that a discussion of the anomalous behavior occurring in on-line algorithms has been deferred to Section 4.4.

3.1 Classical algorithms

We summarize the conventions to be used in this section as follows: In describing an on-line algorithm A for packing list $L = (a_1, \dots, a_n)$, we assume without loss of generality that

1. All bins are initially empty.
2. Bins are opened in the sequence B_1, B_2, \dots
3. A begins by packing a_1 into B_1 and thereafter packs items in the order a_2, a_3, \dots

Occasionally, it will be convenient, just before a decision point, to refer to the *current* item as the next item to be packed; right after A packs a_i , $i < n$, a_{i+1} becomes the new current item.

A simple approach is to pack the bins one at a time according to

Next-Fit (NF): After packing the first item, NF packs each successive item in the bin containing the last item to be packed, if it fits in that bin; if it does not fit, NF closes that bin and packs the current item in an empty bin.

The time complexity of NF is clearly $O(n)$. Note that only one bin is ever open under NF, so it is bounded-space. This advantage is compensated by a relatively poor APR, however.

Theorem 3.1 (Johnson et al. [72]). *We have*

$$R_{\text{NF}}^{\infty}(\alpha) = \begin{cases} 2 & \text{if } \frac{1}{2} < \alpha \leq 1 \\ (1 - \alpha)^{-1} & \text{if } 0 < \alpha \leq \frac{1}{2} \end{cases}$$

Fisher [37] discovered an interesting property that NF does not share with other classical approximation algorithms. He proved that NF packs any list and its reverse into the same number of bins. The conspicuous disadvantage of NF is that it closes bins that could be used for packing later items. An immediate improvement would seem to be never to close bins. But then the next question is: If an item can be put into more than one open bin, which bin should be selected? One possible rule drawn from scheduling theory (where it is known as the greedy or largest-processing-time rule) is the following:

Worst-Fit (WF): If there is no open bin in which the current item fits, then WF packs the item in an empty bin. Otherwise, WF packs the current item into an open bin of smallest content in which it fits; if there is more than one such bin, WF chooses the lowest indexed one.

Although we might expect WF to behave better than NF, it does not.

Theorem 3.2 (Johnson [70]). *We have for all $\alpha \in (0, 1]$*

$$R_{\text{WF}}^{\infty}(\alpha) = R_{\text{NF}}^{\infty}(\alpha)$$

To achieve smaller APR's, there are many better rules for choosing from among the open bins. One that quickly comes to mind is:

First-Fit (FF): FF packs the current item in the lowest indexed nonempty bin in which it fits, assuming there is such a bin. If no such bin exists, FF packs the current item in an empty bin.

A natural complement to WF packs each item into a bin that *minimizes* the space left over.

Best-Fit (BF): If there is no open bin in which the current item fits, then BF packs the item in an empty bin. Otherwise, BF packs the current item into an open bin of largest content in which it fits; if there is more than one such bin BF chooses the lowest indexed one.

By adopting appropriate data structures for representing packings, it is easy to verify that the time complexity of these algorithms is $O(n \log n)$. The analysis of the worst-case behavior of the packings they produce is far more complicated. The basic idea of the upper-bound proofs is the *weighting function* technique, which has played a fundamental role in bin packing theory. So, before proceeding with other algorithms, we describe this technique, which was introduced in [49, 72] and subsequently applied in many other papers (see, e.g, [7, 8, 42, 70, 83]).

3.2 Weighting functions

To bound the asymptotic worst-case behavior of an algorithm A , we can try to find a function $W_A : (0, 1] \rightarrow \mathbb{R}$ with the properties: (i) There exists a

constant $K \geq 0$ such that for any list L

$$\sum_{a \in L} W_A(a) \geq A(L) - K \quad (1)$$

and (ii) there exists a constant K^* such that for any set B of items summing to no more than 1

$$\sum_{a \in B} W_A(a) \leq K^* \quad (2)$$

The value $W_A(a)$ is the *weight* of item a and W_A is a *weighting function* for A . Note that (1) requires that for large packings (large $A(L)$), the average total weight of the items in a bin must satisfy a lower bound close to 1 for all lists L . On the other hand, (2) says that the total weight in any bin of any packing is at most K^* , so for an optimal packing

$$\sum_{a \in L} W_A(a) = \sum_{i=1}^{OPT(L)} \sum_{a \in B_i} W_A(a) \leq K^* OPT(L) \quad (3)$$

Together, (1) and (3) obviously imply the bound $A(L) \leq K^* OPT(L) + K$, and hence $R_A^\infty \leq K^*$. We remark that the technique above is only representative; it is easy to find weaker conditions on the weighting function which will produce the same upper bound for the APR.

The proof of the NF upper bound is not appreciably simplified by a weighting function argument, but it does offer a simple example of such arguments. Consider the case $\alpha = 1$ and define $W_{\text{NF}}(a) = 2s(a)$ for all a . We need but one observation about NF when $NF(L) > 1$: Since the first item of B_{j+1} did not fit in B_j , $1 \leq j < NF(L)$, the sum of the item sizes in $B_j \cup B_{j+1}$ exceeds 1 and hence the sum of the weights of the items in $B_j \cup B_{j+1}$ exceeds 2. Then

$$\begin{aligned} \sum_{i=1}^{NF(L)} \sum_{a \in B_i} W_{\text{NF}}(a) &\geq \sum_{i=1}^{\lfloor NF(L)/2 \rfloor} \sum_{a \in B_{2i-1} \cup B_{2i}} W_{\text{NF}}(a) \\ &\geq 2 \lfloor NF(L)/2 \rfloor \geq NF(L) - 1 \end{aligned}$$

so (1) holds with $K = 1$. The inequality (2) with $K^* = 2$ is immediate from the definition of W_{NF} , so $R_{\text{NF}}^\infty \leq 2$, as desired.

There is no systematic way to find appropriate weighting functions, and the approach can be difficult to work out. For example, consider the proof of the following result.

Theorem 3.3 (Johnson et al. [72]). *We have*

$$R_{\text{FF}}^{\infty}(\alpha) = R_{\text{BF}}^{\infty}(\alpha) = \begin{cases} \frac{17}{10} & \text{if } \frac{1}{2} < \alpha \leq 1 \\ 1 + \lfloor \frac{1}{\alpha} \rfloor^{-1} & \text{if } 0 < \alpha \leq \frac{1}{2} \end{cases}$$

The proof of the $\frac{17}{10}$ upper bound consists of verifying that the following weighting function suffices

$$W(x) = \begin{cases} \frac{6}{5}x & \text{if } 0 \leq x \leq \frac{1}{6} \\ \frac{9}{5}x - \frac{1}{10} & \text{if } \frac{1}{6} < x \leq \frac{1}{3} \\ \frac{6}{5}x + \frac{1}{10} & \text{if } \frac{1}{3} < x \leq \frac{1}{2} \\ \frac{6}{5}x + \frac{2}{5} & \text{if } \frac{1}{2} < x \leq 1 \end{cases}$$

and it requires a substantial effort. (The argument reduces to several pages of case analysis.) Moreover, despite a few hints that emerge in this effort, a clear understanding of how this function was obtained in the first place requires still more effort.

Theorem 3.3 for $\alpha \leq \frac{1}{2}$ can be proved without weighting functions (see [72]), but the reader may find it instructive to prove the $1 + \lfloor \frac{1}{\alpha} \rfloor^{-1}$ upper bound with the weighting function

$$W_{\text{FF}}(a) = \frac{r+1}{r}s(a), \quad \alpha = \frac{1}{r}, \quad r \geq 2$$

Sequences of specific examples establish lower bounds for $R_A^{\infty}(\alpha)$. In particular, one seeks a sequence of lists $L_{(n_1)}, L_{(n_2)}, \dots$ satisfying $L_{(n_k)} \in V_{\alpha}$ ($k = 1, 2, \dots$), $\lim_{k \rightarrow \infty} OPT(L_{(n_k)}) = \infty$, and for some constant K_* ,

$$\lim_{k \rightarrow \infty} \frac{A(L_{(n_k)})}{OPT(L_{(n_k)})} = K_*$$

Then $R_A^{\infty}(\alpha) \geq K_*$, and if K_* is equal to K^* of the upper bound analysis, we have the APR for the algorithm considered. Finding worst-case examples can be anywhere from quite easy to quite hard. For example, the reader would have little trouble with NF, but would probably find FF to be quite challenging.

3.3 Any-Fit and Almost Any-Fit algorithms

The algorithms described so far belong to a much larger class of on-line heuristics satisfying similar worst-case properties. It is clear that FF, WF,

and BF satisfy the following condition:

Any-Fit constraint: *If B_1, \dots, B_j are the current non-empty bins, then the current item will not be packed into B_{j+1} unless it does not fit in any of the bins B_1, \dots, B_j .*

We denote the class of on-line heuristics satisfying the Any-Fit constraint by \mathcal{AF} . The following result shows that FF and WF are best and worst algorithms in \mathcal{AF} , in the APR sense.

Theorem 3.4 (Johnson [70]). *For every algorithm $A \in \mathcal{AF}$ and for every $\alpha \in (0, 1]$*

$$R_{\text{FF}}^{\infty}(\alpha) \leq R_A^{\infty}(\alpha) \leq R_{\text{WF}}^{\infty}(\alpha)$$

By a slight tightening of the Any-Fit constraint, we can eliminate the high-APR algorithms like WF and define a class of heuristics all having the same APR.

Almost Any-Fit constraint: *If B_1, \dots, B_j are the current non-empty bins, and B_k ($k \leq j$) is the unique bin with the smallest content, then the current item will not be packed into B_k unless it does not fit in any of the bins to the left of B_k .*

Clearly, WF does not satisfy this condition, but it is easy to verify that both FF and BF do. We denote the class of on-line algorithms satisfying both constraints above by \mathcal{AAF} .

Theorem 3.5 (Johnson [70]). *$R_A^{\infty}(\alpha) = R_{\text{FF}}^{\infty}(\alpha)$ for every A in \mathcal{AAF} and $\alpha \in (0, 1]$.*

Almost Worst-Fit (AWF) is a modification of WF whereby the current item is always placed in a bin having the second lowest content, if such a bin exists and the current item fits in it; the current item is packed in a bin with smallest content only if it fits nowhere else. Interestingly, though it seems to differ little from WF, AWF has a substantially better APR, since it is in \mathcal{AAF} and hence $R_{\text{AWF}}^{\infty}(\alpha) = R_{\text{FF}}^{\infty}(\alpha)$.

3.4 Bounded-space algorithms

An on-line bin packing algorithm uses *k*-bounded space if, for each item, the choice of where to pack it is restricted to a set of at most *k* open bins. We

obtain bounded-space counterparts of the algorithms of the previous section by specifying a suitable policy for closing bins.

As previously observed, NF uses only 1-bounded space; the only algorithm to use less is the trivial algorithm that puts each item in a separate bin. To improve on the APR of NF, yet stay with bounded-space algorithms, Johnson [69] proposed an algorithm that packs items according to the First-Fit rule, but considers as candidates only the k most recently opened bins; when a new bin has to be opened and there are already k open bins, then the lowest-indexed open bin is closed. We expect that the APR of the resulting algorithm, which is known as *Next- k -Fit* (NF_k), tends to $\frac{17}{10}$ as k increases. Finding the exact bound was not an easy task, although Johnson did give a narrow range for the APR. Later, Csirik and Imreh [30] constructed the worst-case sequences, and later still, Mao was able to prove the exact bound.

Theorem 3.6 (Mao [87]). *For any $k \geq 2$, $R_{\text{NF}_k}^{\infty} = \frac{17}{10} + \frac{3}{10(k-1)}$.*

In general, we define a bounded-space algorithm by specifying the packing and closing rules. An interesting class of such rules is based on FF and BF as follows.

- **Packing rules:** the elements are packed following either the First-Fit rule or the Best-Fit rule.
- **Closing rules:** the next bin to close is either the the lowest indexed one, or the one of largest content.

The algorithm that uses packing rule X, closing rule Y, and k -bounded space is denoted by AXY_k , where $X = F$ or B for FF or BF and $Y = F$ or B for the lowest-indexed (First) open bin or the largest-content (Best) open bin. With this terminology NF_k can also be classified as AFF_k . Note that, independently of the chosen rules, if $k = 1$ then we always get NF.

Algorithm ABF_k was first analyzed by Mao, who called it *Best- k -Fit*. He proved that, for any fixed $k \geq 2$, this algorithm is slightly better than NF_k .

Theorem 3.7 (Mao [86]). *For any $k \geq 2$, $R_{\text{ABF}_k}^{\infty} = \frac{17}{10} + \frac{3}{10k}$.*

The tight asymptotic bound for AFB_k was found by Zhang.

Theorem 3.8 (Zhang [105]). *For any $k \geq 2$, $R_{\text{AFB}_k}^{\infty} = R_{\text{NF}_k}^{\infty}$.*

Finally, consider the algorithm ABB_k whose asymptotic behavior is, rather surprisingly, independent of $k \geq 2$, and equal to that of FF and BF.

Theorem 3.9 (Csirik and Johnson [31]). *If $k \geq 2$ then $R_{\text{ABB}_k}^\infty = \frac{17}{10}$.*

Since all of the above algorithms fulfill the Any-Fit constraint with respect to the open bins, the overall bound $R_A^\infty \geq \frac{17}{10}$ is to be expected from Theorem 3.4. A better on-line algorithm can only be obtained without the Any-Fit constraint. In the remaining part of this section, we show how the fruitful idea of *reservation techniques* (introduced by Yao [102] for unbounded-space algorithms and discussed in the next section) led to on-line algorithms which are neither in class \mathcal{AF} nor in \mathcal{AAF} .

Yao's idea appeared in the work of Lee and Lee [83] who developed the *Harmonic-Fit* algorithm, which will be denoted by HF_k since, for the case $\alpha = 1$, it uses at most k open bins. The algorithm divides the interval $(0, 1]$ into subintervals $I_j = (\frac{1}{j+1}, \frac{1}{j}]$ ($1 \leq j \leq k - 1$) and $I_k = (0, \frac{1}{k}]$. An element is called an I_j -*element* if its size belongs to interval I_j . Similarly, there are k different types of bin: an I_j -*bin* is reserved for I_j -elements only. An I_j -element is always packed into an I_j -bin following the Next-Fit rule, and so at most k bins are open at the same time. Galambos [42] extended the idea to general α . Observe that, if we choose $\alpha \in (\frac{1}{r+1}, \frac{1}{r}]$, then the number, say M , of bin types exceeds the space bound k by $r - 1$; this is because I_j -bins for $j < r$ are never opened. Instead of our notation HF_k with k the space bound, the literature often uses HF_M with $M = k + r - 1$ being the number of bin types.

The general APR can be formulated as follows. (See the end of Section 2 for the definitions of the quantities $t_s(r)$ and $h_\infty(r)$.)

Theorem 3.10 (Lee and Lee [83], Galambos [42]). *Suppose that $L \in V_\alpha$ with $\alpha \in (\frac{1}{r+1}, \frac{1}{r}]$ for some positive integer r , and choose any sequence k_s , $s \geq 1$, such that $t_s(r) < k_s + 1 \leq t_{s+1}(r)$. Then*

$$\lim_{s \rightarrow \infty} R_{\text{HF}_{k_s}}^\infty(\alpha) = \lim_{s \rightarrow \infty} \left(1 + \sum_{j=2}^s \frac{1}{t_j(r) - 1} + \frac{k_s + r - 1}{(t_{s+1}(r) - 1)(M - 1)} \right) = h_\infty(r)$$

The results in [83, 42] gave tight bounds only for the cases $k = t_j(r) - r + 1$ and $k = t_{j+1}(r) - r$ for integers $j \geq 1$. Also, considering only the $\alpha = 1$ case, one can see that, to obtain an APR better than $\frac{17}{10}$, at least 7 open bins are needed. These observations raised two further questions:

- For the case $\alpha = 1$, is there an on-line, bounded-space algorithm that uses fewer than 7 bins and has an APR better than $\frac{17}{10}$?
- What are tight bounds on HF_k for specific k ?

An affirmative answer to the first question was given by Woeginger [100]. Using a more sophisticated interval structure, one based on the Golomb sequences, the performance of his *Simplified Harmonic* (SH_k) algorithm undershot the $\frac{17}{10}$ bound with 6 open bins; precisely, $R_{SH_6}^\infty \approx 1.69444$. Moreover, Woeginger proved the following deeper, more general result.

Theorem 3.11 (Woeginger [100]). *To achieve the worst-case performance ratio of heuristic HF_k with k open bins and $\alpha = 1$, heuristic SH_k only needs $O(\log \log k)$ open bins.*

The second question was investigated by Csirik and Johnson [31] and van Vliet [97, 98]. They gave tight bounds for the case $\alpha = 1$ with $k = 4$ and 5. Tight bounds for further k remain open problems.

Similarly, the general case has not been discussed exhaustively, and some questions raised by Woeginger [100] are still open:

- What is the smallest k such that there exists an on-line heuristic using k -bounded space and having an APR strictly less than $\frac{17}{10}$?
- What is the best possible APR for any on-line heuristic using 2-bounded space? (ABB_2 achieves a worst-case ratio of $\frac{17}{10}$.)
- If we consider only algorithms that pack the items by the Next-Fit rule according to some fixed partition of $(0, 1]$ into k subintervals, which partition gives the best APR ? (It is known that for $k \leq 2$ the best possible APR is 2 (see Csirik and Imreh [30]), but for $k \geq 3$ no tight bound is known.)

Tables 1 and 2 show the best results known for bounded-space algorithms. Note that the worst-case ratios of all algorithms in Table 1 are never smaller than $h_\infty(1) \approx 1.69103$. As pointed out by Lee and Lee [83] for the $\alpha = 1$ case, bounded-space algorithms cannot do better. The result holds for general α too, as shown by Galambos, i.e.,

Theorem 3.12 (Lee and Lee [83], Galambos [42]). *Every bounded-space on-line bin packing algorithm A satisfies $R_A^\infty(\alpha) \geq h_\infty(r)$ for all α , $\frac{1}{r+1} < \alpha \leq \frac{1}{r}$.*

k	NF_k	ABF_k	ABB_k	$\text{HF}_k \leq$	SH_k	best
2	2.00000	1.85000	1.70000	2.00000*	2.00000	1.70000
3	1.85000	1.80000	1.70000	1.75000*	1.75000	1.75000
4	1.80000	1.77500	1.70000	1.71429*	1.72222	1.70000
5	1.77500	1.76000	1.70000	1.70000*	1.70000	1.70000
6	1.76000	1.75000	1.70000	1.70000*	1.69444	1.69444
7	1.75000	1.74286	1.70000	1.69444*	1.69388	1.69388
8	1.74286	1.73750	1.70000	1.69388	1.69106	1.69106
9	1.73750	1.73333	1.70000	1.69345	1.69104	1.69104
10	1.73333	1.73000	1.70000	1.69312	1.69104	1.69104
42	1.70732	1.70714	1.70000	1.69106*	1.69103	1.69103
$+\infty$	1.70000	1.70000	1.70000	1.69103	1.69103	1.69103

Table 1: APR values for bounded-space algorithms, rounded to five decimals. The values in column $\text{HF}_k \leq$ are upper bounds, and are tight if starred. Note that $42 = t_4(1) - 1$.

We note finally that none of the known bounded-space algorithms achieves the lower bound using a finite number of open bins. We will later show (see Section 5.1) that the bound can be achieved with three open bins if repacking among the open bins is allowed, but without such a relaxation the question remains open.

3.5 Variations and the best-in-class

Chronologically, Yao [102] was the first to break through the $h_\infty(1)$ barrier with his *Refined First-Fit* (RFF) algorithm. RFF classifies items into types 1, 2, 3, or 4 according as their sizes are in the respective intervals $(0, \frac{1}{3}]$, $(\frac{1}{3}, \frac{2}{5}]$, $(\frac{2}{5}, \frac{1}{2}]$ and $(\frac{1}{2}, 1]$. RFF packs 4 sequences of bins, one for each type. With one exception, RFF packs type- i items into the sequence of type- i bins using First Fit. The exception is that every sixth type-2 item (with a size in $(\frac{1}{3}, \frac{2}{5}]$) is thrown in with the type-4 items, i.e., packed by First Fit into the sequence of type-4 bins. It is easily verified that RFF uses unbounded space and has a time complexity $O(n \log n)$. Yao proved that $R_{\text{RFF}}^\infty = \frac{5}{3} = 1.666\dots$. We remark that Yao did not use the usual weighting function technique, but based his proof on enumeration of the elements in each class. Also, the APR remains unchanged if the special treatment given every sixth type-2 item is instead given every m th type-2 item, where m is taken to be one of 7, 8, or 9.

k	$r = 1$	$r = 2$	$r = 3$	$r = 4$	$r = 5$	$r = 6$
2	2.00000*	1.50000*	1.33333*	1.25000*	1.20000*	1.18888*
3	1.75000*	1.44444*	1.31250*	1.24000*	1.19444*	1.16326*
4	1.71429*	1.43750	1.31000	1.23888	1.19387	1.16294
5	1.70000*	1.43333	1.30833	1.23809	1.19345	1.16269
6	1.70000*	1.43055	1.30714	1.23750	1.19312	1.16250
7	1.69444*	1.42857	1.30625	1.23703	1.19285	1.16233
8	1.69388	1.42708	1.30555	1.23666	1.19264	1.16220
9	1.69345	1.42592	1.30500	1.23636	1.19246	1.16208
10	1.69312	1.42499	1.30454	1.23611	1.19230	1.16198
$+\infty$	1.69103	1.42307	1.30238	1.23441	1.19102	1.16102

Table 2: APR values for $\text{HF}_k(\frac{1}{r})$. Starred values are tight.

It was immediately clear that this reservation technique was a promising approach, a fact supported by the Harmonic-Fit algorithm discussed in the previous section. The main disadvantage of the latter algorithm is that each I_1 -element, even with a size slightly over $\frac{1}{2}$, is packed alone into a bin. An immediate improvement is to try to add other items to these bins. In their algorithm *Refined Harmonic-Fit* (RHF), Lee and Lee [82, 83] modified HF_{20} by subdividing intervals I_1 and I_2 into two subintervals:

$$\begin{aligned} I_1 &= I_{1,s} \cup I_{1,b}, \\ I_2 &= I_{2,s} \cup I_{2,b} \end{aligned}$$

with $I_{2,s} = (\frac{1}{3}, \frac{37}{96}]$, $I_{2,b} = (\frac{37}{96}, \frac{1}{2}]$, $I_{1,s} = (\frac{1}{2}, \frac{59}{96}]$ and $I_{1,b} = (\frac{59}{96}, 1]$. This brought the number of bin types to $M = 22$. The packing strategy for I_j -elements ($3 \leq j \leq M - 2 = 20$), $I_{1,b}$ -elements and $I_{2,b}$ -elements is the same as in Harmonic Fit, but the $I_{1,s}$ -elements and the $I_{2,s}$ -elements are allowed to share the same bins in certain situations. We omit the details and simply point out that the time complexity of RHF is $O(n)$, but that the algorithm is no longer bounded-space. Its APR is given by

Theorem 3.13 (Lee and Lee [83]). $R_{\text{RHF}}^{\infty} \leq \frac{373}{228} \approx 1.63596$.

It is not known whether this bound is tight.

In 1989 several improved algorithms were presented by Ramanan et al. [91]. The first one, called *Modified Harmonic-Fit* (MHF), applies Yao's idea in a more sophisticated way. Instead of choosing $\frac{59}{96}$ as the point dividing

$(\frac{1}{2}, 1]$, the problem is handled in a more general fashion. Let the number of bin types satisfy $M \geq 5$, and consider the subdivision of $(0, 1]$:

$$(0, 1] = \bigcup_{j=1}^{M-2} I_j$$

with $I_1 = I_{1,s} \cup I_{1,b}$ and $I_2 = I_{2,s} \cup I_{2,b}$ as earlier, but now

$$\begin{aligned} I_{1,s} &= (\frac{1}{2}, 1 - y] & I_{2,s} &= (\frac{1}{3}, y] \\ I_{1,b} &= (1 - y, 1] & I_{2,b} &= (y, \frac{1}{2}] \end{aligned}$$

for some y satisfying $\frac{1}{3} < y < \frac{1}{2}$. Initially, the set of empty bins is divided into M infinite classes, each associated with a subinterval. All $I_{1,b}$ -bins, $I_{2,s}$ -bins, $I_{2,b}$ -bins and I_j -bins ($3 \leq j \leq M - 2$) are only used to pack elements from the associated interval (as in Harmonic Fit). $I_{1,s}$ -bins on the other hand can contain $I_{1,s}$ -elements and some of the $I_{2,s}$ -elements and I_j -elements ($j \in \{3, 6, 7, \dots, M - 2\}$). The algorithm also includes more complicated rules for packing and for deciding when items with sizes in different intervals can share the same bin. For this algorithm with $M = 40$, we have the following bounds.

Theorem 3.14 (Ramanan et al. [91]).

$$1.6156146 < \frac{3}{2} + \frac{1}{9} + \frac{1}{222} - \frac{1}{987012} \leq R_{\text{MHF}}^{\infty} < \frac{3}{2} + \frac{1}{9} + \frac{1}{222} = 1.615615\dots$$

The above algorithm was further generalized by Ramanan et al. [91] who introduced a sequence of classes of on-line linear-time algorithms, called $C^{(h)}$; an algorithm A belongs to $C^{(h)}$, for a given $h \geq 1$, if it divides $(0, 1]$ into disjoint subintervals including

$$\begin{aligned} I_{1,b} &= (1 - y_1, 1], & I_{2,b} &= (y_h, \frac{1}{2}], \\ I_{1,j} &= (1 - y_{j+1}, 1 - y_j], & I_{2,j} &= (y_{h-j}, y_{h-j+1}], & 1 \leq j \leq h. \end{aligned}$$

and

$$I_{\lambda,1} = (0, \lambda], \quad I_{\lambda,2} = (\lambda, \frac{1}{3}], \quad 0 < \lambda \leq \frac{1}{3}$$

where $\frac{1}{3} = y_0 < y_1 < \dots < y_h < y_{h+1} = \frac{1}{2}$, and $I_{\lambda,2} = \emptyset$ if $\lambda = \frac{1}{3}$. Elements are classified, as usual, according to the intervals in which their sizes fall.

Note that algorithm MHF belongs to $C^{(1)}$. Ramanan et al. [91] developed an algorithm (*Modified Harmonic-2*, MH2), which is in $C^{(2)}$, and proved that $R_{\text{MH2}}^{\infty} \leq 1.612\dots$. The algorithm is quite elaborate and beyond the scope of this survey. The authors discussed further improvements aimed at reducing the APR to 1.59. They also proved the lower bound result

Theorem 3.15 (Ramanan et al. [91]). *There is no on-line algorithm A in $C^{(h)}$ such that $R_A^\infty < \frac{3}{2} + \frac{1}{12} = 1.58333\dots$.*

The HARMONIC+1 algorithm of Rickey [92] subdivides intervals $(\frac{1}{2}, 1]$ and $(\frac{1}{3}, \frac{1}{2}]$ very finely (using 76 classes of subintervals!) in order to allow a precise item pairing in these two intervals. It also allows items of size $(\frac{1}{4}, \frac{1}{3}]$ to be mixed with larger items of various sizes and not just with those of size at least $\frac{1}{2}$. This algorithm has the current best APR.

Theorem 3.16 (Rickey [92]). $1.5874 \leq R_{\text{HARMONIC+1}}^\infty \leq 1.587936$.

Rickey also introduced a class of algorithms which differs from $C^{(h)}$ in the way items with sizes at most $\frac{1}{3}$ are packed. Rickey's main result is that the above lower bound remains valid for any algorithm A in this new class.

3.6 APR lower bounds

In previous sections we mentioned some results concerning lower bounds on the APR, but we have yet to consider the fundamental problem: What is the best an on-line algorithm can do in the asymptotic worst case? In this section, we discuss lower bound results in chronological order.

Most of the existing lower bounds come from the same idea. To obtain a good packing, it seems advisable to first pack the large elements so that either a bin is ‘full enough’ or its empty space can be reduced by subsequent small elements. Therefore, if we want to force bad behavior on a heuristic algorithm A , we should challenge it with a list in which small items come first. If A adopts a policy that packs these small items tightly, then it will not be able to find a good packing for the large items which may come later. If instead, A leaves space for large items while packing the small ones, then the expected large items might not appear in the list. In both cases the resulting packing will be poor.

To give a more precise description, let us consider a simple example involving two lists L_1 and L_2 , each containing n identical items. The size of each element in L_1 is $\frac{1}{2} - \varepsilon$, and the size of each element in L_2 is $\frac{1}{2} + \varepsilon$. We investigate the asymptotic behavior of an arbitrary approximation algorithm A on two lists: L_1 alone and the concatenated list L_1L_2 . It is easy to see that $OPT(L_1) = \frac{n}{2}$, and $OPT(L_1L_2) = n$. Let us examine the behavior of our algorithm on L_1 : it will pack some elements alone into bins (say x of them) and it will match up the remaining $n - x$. Hence $A(L_1) = \frac{n+x}{2}$. For the concatenated list L_1L_2 , when processing the elements of L_2 , the

r	Lower bound	R_{HF}^{∞}	R_{FF}^{∞}	R_{NF}^{∞}	Best known
1	1.54015	1.69103	1.70000	2.00000	1.58872
2	1.38965	1.42307	1.50000	2.00000	1.42307
3	1.29144	1.30238	1.33333	1.50000	1.30238
4	1.22986	1.23441	1.25000	1.33333	1.23441
5	1.18888	1.19102	1.20000	1.25000	1.19102
6	1.15891	1.16102	1.16667	1.20000	1.16102

Table 3: Lower bounds and R_A^{∞} for various algorithms.

best that A can do is to add one element of L_2 to each of x bins containing a single element of L_1 , and to pack alone the remaining $n - x$ elements. Therefore, $A(L_1 L_2) = \frac{3n-x}{2}$. Since n may be arbitrarily large,

$$R_A^{\infty} \geq \max \left\{ \frac{A(L_1)}{OPT(L_1)}, \frac{A(L_1 L_2)}{OPT(L_1 L_2)} \right\} = \max \left\{ \frac{n+x}{n}, \frac{3n-x}{2n} \right\}$$

for which the minimum is attained when $x = \frac{n}{3}$, implying a lower bound of $\frac{4}{3}$ for the APR of *any* on-line algorithm A .

The above idea can be easily generalized by taking a carefully chosen series of lists L_1, \dots, L_k , and evaluating the performance of a heuristic on the concatenated lists $L_1 \dots L_j$, ($1 \leq j \leq k$). The first step along these lines was made by Yao [102]. He proved a lower bound of $\frac{3}{2}$ based on three lists of equal-size elements, the sizes being $\frac{1}{2} + \varepsilon$, $\frac{1}{3} + \varepsilon$ and $\frac{1}{6} - 2\varepsilon$. Using Salzer's sequences, Brown [13] and Liang [85] independently gave a further improvement to 1.53634577.... (The largest sizes in their sequences were: $\frac{1}{2} + \varepsilon$, $\frac{1}{3} + \varepsilon$, $\frac{1}{7} + \varepsilon$, $\frac{1}{43} + \varepsilon$, and $\frac{1}{1807} + \varepsilon$.) Galambos [42] used the Golomb sequences to extend the idea to general α . The proof in [42] was considerably simplified by Galambos and Frenk [43]. Van Vliet gave an exhaustive analysis of the lower bound constructions with a linear programming technique applied to all α : his lower bound is the best so far.

Theorem 3.17 (van Vliet [96]). *For any on-line algorithm A , $R_A^{\infty} \geq 1.540$.*

Table 3 gives a comparison for several values of $\alpha = \frac{1}{r}$ between the best lower bounds and the corresponding upper bounds for various algorithms. It is interesting to note that the gap between the lower and upper bounds becomes rather small for $r \geq 2$.

Chandra [17] has examined the effect on lower bounds when randomization is allowed in the construction of on-line algorithms, i.e., when coin

flips are allowed in determining where to pack items. The performance ratio for randomized algorithm A is now $E[A(L)]/OPT(L)$, where $E[A(L)]$ is the expected number of bins needed by A to pack the items in L . Chandra has shown that there are lists such that this ratio exceeds 1.536 for all randomized on-line algorithms, and so from this limited standpoint, results suggest that randomization is not a valuable tool in the design of on-line algorithms.

4 Off-Line Algorithms

An *off-line* algorithm has all items available for preprocessing, reordering, grouping, etc. before packing. We have seen that most of the classical on-line algorithms achieve their worst-case ratio when the items are packed in increasing order of size (see, e.g., FF and BF), or if small and large items are merged (see, e.g., NF). Thus, one is led to expect improved behavior by a sorting of the items in decreasing order of size. Note that $O(n \log n)$ sorting steps puts us outside the class of linear-time algorithms.

We start with results for approaches that sort the items before executing one of the on-line algorithms. We then consider linear-time heuristics. We conclude this section with approximation schemes and a discussion of the anomalous behavior that is exhibited by many bin packing algorithms, including both on-line and off-line algorithms.

4.1 Algorithms with presorting

When the sorted list is packed according to the Next-Fit rule, we obtain the *Next-Fit Decreasing* (NFD) algorithm. This heuristic was investigated by Baker and Coffman, who proved by a weighting function argument that its APR is slightly better than that of FF and BF:

Theorem 4.1 (Baker and Coffman [8]). *If $\alpha \in (\frac{1}{r+1}, \frac{1}{r}]$ ($r \geq 1$) then $R_{\text{NFD}}^\infty(\alpha) = h_\infty(r)$.*

Packing the sorted list according to First-Fit or Best-Fit gives the algorithms *First-Fit Decreasing* (FFD) and *Best-Fit Decreasing* (BFD), with much better asymptotic worst-case performance.

Theorem 4.2 (Johnson et al. [72]). $R_{\text{FFD}}^\infty = R_{\text{BFD}}^\infty = \frac{11}{9}$.

The original proof was based on the weighting function technique but subsequent proofs introduced dramatic changes; the giant case-analysis of

Johnson [69] was considerably shortened by Baker [6], and recently Yue [104] presented the shortest proof known so far. In parallel, the additive constant was also improved; Johnson had proved that $\text{FFD}(L) \leq \frac{11}{9}OPT(L) + 4$, but Baker reduced the constant to 3, and Yue reduced it to 1.

The behavior of BFD for general α is not known, but that of FFD has been intensively investigated. Johnson et al. [72] analyzed several cases, showing that

$$R_{\text{FFD}}^{\infty}(\alpha) = \begin{cases} \frac{11}{9} & \text{if } \frac{1}{2} < \alpha \leq 1 \\ \frac{71}{60} & \text{if } \frac{8}{29} < \alpha \leq \frac{1}{2} \\ \frac{7}{6} & \text{if } \frac{1}{4} < \alpha \leq \frac{8}{29} \\ \frac{23}{20} & \text{if } \frac{1}{5} < \alpha \leq \frac{1}{4} \end{cases}$$

and conjecturing that, for any integer $m \geq 4$,

$$R_{\text{FFD}}^{\infty}\left(\frac{1}{m}\right) = F_m := 1 + \frac{1}{m+2} - \frac{2}{m(m+1)(m+2)}$$

Twenty years later, Csirik [28] proved that the above conjecture is valid only for m even, and that, for m odd,

$$R_{\text{FFD}}^{\infty}\left(\frac{1}{m}\right) = G_m := 1 + \frac{1}{m+2} - \frac{1}{m(m+1)(m+2)}$$

In the same year, the complete analysis of FFD for arbitrary values of $\alpha \leq \frac{1}{4}$ was published by Xu [101]. He showed that if m is even, then F_m is the correct APR for any α in $(\frac{1}{m+1}, \frac{1}{m}]$, while for m odd the interval has to be divided into two parts, with

$$R_{\text{FFD}}^{\infty}(\alpha) = \begin{cases} F_m, & \text{if } \frac{1}{m+1} < \alpha \leq d_m \\ G_m, & \text{if } d_m < \alpha \leq \frac{1}{m} \end{cases}$$

where $d_m := (m+1)^2/(m^3 + 3m^2 + m + 1)$.

Recall that, after a presorting stage, all of the above algorithms belong to the Any-Fit class (see Section 3.3). Johnson showed that, after a presort in increasing order, Any-Fit algorithms do not perform well in the worst case; e.g., their APRs must be at least $h_{\infty}(1)$ when $\alpha = 1$. But presorting in decreasing order is much more useful.

Theorem 4.3 (Johnson [69, 70]). *Any algorithm $A \in \mathcal{AF}$ operating on a list presorted in decreasing-size order must have*

$$\begin{aligned} \frac{11}{9} &\leq R_A^{\infty}(1) \leq \frac{5}{4} \\ \frac{1}{m+2} - \frac{2}{m(m+1)(m+2)} &\leq R_A^{\infty}(\alpha) \leq \frac{1}{m+2}, \quad \text{where } m = \lfloor \frac{1}{\alpha} \rfloor \text{ and } \alpha < 1 \end{aligned}$$

For a long time, the FFD bound was the smallest proved APR. Johnson [69] made an interesting attempt to obtain a better APR. His *Most-k-Fit* (MF_k) algorithm takes elements from both ends of the sorted list, packing bins one at a time. At any step, after trying to place the largest unpacked element into the current bin, the algorithm attempts to fill up the remaining space in the bin using the smallest k (or fewer) as yet unpacked items. As soon as the available space becomes smaller than the smallest unpacked element, the algorithm starts packing a new bin. The algorithm has time complexity $O(n^k \log n)$, so it is practical only for small k . Johnson conjectured that $\lim_{k \rightarrow \infty} R_{\text{MF}_k}^\infty = \frac{10}{9}$, but almost twenty years later the conjecture was contradicted by Friesen and Langston [40], who gave examples for which $R_{\text{MF}_k}^\infty \geq \frac{5}{4}$, $k \geq 2$.

Yao [102] devised the first improvement to FFD. He presented a complicated $O(n^{10} \log n)$ algorithm, called *Refined First-Fit Decreasing* (RFFD), with worst-case ratio $R_{\text{RFFD}}^\infty \leq \frac{11}{9} - 10^{-7}$. Following this result, further efforts were made to develop better off-line algorithms. Garey and Johnson [52] proposed the *Modified First-Fit Decreasing* (MFFD) algorithm. The main idea is to supplement FFD with an attempt to improve that part of the packing containing bins with items of sizes larger than $\frac{1}{2}$, by trying to pack in these bins pairs of items (to be called S items) with sizes in $(\frac{1}{6}, \frac{1}{3}]$. The non-FFD decisions of MFFD occur only during the packing of S items. At the time these items come up for packing, the bins currently containing a single item larger than $\frac{1}{2}$ are packed first, where possible, in decreasing-gap order as follows. In packing the next such bin, MFFD first checks whether there are two still-unpacked S items that can fit into the bin; if not, MFFD finishes out the remaining packing just like FFD. Otherwise, the smallest available S item is packed first in the bin; the largest remaining available S item that fits with it is packed second. The running time of MFFD is not appreciably larger than that for FFD, but Garey and Johnson proved that

Theorem 4.4 (Garey and Johnson [52]). $R_{\text{MFFD}}^\infty = \frac{71}{60} = 1.18333\dots$

Another modification of FFD was presented by Friesen and Langston [40]. Their *Best Two-Fit* (B2F) algorithm starts by filling one bin at a time, greedily; when no further element fits into the current bin, and the bin contains more than one element, an attempt is made to replace the smallest one by two unpacked elements with sizes at least $\frac{1}{6}$. When all the unpacked elements have sizes smaller than $\frac{1}{6}$, the standard FFD algorithm is applied. Friesen and Langston proved that $R_{\text{B2F}}^\infty = \frac{5}{4}$, which is worse than

$\frac{11}{9}$. However, they further showed that a combined algorithm (CFB), which runs both B2F and FFD and takes the better packing, has an improved APR.

Theorem 4.5 (Friesen and Langston [40]). $1.16410\dots = \frac{227}{195} \leq R_{\text{CFB}}^{\infty} \leq \frac{6}{5} = 1.2$.

4.2 Linear time, randomization, and comparisons

The off-line algorithms analyzed so far have time complexity at least $O(n \log n)$. It is also interesting to see what can be obtained in linear time – in particular, without sorting. The first such heuristic was constructed by Johnson [70]. His *Group-X Fit Grouped* (GXFG) algorithm depends on the choice of a set of breakpoints X , defined by a sequence of real numbers $0 = x_0 < x_1 < \dots < x_p = 1$. For a given X , the algorithm partitions the items, according to their size, into at most p classes, and renames them in such a way that items of the same class are consecutive and classes are ordered by decreasing maximum size. The bins are also collected into p groups according to their *actual gap*, defined as the maximum x_j such that the current empty space in the bin is at least x_j . The items are packed using the Best-Fit rule with respect to the actual gaps. The algorithm can be implemented so as to require linear time, and has the following APR.

Theorem 4.6 (Johnson [70]). *For all $m \geq 1$, if X contains $\frac{1}{m+2}, \frac{1}{m+1}$ and $\frac{1}{m}$, then $R_{\text{GXFG}}^{\infty}(\alpha) = \frac{m+2}{m+1}$ for all $\alpha \leq 1$ such that $m = \left\lfloor \frac{1}{\alpha} \right\rfloor$.*

For $\alpha = 1$ the above theorem gives $R_{\text{GXFG}}^{\infty} = \frac{3}{2}$, a bound subsequently improved by Martel. His algorithm, H_4 , uses a set $X = \{\frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$, but it does not reorder the items. It instead inserts them into heaps, and uses a linear search for the median-size item. The packing strategy makes use of an elaborate pairing technique.

Theorem 4.7 (Martel [88]). $R_{H_4}^{\infty} = \frac{4}{3}$.

Very recently, Békési and Galambos [11] applied the Martel idea to a set $X = \{\frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{3}{8}, \frac{1}{2}, \frac{2}{3}, \frac{4}{5}\}$, and improved the bound to $\frac{5}{4}$. Making experimental comparisons, they further showed that this linear-time algorithm is faster than the less complicated FFD rule even for small problem instances. They also mentioned that, by using more breakpoints, one can further improve the above result, but the resulting algorithms would be linear-time with a constant term so large that they would not be useful in practice.

Algorithm	Time	$R_A^\infty(1)$	$R_A^\infty(2)$	$R_A^\infty(3)$	$R_A^\infty(4)$
NFD	$O(n)$	1.691...	1.424...	1.302...	1.234
FFD	$O(n \log n)$	1.222...	1.183...	1.183...	1.15
BFD	$O(n \log n)$	1.222...	1.183...	1.183...	1.15
GXFD	$O(n)$	1.5	1.333...	1.25	1.2
MFFD	$O(n \log n)$	1.183...	1.183...	1.183...	1.15
H ₄	$O(n)$	1.333...	NA	NA	NA
H ₇	$O(n)$	1.25	NA	NA	NA
B2F	$O(n \log n)$	1.25	NA	NA	NA
CFB	$O(n \log n)$	1.2	$\leq 1.183...$	$\leq 1.183...$	≤ 1.15

Table 4: Worst-case ratios of off-line algorithms. NA means “not analyzed”.

To this point, if we consider the established tight bounds, the best is MFFD. However, very recently Kenyon [76] presented an algorithm that could be the new best-in-class, assuming randomization is allowed; her *Best-Fit Randomized* (BFR) algorithm produces a series of packings by applying the Best-Fit strategy to random permutations of the items. If we measure the efficiency by the expected number of bins, $E(A(L))$, then we have

$$\frac{227}{195} \leq \limsup_{k \rightarrow \infty} \left\{ \max \frac{E(A(L))}{OPT(L)} : OPT(L) = k \right\} \leq \frac{3}{2}$$

Although this is worse than R_{MFFD}^∞ , no example has been found so far such that the expected ratio is greater than 1.144.

Table 4 summarizes the best worst-case ratios of off-line algorithms (H₇ denotes the algorithm of Békési and Galambos [11]).

As a final note on fast off-line algorithms, we direct the reader to the work of Anderson, Mayr, and Warmuth [3] for the implementation of approximation algorithms on parallel architectures. They show that, with $n / \log n$ processors in the EREW PRAM model, a packing can be obtained in parallel in $O(\log n)$ time which has the same asymptotic $\frac{11}{9}$ bound as FFD.

4.3 Asymptotic approximation schemes

In 1980, Yao [102] raised an interesting question: Does there exist an $\varepsilon > 0$ such that every $O(n)$ -time algorithm A must satisfy the lower bound $R_A^\infty \geq 1 + \varepsilon$? Fernandez de la Vega and Lueker [36] answered the question in the negative by constructing an asymptotic linear approximation scheme for bin

packing. In their important paper, LP (linear programming) relaxations were first introduced as a technique for devising bin packing approximation algorithms. (See Section 5.2 for an integer programming formulation of the bin packing problem.) In this section, we first discuss their result and then mention some improvements proposed by Johnson and by Karmarkar and Karp. Further discussion can be found in [23, 63].

The main idea in [36] is the following. Given an ε , $0 < \varepsilon < 1$, define ε_1 so that $\frac{\varepsilon}{2} < \varepsilon_1 \leq \frac{\varepsilon}{\varepsilon+1}$. Instead of packing the given list L , we pack a concatenation of three lists $L_1 L_2 L_3$ determined as follows.

- L_1 contains all the elements of L with sizes smaller than ε_1 .
- L_2 is a list of $(m - 1)h$ dummy elements with “rounded” sizes, corresponding to the $(m - 1)h$ smallest elements of $L \setminus L_1$, where $m = \lceil \frac{4}{\varepsilon^2} \rceil$ and $h = \left\lfloor \frac{|L \setminus L_1|}{m} \right\rfloor$. The elements of L_2 have only $m - 1$ different sizes, and, for each size s , the list has h elements; the sizes of the corresponding h elements in $L \setminus L_1$ are no greater than s .
- L_3 contains the remaining elements of $L \setminus L_1$, i.e., those not having a corresponding rounded element in L_2 . We have $|L_3| \leq 2h - 1$.

By construction, for each group of h elements of L_2 having the same size s , there exists a distinct group of h elements of $L \setminus L_1$ having sizes at least s . It follows that $OPT(L_2) \leq OPT(L \setminus L_1)$.

It is first proved that for a given ε , one can construct, in linear time via an LP relaxation, a packing for the elements of L_2 that requires no more than $OPT(L \setminus L_1) + \frac{4}{\varepsilon}$ bins. The next step is to pack each element of L_3 into a separate bin, so the number of open bins at this point is bounded by

$$OPT(L \setminus L_1) + \frac{4}{\varepsilon} + 2h - 1 < OPT(L \setminus L_1) + \frac{4}{\varepsilon} + |L \setminus L_1| \frac{\varepsilon^2}{2} \leq OPT(L \setminus L_1)(1 + \varepsilon) + \frac{4}{\varepsilon}$$

(using the fact that $OPT(L \setminus L_1) \geq \frac{\varepsilon}{2}|L \setminus L_1|$).

Finally, the items of L_1 are added to the current packing, one bin at a time, using the Next-Fit rule. If no new bin is opened in this phase, the resulting packing has the desired performance. If at least one new bin is opened, we know that all the bins, except possibly the last one, are filled to at least $(1 - \varepsilon_1)$, and hence the total number of bins is bounded by

$$\frac{OPT(L)}{1 - \varepsilon_1} + 1 \leq OPT(L)(1 + \varepsilon) + 1 \leq OPT(L)(1 + \varepsilon) + \frac{4}{\varepsilon}$$

Combining this with an analysis of the time to pack L_2 , Fernandez de la Vega and Lueker proved the following result.

Theorem 4.8 (Fernandez de la Vega and Lueker [36]). *For any $\varepsilon > 0$, there exists a bin packing algorithm A with asymptotic worst-case ratio $R_A^\infty \leq 1 + \varepsilon$, requiring time $C_\varepsilon + Cn \log \frac{1}{\varepsilon}$, where C_ε depends only on ε and C is an absolute constant.*

Although the time complexity of the above approximation scheme is polynomial in the number of elements, it is exponential in $\frac{1}{\varepsilon}$.

The first improvement was given by Johnson [71], who observed that, if ε is allowed to grow suitably slowly with $OPT(L)$, one can use the above approach to construct a polynomial-time algorithm A such that $A(L) \leq OPT(L) + o(OPT(L))$; incorporating a scheme suggested by Karmarkar and Karp, he achieved $A(L) \leq OPT(L) + O(OPT(L)^{1-\delta})$, where δ is a positive constant. Thus, as a corollary to Theorem 4.8, there exists a polynomial-time approximation algorithm for bin packing with an APR equal to 1.

Karmarkar and Karp [74] made further improvements and produced an asymptotic fully polynomial-time approximation scheme. They brought several new techniques into play. The ellipsoid method applied to an LP relaxation drove the approach; a feasible packing was determined by an extension of the approach of Fernandez de la Vega and Lueker. A function T , estimated below, describes the time complexity of the LP problem in terms of properties of the problem instance. Let $c(L)$ denote the total size of the items in L and recall that k denotes the number, perhaps infinite, of possible item sizes. The main results are several different forms of asymptotic optimality and their associated running-time bounds. Recalling that the absolute error for an algorithm A is simply $A(L) - OPT(L)$, we have the following

Theorem 4.9 (Karmarkar and Karp [74]). *For each row in the table below, there is an approximation algorithm with the given running-time and absolute-error bounds; the approximation parameters in the last two rows satisfy $\alpha \in [0, 1]$ and $\epsilon > 0$.*

running-time bound	absolute-error bound
$O(T_n(c(L)))$	$O(\log^2 OPT(L))$
$O(T_n(k))$	$O(\log^2 k)$
$O(T_n(c(L)^{1-\alpha}))$	$O(OPT(L)^\alpha)$
$O(T_n(\varepsilon^{-2}))$	$\varepsilon OPT(L) + O(\varepsilon^{-2})$

where

$$T_n(v) = O(v^8 \log v \log^2 n + v^4 n \log v \log n)$$

The bound on $T_n(v)$ is not especially attractive, so this approach as it stands is not likely to be very practical, although the authors mention that a mixture of their technique with a column generation method [53, 54] may be very efficient in practice.

4.4 Anomalies

For a given algorithm A , one usually expects that if a list is made shorter, or its items made smaller, then the number of bins needed by A to pack the list could not increase; and if the reductions were large enough, then the number of bins required would actually decrease. This is certainly true of optimal algorithms, but as this section shows, it is not the case for many of the better on-line and off-line approximation algorithms. This anomalous behavior can explain the difficulty in analyzing algorithms; with such behavior, inductive arguments can not normally be expected to work. The following instances illustrate bin packing anomalies. Define

$$\begin{aligned} L &= (0.7, 0.68, 0.5399, 0.3201, 0.15, 0.14, 0.08, 0.08, 0.08, 0.08, 0.08, 0.07) \\ L' &= (0.7, 0.68, 0.5399, 0.3200, 0.15, 0.14, 0.08, 0.08, 0.08, 0.08, 0.08, 0.07) \end{aligned}$$

and note that the two instances differ only in the fourth element, which is slightly smaller in L' . BF and BFD produce the same packing of L :

$$\begin{aligned} c(B_1) &= 0.7 + 0.15 + 0.08 + 0.07 \\ c(B_2) &= 0.68 + 0.08 + 0.08 + 0.08 + 0.08 \\ c(B_3) &= 0.5399 + 0.3201 + 0.14 \end{aligned}$$

BF and BFD also produce the same packing of L' , which is larger by one bin:

$$\begin{aligned} c(B_1) &= 0.7 + 0.15 + 0.14 \\ c(B_2) &= 0.68 + 0.32 \\ c(B_3) &= 0.5399 + 0.08 + 0.08 + 0.08 + 0.08 + 0.08 \\ c(B_4) &= 0.07 \end{aligned}$$

We say that a list $L_{(n)} = (a_1, a_2, \dots, a_n)$ *dominates* a list $L'_{(m)} = (a'_1, a'_2, \dots, a'_m)$ if $n \geq m$ and $s(a_i) \geq s(a'_i)$ for $i = 1, 2, \dots, m$.

An algorithm A is *monotonic* if $A(L) \geq A(L')$ whenever L dominates L' . An *anomalous* algorithm is one that is not monotonic.

Graham [58] and Johnson [69] observed that algorithms FF and FFD are anomalous. However, our current insights into the anomalous behavior of bin packing algorithms are due mainly to Murgolo. In the notation of Section 3.4 (with W meaning Worst-Fit), he proved

Theorem 4.10 Murgolo [90]: *Algorithms NF and NF₂ are monotonic but each of BF, BFD, WF, WFD, ABF₂, AWF₂, and AFF_k with k ≥ 3 is anomalous.*

Murgolo also obtained upper and lower bounds on the behavior of anomalous algorithms. An algorithm is called *conservative* if it never opens a new bin unless the current element can not fit into an open bin. For conservative algorithms, the following bound applies.

Theorem 4.11 Murgolo [90]: *If L' dominates L and A is conservative then $A(L') \leq 2A(L)$. In addition, if $A \in \{FFD, BFD\}$ then $A(L') \leq \frac{11}{9}A(L)+4$.*

He also proved the following complementary results.

Theorem 4.12 Murgolo [90]: *There exist arbitrarily long lists L_i and L'_i , with L'_i dominating L_i , ($i = 1, 2, 3$), such that*

$$\begin{aligned} &\text{if } A \in \{BF, BFD\} \quad \text{then } A(L_1) \geq \frac{43}{42}A(L'_1), \\ &\text{if } A \in \{FF\} \quad \text{then } A(L_2) \geq \frac{76}{75}A(L'_2), \\ &\text{if } A \in \{WF, WFD\} \quad \text{then } A(L_3) \geq \frac{16}{15}A(L'_3). \end{aligned}$$

5 Variations and Special Lists

In this section, we discuss problems in which special types of lists, alternative objective functions, or any of various constraints on items or packings are considered. We will see that in some cases the problem becomes easier, in the sense of approximability, while in others it becomes harder. Similarly, adaptations of classical heuristics give more or less attractive results depending on the new problem.

5.1 Semi-on-line algorithms

We know that the APR of on-line algorithms cannot be less than 1.540... (see Section 3.6). For almost twenty years only the pure on-line and off-line algorithms were analyzed, and no attention was paid to algorithms lying between these two classes. In a more general setting, we can consider giving the algorithm more information about the list and/or more freedom with respect to the pure on-line case. Here, we will consider semi-on-line algorithms that can

- repack elements;
- look ahead to later elements before assigning the current one;
- assume some preordering of the elements.

We first consider the case where repacking is allowed. We have seen in Section 3.4 that no known on-line bounded-space algorithm reaches the bound $h_\infty(1)$ using finitely many open bins. We will see that this is possible with semi-on-line algorithms.

In 1985, Galambos [41] made a first step in this direction. His *Buffered Next-Fit* (BNF) algorithm uses two open bins, say B_1 and B_2 . The arriving elements are initially packed into B_1 , until the first element arrives for which B_1 does not have enough space. This element and those currently packed in B_1 are then reordered by decreasing size and repacked in B_1 and B_2 following the Next-Fit rule. B_1 is now closed, B_2 is renamed B_1 , and a new bin B_2 is opened. Using a weighting function approach, Galambos [41] proved that $h_\infty(1) \leq R_{\text{BNF}}^\infty \leq \frac{18}{10}$.

Galambos and Woeginger [45] generalized the above idea, adopting a better weighting function. Let $w(B)$ be the sum of the weights associated with the items currently packed in bin B . Their *Repacking* algorithm (REP₃) uses three open bins. When a new item a_i arrives, the following steps are performed: (i) a_i is packed into an empty bin; (ii) all the elements in the three open bins are repacked by the FFD strategy, with the result that either one bin becomes empty or at least one bin B has $w(B) \geq 1$; (iii) all bins B with $w(B) \geq 1$ are closed and replaced by new empty bins. In [45], it was proved that this repacking in fact helps, since $R_{\text{REP}_3}^\infty = h_\infty(1)$. It is not known whether the same result can be obtained with two bins.

Gambosi, Postiglione and Talamo [47] were the first to beat the 1.540 on-line bound via repacking. They gave two algorithms. In the first one, A₁, the interval (0, 1] is divided into 4 subintervals, and the elements are packed

into bins in a Harmonic-like way. As each new item is packed, groups of small elements can be repacked so as to fill up gaps in bins that are not full enough. By using appropriate data structures, this repacking is performed in constant time. The algorithm has linear time and space complexity, and it has an APR, $R_{A_1}^{\infty} \leq \frac{3}{2}$. In the second algorithm, A_2 , the unit interval is divided into 6 subintervals, and, as each new item is encountered, the elements are repacked more carefully, in $O(\log n)$ time, by means of a pairing technique analogous to that introduced by Martel (see Section 4.2). The time complexity of A_2 is $O(n \log n)$ and its APR is $R_{A_2}^{\infty} \leq \frac{4}{3}$.

Ivkovic and Lloyd [65] gave a further improvement achieving a $\frac{5}{4}$ worst-case ratio. Their algorithm is much more complicated than the previous ones, as it was designed for handling the dynamic packing case (see Section 5.4). They also proved a lower bound for the special class of semi-on-line algorithms that only use *atomic* repacking moves; such a move is limited to the transfer of a single item from one bin to another. Then

Theorem 5.1 (Ivkovic and Lloyd [66]). *For any semi-on-line algorithm A that makes only a bounded number of atomic repacking moves at each step, the APR satisfies $R_A^{\infty} \geq \frac{4}{3}$.*

Quite recently, Ivkovic and Lloyd [67] presented approximation schemes for their dynamic, semi-on-line model, applying the techniques of Fernandez de la Vega and Lueker and those of Karmarkar and Karp cited in Section 4.3.

We now consider the case in which the algorithm is allowed to look ahead, in the sense that, when an element arrives, it is not necessary to pack it immediately; one is allowed to collect further elements whose sizes can effect the packing decision. In order to avoid degeneration to an off-line algorithm, we will consider the case of *bounded* look-ahead. Grove [59] proposed a k -bounded algorithm which had, in addition, a capacity (or *warehouse*) constraint W . The algorithm can delay the packing of item a_i until it has collected all subsequent elements a_{i+1}, \dots, a_j such that $\sum_{r=i}^j a_r \leq W$. For any fixed k and W the $h_{\infty}(1)$ lower bound (see Theorem 3.12) remains valid, but Grove's *Revised Warehouse* (RW) algorithm reaches the bound if W is sufficient large. In his proof, Grove uses a weighting function argument.

Finally, we examine the rarely considered class of algorithms in which it is assumed that the input list is *presorted*. Because of the lower bound constructions, it is easy to see that, if the list is presorted by increasing size, the on-line lower bounds remain valid. Moreover, the Johnson result holds: if the list is preordered by decreasing item size then $\frac{11}{9} \leq R_A^{\infty} \leq \frac{5}{4}$ for any

algorithm $A \in \mathcal{AF}$ (see Theorem 4.3). This begged the broader question of lower bounds: how good can an arbitrary algorithm be? A partial answer is given by the following result.

Theorem 5.2 (Csirik, Galambos and Turan [29]). *If the list is preordered by decreasing size, then $R_A^\infty \geq \frac{8}{7}$ for all algorithms A .*

This is the best lower bound known.

5.2 Item-size restrictions

Perhaps the simplest (and most practical) restriction is simply that we have a finite number k of item sizes, say s_1, \dots, s_k , and thus a finite number N of feasible item configurations in a bin. This class of problems arose in the work on approximation schemes [36, 74], and was considered in its own right by Blazewicz and Ecker [12]. Let the i th configuration be denoted by $p_i = (p_{i1}, \dots, p_{ik})$ where p_{ij} is the number of items of size s_j in p_i . By the definition of feasibility, $\sum_{j=1}^k p_{ij} s_j \leq 1$ for all i . Let b_i be the number of bins in a packing of a given list L that contain configuration p_i , and let n_j be the number of items of size s_j in L . Then a solution to the bin packing problem for L is a solution to the integer programming formulation

$$\begin{aligned} \min \quad & \sum_{i=1}^N b_i \\ & \sum_{i=1}^N p_{ij} b_i \geq n_j \quad (j = 1, \dots, k) \\ & b_i \geq 0 \text{ and integer } (i = 1, \dots, N) \end{aligned}$$

Note that, if $\frac{1}{r}$ lower bounds the item sizes, then $N = O(k^{r-1})$, so the constraint matrix above has k rows and $O(k^{r-1})$ columns. This problem can be solved in time polynomial in the number of constraints (see Lenstra [84]), which is a constant independent of the number of items. The optimal overall packing can then be constructed in linear time.

If we do not need the exact solution, then we can use the classical approach of Gilmore and Gomory [53, 54]) and compute LP relaxations (see Section 4.3); this method gives solutions that are at most k off the optimal in significantly less time.

We consider next the classical problem restricted to lists L of items whose sizes form a *divisible sequence*, i.e., the distinct sizes $s(a_1) > s(a_2) > \dots$

taken on by the items are such that $s(a_{i+1})$ divides $s(a_i)$ for all $i \geq 1$. The number of items of each size is arbitrary. Given the bin capacity C , the pair (L, C) is *weakly divisible* if L has divisible item sizes and *strongly divisible* if in addition the largest item size $s(a_1)$ in L divides C .

This variant has practical applications in computer memory allocation, where device capacities and memory block sizes are commonly restricted to powers of 2. It is important to recognize such applications when they are encountered, because approximation algorithms for many types of NP-hard bin packing problems generate significantly better packings when items satisfy divisibility constraints. We emphasize here the cases where the restriction leads to algorithms that are asymptotically optimal. The problem was studied by Coffman, Garey and Johnson, who proved the following result for classical off-line algorithms.

Theorem 5.3 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible, then NFD and FFD packings are optimal.*

Indeed, the residual capacity in a bin is always either zero or at least as large as the last (smallest) item packed in the bin. The last packed item is at least as large as any remaining (unpacked) item, so either the bin is totally full or it has room for the next item. Therefore the FFD and NFD algorithms have the same behavior: they initialize a new bin only when the previous one is totally full, so the packings they produce are identical and perfect.

The optimality of FFD holds in the less restrictive case of the following theorem as well.

Theorem 5.4 (Coffman, Garey and Johnson [20]). *If (L, C) is weakly divisible, then an FFD packing is always optimal.*

For strongly divisible instances, we can also obtain optimal performance without sorting the items.

Theorem 5.5 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible, then an FF packing is always optimal.*

It would be interesting to examine how approximation algorithms behave with other special size sequences (e.g., Fibonacci numbers), as mentioned by Chandra, Hirschler and Wong [16] in the context of memory allocation.

5.3 Variable size bins

Taking a complementary approach to the approximation schemes of Section 4.3, Hochbaum and Shmoys [61] define an ε -dual approximation algorithm for bin packing, which we denote by M_ε . For a given $\varepsilon > 0$, M_ε finds in polynomial time a packing of L in $OPT(L)$ bins, each of capacity $C = 1 + \varepsilon$. A primary objective of M_ε is the approximation scheme for the capacity minimization problem discussed in Section 5.6, but such algorithms also find use in bin packing settings where precise bin capacities vary or are unknown.

We remark that the above dual approach can be likened to a search for near-feasible, optimal solutions rather than feasible, near-optimal solutions. The construction of M_ε exploits a reduction of the problem to bin packing with a finite number of item sizes, and hence the integer program formulation in Section 5.2. A recent, general discussion of the details has been given by Hochbaum [63], who describes a version of M_ε that requires only linear running time (with constants depending on ε).

Consider now the case where we are given different bin types, B_1, B_2, \dots, B_k with sizes $1 = s(B_1) > s(B_2) > \dots > s(B_k)$. For each type, an unlimited number of bins is available, and the aim is to pack a list L of elements a_i , with $s(a_i) \in [0, 1]$, into a subset of bins having the smallest total size. Let $s(A, L)$ denote the total size of the bins used by algorithm A to pack the items of L . Then

$$R_A^\infty = \lim_{k \rightarrow +\infty} \sup \left\{ \frac{s(A, L)}{s(OPT, L)} : s(OPT, L) \geq k \right\}$$

The problem has practical importance in many of the classical bin packing applications, e.g., cutting-stock problems, the assignment of commercials to variable size breaks in television or radio broadcasting, memory allocation for computer operating systems, etc.

At first glance, one might expect variable size bin packing to be harder than the classical problem. However, this need not be true in general. For example, if we have bin types for all sizes between $\frac{1}{2}$ and 1, then packing large elements with sizes at least $\frac{1}{2}$ can be done without wasted space. Similarly, we can pack smaller elements together without any waste, so long as their sum is at least $\frac{1}{2}$.

For general sets of bin sizes, an on-line algorithm must select the bin size for packing the current element whenever a new bin is opened. Friesen and Langston [39] proposed an on-line algorithm based on the Next-Fit rule,

which always selects the largest bin-size (*Next-Fit, using Largest possible bins*, NFL), and proved that its APR is 2. Kinnersley and Langston [78] showed that the same APR is obtained by adopting the First-Fit rule, both for an algorithm that uses the largest possible bins and for an algorithm that uses the smallest bins. Very recently, Burkard and Zhang [14] pointed out that this APR characterizes a large class of algorithms that use one of the above bin-opening rules.

Kinnersley and Langston [78] analyzed other fast on-line algorithms. They proposed a scheme based on a user-specified *fill factor*, $f \geq \frac{1}{2}$, and proved that this strategy guarantees an APR smaller than $\frac{3}{2} + \frac{f}{2} \leq 1.75$. Zhang [106] proved that, with $f = \frac{1}{2}$, the APR is $\frac{17}{10}$.

Csirik [27] investigated an algorithm based on the Harmonic-Fit strategy (*Variable Harmonic-Fit*, VH), and showed that its APR is at most $h_\infty(1) \approx 1.69103$. For special collections of bin types the algorithm may perform better; for example, Csirik proved that if there are only two types of bins, with $s(B_1) = 1$ and $s(B_2) = \frac{7}{10}$, we have $R_{\text{VH}}^\infty = \frac{7}{5}$. This result is very interesting since the bound is smaller than the 1.54 on-line lower bound of van Vliet [96] for the classical problem (see Section 3.6), and implies that, for certain sets of two or more bin sizes, on-line algorithms can behave better than those restricted to a single bin size. This raised further, still unanswered questions. If there are only two different bin types, which combination of sizes produces the smallest APR? What lower bounds depending on bin sizes can be proved? What can be said about the problem with at least three bin sizes?

Csirik's VH algorithm suffers from the same “illness” as the classical HF_k algorithm (see Section 3.4); it reaches the $h_\infty(1)$ bound only for a very large number of open bins. More precisely, let $M > 1$ be a positive integer, and suppose that the algorithm can use l different bin types. Let $M_j = \lceil M s(B_j) \rceil$ ($j = 1, \dots, l$), and denote by VH_M the resulting VH algorithm, which is a k -bounded space algorithm with $k = \sum_{j=1}^l M_j - l + 1$. If $M_l \leq 5$ then $R_{\text{VH}_M}^\infty \geq \frac{17}{10}$.

Burkard and Zhang [14] investigated other bounded-space algorithms for variable-size bin packing. They distinguished three different components: the opening rule, packing rule, and closing rule. The opening rule was fixed as follows. If the current item has size $s(a_i) > \frac{1}{2}$ and there exist bin sizes smaller than 1 that can accommodate a_i , then the rule opens the smallest such bin; otherwise, it opens a bin of size 1. The packing rules analyzed were First-Fit (the F rule) and Best-Fit (the B rule). The closing rule closes

a bin of size less than 1, if one exists; otherwise, it closes either the lowest indexed bin (the first-bin or F rule) or the most nearly full bin (the best-bin or B rule). With this terminology, VXY_k denotes the k -bounded algorithm that incorporates packing rule X and closing rule Y . Burkard and Zhang showed that, among the four resulting algorithms (VFF_k , VFB_k , VBF_k and VBB_k), VBB_k is the best, and that the following holds.

Theorem 5.6 (Burkard and Zhang [14]). $R_{VBB_k}^\infty = \frac{17}{10}$. Moreover, $R_{VH_M}^\infty \geq R_{VBB_k}^\infty$ if and only if $M_l < 7$, with $k \geq 6l + 1$.

The authors proved the theorem by a weighting function technique. They also mentioned that, with a small modification in the weighting function, they could prove that $R_{VFB_k}^\infty = \frac{17}{10} + \frac{3}{10(k-1)}$ for any $k \geq 2$. So, one of the more interesting questions remains open: Does there exist an on-line algorithm with an APR strictly less than $h_\infty(1)$ for all collections of bin sizes?

For the off-line case, few results have been proved. Friesen and Langston [39] presented two algorithms based on the First-Fit Decreasing strategy. The first one, *FFD using Largest bins, and Repack* (FFDLR), begins by packing the presorted elements into the largest bins, then repacks the contents of each open bin into the smallest possible empty bin. The second algorithm, *FFD using Largest bins and Shift* (FFDLS), improves on the first phase of FFDLR: whenever the bin (say of type B_j) where the current item has been packed contains an item of size at least $\frac{1}{3}$, the contents of the bin are shifted, if possible, to the smallest empty bin (say of type B_h) such that $s(B_h) \geq \tilde{c} \geq \frac{3}{4}s(B_h)$, where \tilde{c} denotes the total item size packed into the current bin. Friesen and Langston proved that $R_{FFDLR}^\infty = \frac{3}{2}$ and $R_{FFDLS}^\infty = \frac{4}{3}$.

Murgolo [89] proposed an efficient approximation scheme. He first gave an algorithm with a running time linear in the number of items but exponential in $\frac{1}{\epsilon}$ and the number of bin sizes. Then, following the ideas in Karmarkar and Karp [74], he solved a linear programming formulation of the problem by the ellipsoid method, thus obtaining a fully polynomial-time approximation scheme.

Very recently, Zhang [107] considered a variant of the on-line version of the problem, in which item-size information is known in advance, but no information on bin sizes is available, except that each bin size is known to be no less than the size of the largest item. Bins arrive one at a time, and we have to decide which items to pack into the current bin. Zhang proved

that the analogues of the classical NF, FF, NFD and FFD algorithms all have APR equal to 2, and left as an open question whether one can devise an algorithm with an APR better than 2.

It is not difficult to extend the strong divisibility results of the previous section to the variable-size bin case. If the pair $(L_i, s(B_i))$, where L_i is the sublist of L containing the elements with size not exceeding $s(B_i)$, is strongly divisible for all $i = 1, \dots, k$, then the following algorithm produces an optimal packing (see Coffman, Garey and Johnson [20]). First use the FFD algorithm to pack items in bins of type B_1 , until either all items are packed or the total size of the unpacked items is less than $s(B_1)$. In the latter case, if the size of the largest unpacked item exceeds $s(B_2)$, place all these items in a bin of type B_1 ; otherwise, repeat the process for the remaining items and bins of types B_2, B_3, \dots . The case of weak divisibility has not been treated.

5.4 Dynamic bin packing

In this section we consider a generalization in which each item a_i is characterized by a triple $(b(a_i), d(a_i), s(a_i))$, where $b(a_i)$ is the start (arrival) time, $d(a_i)$ ($d(a_i) > b(a_i)$) is the departure time, and, as usual, $s(a_i)$ is the size. Item a_i remains in the packing during the time interval $[b(a_i), d(a_i))$. We assume that $b(a_i) \leq b(a_j)$ if $i < j$. The problem calls for the minimization of the maximum number, $OPT_D(L)$, of bins ever required in dynamically packing list L when repacking of the current set of items is allowed each time a new item arrives. More formally, if $A(L, t)$ denotes the number of bins used by algorithm A to pack the current set of items at time t , the objective is to minimize

$$A(L) = \max_{0 \leq t \leq b(a_n)} A(L, t)$$

Note that, in this case, an opened bin can later become empty, so the classical open/close terminology is no longer valid. The definition of the APR is straightforward:

$$R_A^\infty = \lim_{n \rightarrow \infty} \sup \left\{ \frac{A(L)}{OPT_D(L)} : |L| = n \right\}$$

The problem models an important aspect of multiprogramming operating systems: the dynamic memory allocation for paged or other virtual memory systems (see, e.g., Coffman [19]), or data storage problems where the bins

correspond to storage units (e.g., disk cylinders or tracks), and the items correspond to records which must be stored for certain specified periods of time (see Coffman, Garey and Johnson [22]).

Research on dynamic packing is at the boundary between bin packing and dynamic storage allocation. The most significant difference between the two areas lies in the repacking assumption of dynamic bin packing; repacking is disallowed in dynamic storage allocation, so fragmentation of the occupied space can occur. Coffman, Garey and Johnson [22] studied two versions of the classical FF algorithm. The first is a direct generalization of the classical algorithm. The second is a variant (*Modified First-Fit*, MFF) in which the elements with $s(a_i) \geq \frac{1}{2}$ are handled separately, in an attempt to pair each of them with smaller elements.

Theorem 5.7 (Coffman, Garey and Johnson [22]). *If $\frac{1}{k+1} < \max\{s(a_i)\} \leq \frac{1}{k}$, then*

$$\frac{11}{4} \leq R_{\text{FF}}^\infty(k) \leq \frac{5}{2} + \frac{3}{2} \ln \frac{\sqrt{13}-1}{2} = 2.89674\dots \quad \text{if } k = 1;$$

$$\frac{k+1}{k} + \frac{1}{k^2} \leq R_{\text{FF}}^\infty(k) \leq \frac{k+1}{k} + \frac{1}{k-1} \ln \frac{k^2}{k^2-k+1} \quad \text{if } k \geq 2;$$

$$2.77 \leq R_{\text{MFF}}^\infty(1) \leq \frac{5}{2} + \ln \frac{4}{3} = 2.78768\dots$$

For $k = 2$, we get $\frac{7}{4} \leq R_{\text{FF}}^\infty(2) \leq 1.78768\dots$. Although these results are worse than their classical counterparts, relative performance is much better than one might think. This can be seen in the following lower bounds.

Theorem 5.8 (Coffman, Garey and Johnson [22]). *For any on-line dynamic bin packing algorithm A , $R_A^\infty \geq \frac{5}{2}$ and $R_A(k) \geq 1 + \frac{k+2}{k(k+1)}$ if $k \geq 2$, which gives $R_A(2) \geq 1.666\dots$*

If we restrict our attention to strongly divisible instances (see Section 5.2), then matters improve, as expected. The gap between the general lower bound and that for FF disappears. Indeed, we have

Theorem 5.9 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible and L is such that $s(a_1) > s(a_2) > \dots > s(a_k)$ ($k \geq 2$), then the APR of the dynamic version of FF is*

$$R_{\text{FF}}^\infty = \prod_{i=1}^{k-1} \left(1 + \frac{s(a_i)}{C} - \frac{s(a_{i+1})}{C} \right)$$

Moreover, for any on-line algorithm A that operates only on strongly divisible instances, $R_A^\infty \geq R_{\text{FF}}^\infty$.

By a straightforward inductive argument, it can be shown that the continued product has the upper bound

$$\lim_{k \rightarrow +\infty} \left(1 + \frac{1}{2^{k-2}}\right) \prod_{i=1}^{k-2} \left(1 + \frac{1}{2^i}\right) = 2.384\dots$$

5.5 Bounds on the number of items per bin

The present section deals with the generalization in which the maximum number of items packed into a bin is bounded by a positive integer p . The problem has practical applications in multiprocessor scheduling with a single resource constraint, when the number p of processors is fixed, or in multitasking operating systems where the number of tasks is bounded. In the context of these applications, Krause, Shen and Schwetman [79] modified classical algorithms so as to deal with the restricted number of items per bin. Their on-line algorithm, pFF, is FF with an additional check on the number of items in open bins. They proved that, if $p \geq 3$, then

$$\frac{27}{10} - \frac{37}{10p} \leq R_{\text{pFF}}^\infty \leq \frac{27}{10} - \frac{24}{10p}.$$

As p tends to ∞ , the bound is much worse than the corresponding APR of the unrestricted problem (2.7 versus 1.7). Whether this upper bound can be improved remains an open question.

Their second (off-line) algorithm (*Largest Memory First*, LMF) is an adaptation of FFD. They proved that $R_{\text{LMF}}^\infty = 2 - \frac{2}{p}$ if $p \geq 2$. Here too the gap between the unrestricted and the restricted case is large (2 versus $\frac{11}{9}$). The authors' search for an algorithm with better worst-case behavior resulted in the *Iterated Worst-Fit Decreasing* (IWFD) algorithm. IWFD starts by sorting the items according to nonincreasing size, and by opening q empty bins, where $q := \lceil \max(\frac{n}{p}, \sum_{j=1}^n s(a_j)) \rceil$ is an obvious lower bound on $OPT(L)$. Then (i) IWFD tries to place the items into these q bins using the Worst-Fit rule (an item is placed into the open bin whose current number of items is less than p and whose current content is minimum, breaking ties by highest bin index); (ii) whenever the current item a_i does not fit in any of the q bins, q is increased by 1, all items are removed from the bins and the processing is restarted from (i). If one implements this approach using a binary search on q , the time complexity of IWFD is $O(n \log^2 n)$. Krause, Shen and Schwetman proved that the algorithm behaves very well in certain special cases:

- if $p = 2$ then $IWFD(L) = OPT(L)$ for any list L ;
- if in the final schedule no capacity constraint is operative, i.e., no open bin has residual capacity smaller than the size of the smallest item, then $IWFD(L) = OPT(L)$;
- if in the final schedule no cardinality constraint is operative, i.e., no open bin containing p items has residual capacity at least equal to the size of the smallest item, then $R_{IWFD}^\infty < \frac{4}{3}$.

For the general case, where the final packing contains bins for which the capacity constraint is operative and bins for which the cardinality constraint is operative, the constants are significantly worse but still small: $\frac{4}{3} \leq R_{IWFD}^\infty \leq 2$. The authors conjecture that $R_{IWFD}^\infty = \frac{4}{3}$.

For a long time it was an open question whether there exists a heuristic algorithm with an APR better than 2. Very recently, Kellerer and Pferschy [75] proposed a new algorithm based on a method that proved to be fruitful in scheduling theory, namely one that performs a binary search on the possible $OPT(L)$ values. At each search step they run a procedure that tries to pack all items into a number of bins equal to $\frac{3}{2}$ times the current search value. The procedure basically tries to solve a multiprocessor scheduling problem using the classical LPT rule with a cardinality constraint. The time complexity of this *Binary Search* (BS) algorithm is $O(n \log^2 n)$, and the improved bound is $\frac{4}{3} \leq R_{BS}^\infty \leq \frac{3}{2}$. Although the gap has been reduced, the exact bound is still not known.

5.6 Minimizing the bin capacity

Suppose we fix the number of bins, and let the objective be a smallest C such that a given list L of items can be packed in m bins of capacity C . This dual of bin packing actually predates bin packing and is known as multiprocessor or parallel-machine scheduling. It is a central problem of scheduling theory, and is surveyed in depth elsewhere (see e.g., Lawler et al. [81] and Hall [60]), so we will only highlight recent results and research directions. The problem is referred to as $P||C_{\max}$ in scheduling notation.

Optimal and heuristic capacities normally differ by at most a maximum item size, so we deal with absolute worst-case ratios rather than asymptotic ones. The first analysis of approximation algorithms was presented by Graham [56, 57], who studied the worst-case performance of various algorithms. For the on-line case, he investigated the Worst-Fit algorithm (better

known as the *List Scheduling*, LS algorithm), which initially opens m empty bins and then iteratively packs the current item into the bin having the minimum current content (breaking ties in favor of the lowest bin index). Graham proved that $R_{\text{LS}} = 2 - \frac{1}{m}$, a bound that for many years was thought to be best among on-line algorithms. Galambos and Woeginger were first to prove that the bound could be improved. Let $A(L, m)$ denote the bin capacity needed under A for m bins, and define

$$R_A(m) = \sup_L \left\{ \frac{A(L, m)}{\text{OPT}(L, m)} \right\} \quad \text{and} \quad R(m) = \inf_A R_A(m)$$

Theorem 5.10 (Galambos and Woeginger [44]). *There exists a sequence of nonnegative numbers $\epsilon_1, \epsilon_2, \dots$, with $\epsilon_m > 0$, $m \geq 4$ and $\epsilon_m \rightarrow 0$ as $m \rightarrow \infty$, such that $R(m) \leq 2 - \frac{1}{m} - \epsilon_m$.*

This result encouraged further research; important milestones on the way to our current position on the problem are as follows. Bartal, Karloff and Rabani [9] presented an algorithm with a worst-case ratio $2 - \frac{1}{70} = 1.98571\dots$ for all $m \geq 70$. Earlier, Faigle, Kern and Turan [35] proved lower bounds for different values of m . They pointed out that for $m \leq 3$ the LS algorithm is optimal among on-line algorithms, and they proved a $1 + \frac{1}{\sqrt{2}} = 1.70710\dots$ lower bound for $m \geq 4$. Then these bounds were improved, as shown below.

Theorem 5.11 (Chen, van Vliet and Woeginger [18]). *For all $m = 80 + 8k$ with k a nonnegative integer, $R(m) \geq 1.83193$.*

Theorem 5.12 (Bartal, Karloff and Rabani [10]). *For all $m \geq 3454$, $R(m) \geq 1.8370$.*

Until recently, the following result gave the best upper bound for $R(m)$, and was a generalization of the methods in [10].

Theorem 5.13 (Karger, Phillips and Torng [73]). *$R(m) \leq 1.945$ for all m .*

The tightest results currently known are those of Albers [2] who recently proved the next two bounds.

Theorem 5.14 (Albers [2]). *$R(m) \leq 1.923$ for all m .*

Just as previous authors, Albers recognized that, during the packing process, a good algorithm must try to avoid packings in which the content of each of the bins is about the same, for in such cases many small items

followed by a large item can create a poor worst-case ratio. Albers' new algorithm attempts to maintain throughout the packing process $\lfloor \frac{m}{2} \rfloor$ bins with small total content, and $\lceil \frac{m}{2} \rceil$ bins with large total content. The precise aim is always to have a total content in the small-content bins that is at most γ times the total content in the large-content bins. With an optimal choice of γ , one obtains a worst-case ratio of at most 1.923.

For her new lower bound, Albers proved

Theorem 5.15 (Albers[2]). $R(m) \geq 1.852$ for all $m \geq 80$.

Attempts to further reduce the general gap of about .07 continue. For the special case $m = 4$, it is proved in [18] that $1.73101 \leq R(4) \leq \frac{52}{30} = 1.7333\dots$, but the exact value of $R(4)$ remains an open problem. Another enticing open problem is: Does $R(m) < R(m + 1)$ hold for any m ?

Let us turn now to the off-line case. By preordering the elements according to nonincreasing size and applying the LS algorithm, we obtain the so-called *Longest Processing Time* (LPT) algorithm for $P||C_{\max}$. Graham [57] proved that $R_{\text{LPT}} = \frac{4}{3} - \frac{1}{3m}$. The *Multifit* algorithm by Coffman, Garey and Johnson [21] adopts a different strategy, leading to better worst-case behavior. The algorithm determines, by binary search over an interval with crude but easily established lower and upper bounds, the smallest capacity C such that the FFD algorithm can pack all the elements into m bins. Coffman, Garey and Johnson proved that, if k binary search iterations are performed, then the algorithm, denoted by MF_k , requires $O(n \log n + kn \log m)$ time and has an absolute worst-case ratio satisfying

$$R_{\text{MF}_k} \leq 1.22 + 2^{-k}$$

Friesen [38] improved the bound uniform in k to 1.2, but Yue [103] later settled the question by proving that this bound is $\frac{13}{11}$. Friesen and Langston [39] developed a different version of the Multifit algorithm, proving that its worst-case ratio is bounded by $\frac{72}{61} + 2^{-k}$.

A different off-line approach was proposed by Graham [57]. His algorithm Z_k optimally packs the $\min\{k, n\}$ largest elements and completes the solution by packing the remaining elements, if any, according to the LS algorithm. Graham proved that

$$R_{Z_k} \leq 1 + \frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{k}{m} \right\rfloor}$$

and that the bound is tight when m divides k . Algorithm Z_k implicitly defines an approximation scheme. By selecting $k = k_\varepsilon = \lceil \frac{m(1-\varepsilon)-1}{\varepsilon} \rceil$, we obtain $R_{Z_{k_\varepsilon}} \leq 1 + \varepsilon$. The running time is $O(n \log n + m^{m(1-\varepsilon)/\varepsilon})$ and so the method is unlikely to be practical. The result was improved by Sahni [93], in the sense that the functional dependence on m and ε was reduced substantially. His algorithm A_ε has running time $O(n(\frac{n^2}{\varepsilon})^{m-1})$ and satisfies $R_{A_\varepsilon} \leq 1 + \varepsilon$; hence it is a fully polynomial-time approximation scheme for any fixed m .

For m even moderately large, the approach of Hochbaum and Shmoys [61] may offer major improvements. They developed an ε -approximate algorithm that removed the running-time dependence on m and reduced the dependence on n . The dependence on $\frac{1}{\varepsilon}$ is exponential, which is to be expected, since a fully polynomial-time approximation scheme would imply $P = NP$. Their algorithm (like MF_k , for example) is based on the binary search of an interval, but it uses the ε -dual approximation algorithm of Section 5.3 at each iteration. Hochbaum and Shmoys show that, after k steps of the binary search, the bin capacity is at most $(1 + \varepsilon)(1 + 2^{-k})$ times optimal. From this fact and the properties of M_ε given in Section 5.3, one obtains a linear-time approximation scheme for the capacity minimization problem. (See also Hochbaum's [63] more extensive discussion of this interesting technique.)

5.7 Maximizing the number of items packed

In this section, we consider another variant in which the number of bins is fixed; the objective now is to pack a maximum cardinality subset of a given list $L_{(n)}$. The problem arises in computing applications, when we want to maximize the number of records stored in one or more levels of a memory hierarchy, or to maximize the number of tasks performed on multiple processors within a given time interval.

We extend the classical bin packing notation in the obvious way and define

$$\hat{R}_A(m, n) = \min \left\{ \frac{A(L_{(n)}, m)}{OPT(L_{(n)}, m)} \right\}$$

so that the APR is defined by

$$\hat{R}_A^\infty(m) = \liminf_{n \rightarrow \infty} \hat{R}_A(m, n)$$

and has values less than or equal to 1.

The problem was first studied by Coffman, Leung and Ting [25], who adapted the *First-Fit Increasing* (FFI) heuristic. The items are preordered according to nondecreasing size, the m bins are opened, and the current item is packed into the lowest indexed bin into which it fits, stopping the process as soon as an item is encountered that does not fit into any bin. It is proved in [25] that $\widehat{R}_{\text{FFI}}^\infty(m) = \frac{3}{4}$ for all m .

The *Iterated First-Fit Decreasing* (IFFD) algorithm was investigated by Coffman and Leung [24]. The algorithm sorts the items by nonincreasing size, and iteratively tries to pack all the items into the m bins following the First-Fit rule. Whenever the current item cannot be packed, the process is stopped, the largest item is removed from the list, and a new FFD iteration is performed on the shortened list. The search terminates as soon as FFD is able to pack all the items of the current list. Coffman and Leung showed that IFFD performs at least as well as FFI on every list, and that $\frac{6}{7} \leq \widehat{R}_{\text{IFFD}}^\infty(m) \leq \frac{7}{8}$. The time complexity of IFFD is $O(n \log n + mn \log m)$, but a tight APR is not known.

Let us now consider lower bounds on the APR. If an algorithm A packs items a_1, \dots, a_t from a list $L = (a_1, \dots, a_n)$ so that $s(a_j) > 1 - s(B_i)$ for any i and for any $j > t$ (i.e., no unpacked item fits into any bin), then it is said to be a *prefix algorithm*. Note that both FFI and IFFD are prefix algorithms).

Theorem 5.16 (Coffman, Leung and Ting [25]). *Let A be a prefix algorithm and let $k = \min_{1 \leq i \leq m} \{|B_i|\}$ be the least number of items packed into any bin. Then*

$$\widehat{R}_A^\infty(m) \geq \frac{mk}{mk + m - 1}$$

Moreover, the bound is achievable for all $m \geq 1$ and $k \geq 1$.

The case of divisible item sizes was investigated by Coffman, Garey and Johnson [20]. They proved that both FFI and IFFD produce optimal packings if (L, C) is strongly divisible. IFFD remains optimal even if (L, C) is weakly divisible, but FFI does not.

The case of variable sized bins was analyzed by Langston [80], who proved that, if the bins are rearranged by nonincreasing size, then $\widehat{R}_{\text{FFI}}^\infty(m) = \frac{1}{2}$ and $\frac{2}{3} \leq \widehat{R}_{\text{IFFD}}^\infty(m) \leq \frac{8}{11}$ for all m .

5.8 Maximizing the number of full bins

In this variant of the problem the objective is to pack a list of items into a maximum number of bins subject to the constraint that the content of each bin be no less than a given threshold T . Each such bin is said to be full. Potential practical applications are (i) the packing of canned goods so that each can contains at least its advertised net weight, and (ii) the stimulation of economic activity during a recession by allocating tasks to a maximum number of factories, all working at or beyond the minimal feasible level.

This problem is yet another dual of bin packing. A comprehensive treatment can be found in Assmann's Ph.D. thesis [4]; Assmann's collaboration with Johnson, Kleitman and Leung [5] established the main results. They first analyzed an on-line algorithm, a dual version of NF (*Dual Next-Fit*, DNF): pack the elements into the current bin B_j until $s(B_j) \geq T$, then open a new empty bin B_{j+1} as the current bin. If the current bin is not full when all items have been packed, then merge its contents with those of other full bins. All on-line algorithms are allowed this last repacking step to ensure that all bins are full. It is proved in [5] that $\tilde{R}_{\text{DNF}}^{\infty} = \frac{1}{2}$, where

$$\tilde{R}_A^{\infty} = \liminf_{m \rightarrow \infty} \left(\min \left\{ \frac{A(L)}{OPT(L)} : OPT(L) = m \right\} \right)$$

Assmann et al. [5] also studied a parametrized dual of FF, called *Dual First-Fit* (DFF[r]). Given a parameter r ($1 < r < 2$), the current item a_i is packed into the first bin B_j for which $c(B_j) + s(a_i) \leq rT$. If there are at least two nonempty and unfilled bins (i.e., bins B_j with $0 < c(B_j) < T$) when all items have been packed, an item is removed from the rightmost such bin and added to the leftmost, thus filling it. Finally, if a single nonempty and unfilled bin remains, then its contents are merged with those of previously packed bins. It was shown that $\tilde{R}_{\text{DFF}[r]}^{\infty} = \frac{1}{2}$, although a better bound might have been expected. But some years later Csirik and Totik proved that DNF is optimal among the on-line algorithms.

Theorem 5.17 (Assmann et al. [5] and Csirik and Totik [32]). *We have that*

$$\tilde{R}_{\text{DFF}[r]}^{\infty} = \frac{1}{2}$$

and that there is no on-line dual bin packing algorithm A for which $\tilde{R}_A^{\infty} > \frac{1}{2}$

Turning now to off-line algorithms, we mention first the observation of Assmann et al. [5] that presorting does not help the adaptations of algorithms Next-Fit Decreasing and Next-Fit Increasing, for which $\tilde{R}_{\text{NFD}}^{\infty} =$

$\tilde{R}_{\text{NFI}}^{\infty} = \frac{1}{2}$. On the other hand, the off-line version DFFD[r] of DFF[r], obtained by presorting the items according to nonincreasing size, has better performance.

Theorem 5.18 (Assmann et al. [5]). $\tilde{R}_{\text{DFFD}[r]}^{\infty} = \frac{2}{3}$ if $\frac{4}{3} < r < \frac{3}{2}$ and $\lim_{r \rightarrow 1} \tilde{R}_{\text{DFFD}[r]}^{\infty} = \lim_{r \rightarrow 2} \tilde{R}_{\text{DFFD}[r]}^{\infty} = \frac{1}{2}$

The *Iterated Lowest-Fit Decreasing* (ILFD) algorithm was also investigated in [5]. It is analogous to algorithm IFFD described in Section 5.7. The items are preordered by nonincreasing size. At each iteration, a pre-fixed number m of bins is considered, and a Lowest-Fit packing is obtained by iteratively assigning the current item to the bin with minimum contents. Binary search on m determines the maximum value for which all m bins are filled. Since n is an obvious upper bound on the optimal solution value, it is easily seen that the algorithm has $O(n \log^2 n)$ time complexity. Moreover,

Theorem 5.19 (Assmann et al. [5]). $\tilde{R}_{\text{ILFD}}^{\infty} = \frac{3}{4}$.

It is not difficult to see that DNF does not produce optimal packings even when (L, C) is strongly divisible. However, we do have the following optimality results.

Theorem 5.20 (Coffman, Garey and Johnson [20]). *If (L, C) is strongly divisible, then the dual version of NFD (DNFD) produces an optimal packing. For weakly divisible lists DNFD is no longer optimal, but ILFD is.*

Finally, we observe that obtaining an approximation scheme for the dual bin packing problem is still a research challenge. The approach used by Fernandez de la Vega and Lueker [36] and Karmarkar and Karp [74] (see Section 4.3), which eliminates the effect of small items on worst-case behavior, does not appear applicable, as in this dual problem, small items can play an important role in filling small gaps.

5.9 Further directions

5.9.1 Partial orders

In this generalization, precedence constraints among the elements are given, where precedence refers to the relative ordering of bins. Let \prec denote the partial order giving the precedence constraints. Then $a_i \prec a_j$ means that, if

a_i and a_j are packed in B_r and B_s , respectively, then $r \leq s$. Call the model *strict* if $r < s$ replaces $r \leq s$ in this definition.

Two practical applications have been considered in the literature. The first one is the assembly line balancing problem, in which the assembly line consists of identical workstations (the bins) where the products stop for a period of time equal to the bin capacity. The item sizes are the durations of tasks to be performed, and a partial order is imposed: $a_i \prec a_j$ means that the workstation to which a_i is assigned cannot be downstream of the one to which a_j is assigned. The second application arises in multiprocessing scheduling; here each item corresponds to a unit-duration process having a memory (or other resource) requirement equal to the item size. The bin capacity measures the total memory availability. In the given partial order, $a_i \prec a_j$ imposes the requirement that a_i must be executed before a_j finishes. The objective is then to find a feasible schedule that finishes the set of processes in minimum time (number of bins).

The first problem was studied by Wee and Magazine [99] and the second by Garey et al. [48]. In both cases the *Ordered First-Fit Decreasing* (OFFD) algorithm was applied. An item is called *available* if all its immediate predecessors have already been packed. At each stage, the set of currently available items is sorted according to nonincreasing size, and each item is packed into the lowest indexed bin where it fits and no precedence constraint is violated. Note that, if no partial order is given, this algorithm produces the same packing as FFD. In general, however, its worst-case behavior is considerably worse. The APR is $R_{\text{OFFD}}^{\infty} = 2$, except in the strict model, where $R_{\text{OFFD}}^{\infty} = \frac{27}{10}$.

5.9.2 Clustered Items

In this generalization, a function $d_{ij} = f(a_i, a_j)$ is given, measuring the ‘distance’ between items a_i and a_j). A distance constraint D is also given; two items a_i and a_j may be packed into the same bin only if $d_{ij} \leq D$. The problem has several obvious practical applications in contexts where geographical location constraints are present. Chandra, Hirschler and Wong [16] studied different cases, their main result being for the case where the items in a bin must all reside within the same unit square. They proposed a geometric algorithm A and proved that $3.75 \leq R_A^{\infty} \leq 3.8$.

5.9.3 Item types

In studying the packing of advertisements on Web pages, Adler, Gibbons and Matias [1] encountered a variant of bin packing in which items are classified into *types*; the set of types is given and there are no constraints on the number of items of any type. The problem is to pack the items into as few bins as possible subject to the constraint that no bin can have two items of the same type. They design optimal algorithms for restricted cases and then bound the performance of these algorithms viewed as approximations to the general problem.

5.9.4 Packing sets and graphs

Let the bin capacity C be a positive integer and suppose that items are sets of elements drawn from some universal set. A collection of items can fit into a bin if and only if their union has at most C elements. As usual, the object is to pack the items in a smallest number of bins. Good approximation algorithms have yet to be analyzed for this problem; adaptations of the classical bin packing approximation algorithms do not have finite APR's. One observes immediately that while classical algorithms prioritize items based on cardinality and position in sequence, a good algorithm for set packing will have to consider the intersections among a given set of items, an orthogonal basis for prioritizing the items. It is thought that approximation algorithms with finite APR's for this problem are unlikely to exist, but a rigorous treatment of this question has yet to appear.

M. Dror [34] points out that scheduling set-ups in a manufacturing process is an application of set packing. As an example, he describes a machine that produces many types of circuit boards, each requiring a given set of components. The machine has a magazine (bin) that holds at most C components. At any time, only the circuit boards with their component sets (items) in the magazine can be in production. The problem is to plan the overall production process so as to minimize the number of set-ups.

S. Khanna [77] mentions, in connection with studies of multimedia communications, that graph packing is an interesting special case. Items are edges (pairs of vertices) in a given graph G . An edge is packed in a bin if both of the vertices to which it is incident are in the bin, and so the problem is to pack the edges of G into as few bins as possible subject to the constraint that there can be at most C vertices in any bin. The approximability of this problem has yet to be studied.

References

- [1] M. Adler, P. B. Gibbons, and Y. Matias. Scheduling space sharing for internet advertising. Technical report, Bell Labs, Lucent Technologies, Murray Hill, NJ 07974, 1997.
- [2] S. Albers. Better bounds for on-line scheduling. In *Proc. 29th Annual ACM Symp. Theory of Comput.*, pages 130–139, 1997.
- [3] R. J. Anderson, E. W. Mayr, and M. K. Warmuth. Parallel approximation algorithms for bin packing. *Inf. and Comput.*, 82:262–277, 1989.
- [4] S. F. Assmann. *Problems in Discrete Applied Mathematics*. PhD thesis, Mathematics Department MIT, Cambridge, MA, 1983.
- [5] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J. Y.-T. Leung. On a dual version of the one-dimensional bin packing problem. *J. Algorithms*, 5:502–525, 1984.
- [6] B. S. Baker. A new proof for the first-fit decreasing bin-packing algorithm. *J. Algorithms*, 6:49–70, 1985.
- [7] B. S. Baker, D. J. Brown, and H. P. Katseff. A $5/4$ algorithm for two-dimensional packing. *J. Algorithms*, 2:348–368, 1981.
- [8] B. S. Baker and E. G. Coffman, Jr. A tight asymptotic bound for next-fit-decreasing bin-packing. *SIAM J. Algebraic Discr. Meth.*, 2(2):147–152, 1981.
- [9] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th Annual ACM Symp. Theory of Comput.*, pages 51–58, Victoria, Canada, 1992.
- [10] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Inform. Process. Lett.*, 50:113–116, 1994.
- [11] J. Békési, G. Galambos, and H. Kellerer. A $5/4$ linear time bin packing algorithm. Technical Report OR-97-2, Teachers Trainer College, Szeged, Hungary, 1997.
- [12] J. Blazewicz and K. Ecker. A linear time algorithm for restricted bin packing and scheduling problems. *Oper. Res. Lett.*, 2:80–83, 1983.

- [13] D. J. Brown. A lower bound for on-line one-dimensional bin-packing algorithms. Technical Report R-864, University of Illinois, Coordinated sc. lab., Urbana, 1979.
- [14] R. E. Burkard and G. Zhang. Bounded space on-line variable-sized bin packing. *Acta Cybern.*, 13:63–76, 1997.
- [15] L. M. A. Chan, D. Simchi-Levi, and J. Bramel. Worst-case analyses, linear programming, and the bin-packing problem. Unpublished manuscript, 1994.
- [16] A. K. Chandra, D. S. Hirschler, and C. K. Wong. Bin packing with geometric constraints in computer network design. *Oper. Res.*, 26:760–772, 1978.
- [17] B. Chandra. Does randomization help in on-line bin packing? *Inform. Process. Lett.*, 43:15–19, 1992.
- [18] B. Chen, A. van Vliet, and G.J. Woeginger. New lower and upper bounds for on-line scheduling. *Oper. Res. Lett.*, 16:221–230, 1994.
- [19] E. G. Coffman, Jr. An introduction to combinatorial models of dynamic storage allocation. *SIAM Rev.*, 25(3):311–325, 1983.
- [20] E. G. Coffman, Jr., M. Garey, and D. S. Johnson. Bin packing with divisible item sizes. *J. Complexity*, 3:405–428, 1987.
- [21] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.*, 7(1):1–17, 1978.
- [22] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing: An updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49–106. Springer Verlag, Wien, 1984.
- [23] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 46–93. PWS Publ. Company, 1997.
- [24] E. G. Coffman, Jr. and J. Y.-T. Leung. Combinatorial analysis of an efficient algorithm for processor and storage allocation. *SIAM J. Comput.*, 8(2):202–217, 1979.

- [25] E. G. Coffman, Jr., J. Y.-T. Leung, and D. W. Ting. Bin packing: Maximizing the number of pieces packed. *Acta Inform.*, 9:263–271, 1978.
- [26] E. G. Coffman, Jr. and G. S. Lueker. *Probabilistic analysis of packing and partitioning algorithms*. John Wiley & Sons, New York, 1991.
- [27] J. Csirik. An on-line algorithm for variable-sized bin packing. *Acta Inform.*, 26:697–709, 1989.
- [28] J. Csirik. The parametric behaviour of the first fit decreasing bin-packing algorithm. *J. Algorithms*, 15:1–28, 1993.
- [29] J. Csirik, G. Galambos, and G. Turan. Some results on bin-packing. In *Proc. EURO VI Conf.*, page 52, Vienna, Austria, 1983.
- [30] J. Csirik and B. Imreh. On the worst-case performance of the next-k-fit bin-packing heuristic. *Acta Cybern.*, 9:89–105, 1989.
- [31] J. Csirik and D. S. Johnson. Bounded space on-line bin-packing: best is better than first. In *Proc. 2nd Annual ACM-SIAM Symp. Discr. Algorithms*, pages 309–319, Philadelphia, 1991.
- [32] J. Csirik and V. Totik. Online algorithms for a dual version of bin paking. *Discr. Appl. Math.*, 21:163–167, 1988.
- [33] J. Csirik and G. J. Woeginger. Online packing and covering problems. Technical Report No. 83, T.U. Graz (Austria), 1996.
- [34] M. Dror. Private communication.
- [35] U. Faigle, W. Kern, and G. Turan. On the performance of on-line algorithms for particular problems. *Acta Cybern.*, 9:107–119, 1989.
- [36] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
- [37] D. C. Fisher. Next-fit packs a list and its reverse into the same number of bins. *Oper. Res. Lett.*, 7(6):291–293, 1988.
- [38] D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM J. Comput.*, 13(1):170–181, 1984.

- [39] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM J. Comput.*, 15(1):222–230, 1986.
- [40] D. K. Friesen and M. A. Langston. Analysis of a compound bin packing algorithm. *SIAM J. Discr. Math.*, 4(1):61–79, 1991.
- [41] G. Galambos. A new heuristic for the classical bin-packing problem. Technical Report 82, Institute fuer Mathematik, Augsburg, 1985.
- [42] G. Galambos. Parametric lower bound for on-line bin-packing. *SIAM J. Algebraic Discr. Meth.*, 7(3):362–367, 1986.
- [43] G. Galambos and J. B. G. Frenk. A simple proof of Liang’s lower bound for on-line bin packing and the extension to the parametric case. *Discr. Appl. Math.*, 41:173–178, 1993.
- [44] G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham’s list scheduling. *SIAM J. Comput.*, 22(2):345–355, 1993.
- [45] G. Galambos and G. J. Woeginger. Repacking helps in bounded space on-line bin-packing. *Computing*, 49:329–338, 1993.
- [46] G. Galambos and G. J. Woeginger. On-line bin packing – a restricted survey. *Z. Oper. Res.*, 42:25–45, 1995.
- [47] G. Gambosi, A. Postiglione, and M. Talamo. New algorithms for on-line bin packing. In R. Petreschi, G. Ausiello, and D. P. Bovet, editors, *Algorithms and Complexity*, pages 44–59. World Scientific, Singapore, 1990.
- [48] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao. Resource constrained scheduling as generalized bin packing. *J. Combin. Theory, Ser. A*, 21:257–298, 1976.
- [49] M. R. Garey, R. L. Graham, and J. D. Ullmann. Worst-case analysis of memory allocation algorithms. In *Proc. 4th Annual ACM Symp. Theory of Comput.*, pages 143–150, New York, 1972.
- [50] M. R. Garey and D. S. Johnson. *Computers and intractability (A Guide to the theory of NP-Completeness)*. W. H. Freeman and Company, San Francisco, 1979.

- [51] M. R. Garey and D. S. Johnson. Approximation algorithm for bin-packing problems: a survey. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithm in Combinatorial Optimization*, pages 147–172. Springer Verlag, New York, 1981.
- [52] M. R. Garey and D. S. Johnson. A $71/60$ theorem for bin packing. *J. Complexity*, 1:65–106, 1985.
- [53] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Oper. Res.*, 9:849–859, 1961.
- [54] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem – (Part II). *Oper. Res.*, 11:863–888, 1963.
- [55] S. W. Golomb. On certain nonlinear recurring sequences. *American Math. Monthly*, 70:403–405, 1963.
- [56] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Syst. Tech. J.*, 45(45):1563–1581, 1966.
- [57] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):263–269, 1969.
- [58] R. L. Graham. Bounds on multiprocessing anomalies and related packing algorithms. In *Proc. 1972 Spring Joint Computer Conf.*, pages 205–217, Montvale NJ, 1972. AFIPS Press.
- [59] E. F. Grove. Online bin packing with lookahead. Unpublished manuscript, 1994.
- [60] L. A. Hall. Approximation algorithms for scheduling. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 1–45. PWS Publ. Company, 1997.
- [61] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM*, 34:144–162, 1987.
- [62] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publ. Company, Boston, 1997.
- [63] D. S. Hochbaum. Various notions of approximation: Good, better, best, and more. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 389–391. PWS Publ. Company, 1997.

- [64] M. Hofri. *Analysis of Algorithms*. Oxford University Press, New York, 1995.
- [65] Z. Ivković and E. Lloyd. Fully dynamic algorithms for bin packing: being myopic helps. In *Proc. 1st European Symp. on Algorithms*, volume 726 of *Lecture Notes in Computer Science*, pages 224–235. Springer Verlag, New York, 1993.
- [66] Z. Ivković and E. Lloyd. A fundamental restriction on fully dynamic maintenance of bin packing. *Inf. Proc. Lett.*, 59(4):229–232, 1996.
- [67] Z. Ivković and E. Lloyd. Partially dynamic bin packing can be solved within $1 + \epsilon$ in (amortized) polylogarithmic time. *Inf. Proc. Lett.*, 63(1):45–50, 1997.
- [68] D. S. Johnson. Fast allocation algorithms. In *Proc. 13th IEEE Symp. Switching and Automata Theory*, pages 144–154, New York, 1972.
- [69] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.
- [70] D. S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8:272–314, 1974.
- [71] D. S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 3:89–99, 1982.
- [72] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974.
- [73] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20:400–430, 1996.
- [74] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. 23rd Annual IEEE Symp. Found. Comput. Sci.*, pages 312–320, 1982.
- [75] H. Kellerer and U. Pferschy. An on-line alorithm for cardinality constrained bin packing problem. Technical report, Universitaet Graz und TU Graz, 1997.

- [76] C. Kenyon. Best-fit bin-packing with random order. In *Proc. 7th Annual ACM-SIAM Symp. Discr. Algorithms*, pages 359–364, Philadelphia, 1996.
- [77] S. Khanna. Private communication.
- [78] N. G. Kinnersley and M. A. Langston. On-line variable sized bin packing. *Discr. Appl. Math.*, 22:143–148, 1988.
- [79] K. L. Krause, Y. Y. Shen, and H. D. Schwetman. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. *J. ACM*, 22(4):522–550, 1975.
- [80] M. A. Langston. Improved 0/1 interchanged scheduling. *BIT*, 22:282–290, 1982.
- [81] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Logistics of production and inventory*, volume 4 of *Handbooks in operations research and management science*, pages 445–522. North-Holland, Amsterdam, 1993.
- [82] C. C. Lee and D. T. Lee. A new algorithm for on-line bin-packing. Technical Report 83-03-FC-02, Department of Electrical Engineering and computer Science Northwestern University, Evanston, IL, 1983.
- [83] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985.
- [84] H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
- [85] F. M. Liang. A lower bound for on-line bin packing. *Inform. Process. Lett.*, 10(2):76–79, 1980.
- [86] W. Mao. Best-k-fit bin packing. *Computing*, 50:265–270, 1993.
- [87] W. Mao. Tight worst-case performance bounds for next-k-fit bin packing. *SIAM J. Comput.*, 22(1):46–56, 1993.
- [88] C. U. Martel. A linear time bin-packing algorithm. *Oper. Res. Lett.*, 4(4):189–192, 1985.

- [89] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM J. Comput.*, 16(1):149–161, 1987.
- [90] F. D. Murgolo. Anomalous behaviour in bin packing algorithms. *Discr. Appl. Math.*, 21:229–243, 1988.
- [91] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. On-line bin packing in linear time. *J. Algorithms*, 10:305–326, 1989.
- [92] M. B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discr. Appl. Math.*, 34:203–227, 1991.
- [93] S. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23:116–127, 1976.
- [94] H. E. Salzer. The approximation of number as sums of reciprocals. *American Math. Monthly*, 54:135–142, 1947.
- [95] D. Simchi-Levi. New worst-case results for the bin packing problem. *Naval Res. Log. Quart.*, 41:579–585, 1994.
- [96] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.
- [97] A. van Vliet. *Lower and Upper Bounds for On-line Bin Packing and Scheduling Heuristic*. PhD thesis, Erasmus University, Rotterdam, 1995.
- [98] A. van Vliet. On the asymptotic worst case behavoir of harmonic fit. *J. Algorithms*, 20:113–136, 1996.
- [99] T. S. Wee and M. J. Magazine. Assembly line balancing as generalized bin packing. *Oper. Res. Lett.*, 1(2):56–58, 1982.
- [100] G. J. Woeginger. Improved space for bounded-space on-line bin-packing. *SIAM J. Discr. Math.*, 6:575–581, 1993.
- [101] K. Xu. *A bin-packing problem with Item Sizes in the Interval $(0, \alpha]$ for $\alpha \leq \frac{1}{2}$* . PhD thesis, Chinese Academy of Sciences, Institute of Applied Mathematics, Beijing, China, 1993.
- [102] A. C.-C. Yao. New algorithms for bin packing. *J. ACM*, 27(2):207–227, 1980.

- [103] M. Yue. On the exact upper bound for the multifit processor scheduling algorithm. *Ann. Oper. Res.*, 24:233–259, 1991.
- [104] M. Yue. A simple proof of the inequality $FFD(L) \leq \frac{11}{9}OPT(L) + 1 \forall L$ for the FFD bin packing algorithm. *Acta Math. App. Sinica*, 7(4):321–331, 1991.
- [105] G. Zhang. Tight worst-case performance bound for AFB_k . Technical Report 15, Inst. of Applied Mathematics. Academia Sinica, Beijing, China, 1994.
- [106] G. Zhang. Worst-case analysis of the FFH algorithm for on-line variable-sized bin packing. *Computing*, 56:165–172, 1996.
- [107] G. Zhang. A new version of on-line variable-sized bin packing. *Discr. Appl. Math.*, 72:193–197, 1997.

Feedback Set Problems

Paola Festa

*Mathematics and Computer Science Department
University of Salerno, 84081 Baronissi (SA), ITALY
E-mail: paofes@dsab.dia.unisa.it*

Panos M. Pardalos

*Center for Applied Optimization
Department of Industrial and Systems Engineering
University of Florida, Gainesville, FL 32611 USA
E-mail: pardalos@ufl.edu*

Mauricio G.C. Resende

*Information Sciences Research Center
AT&T Labs Research, Shannon Laboratory
Florham Park, NJ 07932 USA
E-mail: mgcr@research.att.com*

Contents

1	Introduction	209
2	Notation and graph representation	210
3	The feedback vertex set problem	211
3.1	Mathematical model of the feedback vertex set problem	212
3.2	Polynomially solvable cases	213
3.2.1	The feedback vertex set problem on chordal, permutation, and interval graphs	215
3.2.2	The feedback vertex set problem on cocomparability graphs .	216
3.2.3	The feedback vertex set problem on meshes and butterflies .	217

3.2.4	The feedback vertex set problem on cube connected cycle networks	219
3.2.5	The feedback vertex set problem on convex bipartite graphs .	220
3.3	Approximation algorithms and provable bounds	220
3.3.1	Undirected graphs	222
3.3.2	Directed graphs	231
3.4	Exact algorithms	233
3.5	Practical heuristics	236
4	The feedback arc set problem	238
4.1	Mathematical model of the feedback arc set problem	238
4.2	Relation between the feedback vertex set and the feedback arc set problems	239
4.3	State of the art of feedback arc set problems	240
5	Applications	243
6	Future directions	247
References		

1 Introduction

Not long ago, there appeared to be a consensus in the literature that feedback set problems, which originated from the area of combinational circuit design, were the least understood among all the classical combinatorial optimization problems due to the lack of positive results in efficient exact and approximating algorithms. This picture has been totally changed in recent years. Dramatic progress has occurred in developing approximation algorithms with provable performance; new bounds have been established one after the other and it is probably fair to say that feedback set problems are becoming among the most exciting frontend problems in combinatorial optimization.

The most general feedback set problem consists in finding a minimum-weight (or minimum cardinality) set of vertices (arcs) that meets all cycles in a collection C of cycles in a graph (G, w) , where w is a nonnegative function defined on the set of vertices $V(G)$ (on the set of edges $E(G)$). This kind of problem is also known as the *hitting cycle problem*, since one must hit every cycle in C . It generalizes a number of problems, including the

minimum feedback vertex (arc) set problem in both directed and undirected graphs, the *subset minimum feedback vertex (arc) set problem* and the *graph bipartization problem*, in which one must remove a minimum-weight set of vertices so that the remaining graph is bipartite. In fact, if C is the set of all cycles in G , then the hitting cycle problem is equivalent to the problem of finding the minimum feedback vertex (arc) set in a graph. If we are given a set of *special* vertices and C is the set of all cycles of an undirected graph G that contains some special vertex, then we have the *subset feedback vertex (arc) set problem* and, finally, if C contains all odd cycles of G , then we have the *graph bipartization problem*. All these problems are also special cases of *vertex (arc) deletion problems*, where one seeks a minimum-weight (or minimum cardinality) set of vertices (arcs) whose deletion gives a graph satisfying a given property.

There are different versions of *feedback set problems*, depending on whether the graph is directed or undirected and/or the vertices (arcs) are weighted or unweighted. Yannakakis [88] has given a general NP-hardness proof for almost all vertex and arc deletion problems restricted to planar graphs. These results apply to the planar bipartization problem, the planar (directed, undirected, or subset) feedback vertex set problems, already proved to be NP-hard [45, 32]. Furthermore, it is NP-complete for planar graphs with no indegree or outdegree exceeding three [45], general graphs with no indegree or outdegree exceeding two [45], and edge-directed graphs [45].

2 Notation and graph representation

Throughout this chapter, we use the following notation and definitions.

A *graph* $G = (V, E)$ consists of a finite set of vertices $V(G)$, and a set of arcs $E(G) \subseteq V(G) \times V(G)$. An *arc* (or edge) $e = (v_1, v_2)$ of a directed graph (digraph) $G = (V, E)$ is an *incoming* arc to v_2 and an *outgoing* arc from v_1 and it is *incident* to both v_1 and v_2 .

If G is undirected, then e is said only incident to v_1 and v_2 .

For each vertex $i \in V(G)$, let $\text{in}(i)$ and $\text{out}(i)$ denote the set of *incoming* and *outgoing* edges of i , respectively. They are defined only in case of a digraph G . If G is undirected, we will take into account only the *degree* $\Delta_G(i)$ of i as the number of edges that are incident to i in G . $\Delta(G)$ denotes the maximum degree among all vertices of a graph G and it is called the *graph degree*. A vertex $v \in G$ is called an *endpoint* if it has degree one, a *linkpoint* if it has degree two, while a vertex having degree higher than two

is called a *branchpoint*.

A *path* P in G connecting vertex u to vertex v is a sequence of arcs e_1, \dots, e_r in $E(G)$, such that $e_i = (v_i, v_{i+1})$, $i = 1, \dots, r$ with $v_1 = u$ and $v_{r+1} = v$. A *cycle* C in G is a path $C = (v_1, \dots, v_r)$, with $v_1 = v_r$.

A *subgraph* $G' = (V', E')$ of $G = (V, E)$ induced by V' is a graph such that $E' = E \cap (V' \times V')$. A graph G is said to be a *singleton*, if $|V(G)| = 1$. Any graph G can be partitioned into isolated connected components G_1, G_2, \dots, G_k and the partition is unique. Similarly, every feedback vertex set V' of G can be partitioned into feedback vertex sets F_1, F_2, \dots, F_k such that F_i is a feedback vertex set of G_i . Therefore, following the additive property and denoting by $\mu(G, w)$ the weight of a minimum feedback vertex (arc) set for (G, w) , we have:

$$\mu(G, w) = \sum_{i=1}^k \mu(G_i, w).$$

3 The feedback vertex set problem

Most of the known results on vertex deletion problems deal with the feedback vertex set problem, that can be easily understood by the following deadlock prevention example in computer systems. Consider an operating system which schedules different processes with requests on different resources, which they need to use exclusively before being released by the process. A directed graph modeling these resource requirements has a node i for each process i with directed arc $e(i, j)$ implying that process i requests a resource already allocated to process j . Therefore, if there is a directed cycle in such a graph, a deadlock occurs and every process in the cycle will wait for the requested resource and will never release the resources already allocated to it. To break such cycles, one can remove some processes from the graph and put them in a waiting queue. It is clear that we want to minimize the number of processes removed.

Formally, the feedback vertex set problem can be described as follows. Let $G = (V, E)$ be a graph and let $w : V(G) \rightarrow R^+$ be a weight function defined on the vertices of G . A *feedback vertex set* of G is a subset of vertices $V' \subseteq V(G)$ such that each cycle in G contains at least one vertex in V' . In other words, a feedback vertex set V' is a set of vertices of G such that by removing V' from G along with all the edges incident to V' , results in a forest. The weight of a feedback vertex set is the sum of the weights of its vertices, and a *minimum feedback vertex set* of a *weighted graph* (G, w) is

a feedback vertex set of G of minimum weight. The weight of a minimum feedback vertex set will be denoted by $\mu(G, w)$. The *minimum weighted feedback vertex set problem* (MWFVS) is to find a minimum feedback vertex set of a given weighted graph (G, w) . The special case of identical weights is called the *unweighted feedback vertex set problem* (UFVS).

3.1 Mathematical model of the feedback vertex set problem

The feedback vertex set problem is a specific type of *set covering problem*, as the objective is to cover all cycles with a minimum cost collection of vertices. A simple polynomial reduction procedure from the vertex cover problem to the feedback vertex set problem is described in [2]. As a covering-type problem, it admits an integer zero-one programming formulation. Given a feedback vertex set V' for a graph (G, w) , $G = (V, E)$, and a set of weights $w = \{w(v)\}_{v \in V(G)}$, let $x = \{x_v\}_{v \in V(G)}$ be a binary vector such that $x_v = 1$ if $v \in V'$, and $x_v = 0$ otherwise. Let C be the set of cycles in (G, w) . The problem of finding the minimum feedback vertex set of G can be formulated as an integer programming problem as follows:

$$(MFVS) \quad \left\{ \begin{array}{l} \min \sum_{v \in V(G)} w(v) x_v \\ \text{s.t. } \sum_{v \in V(\Gamma)} x_v \geq 1 \quad \forall \Gamma \in C \\ 0 \leq x_v \leq 1 \text{ integer, } v \in V(G). \end{array} \right.$$

If one denotes by C_v the set of cycles passing through vertex $v \in V(G)$, then the dual of the linear programming relaxation of (MFVS), is a *packing problem*:

$$(DMFVS) \quad \left\{ \begin{array}{l} \max \sum_{\Gamma \in C} y_\Gamma \\ \text{s.t. } \sum_{\Gamma \in C_v} y_\Gamma \leq w(v) \quad \forall v \in V(G) \\ y_\Gamma \geq 0 \quad \forall \Gamma \in C. \end{array} \right.$$

3.2 Polynomially solvable cases

As it is unlikely that there exist polynomial time algorithms to solve NP-hard problems, to obtain polynomial time algorithms one has to restrict these problems to special classes of graphs. Perfect graphs are a class of graphs

on which polynomial time algorithms have been found for a variety of problems. Because the subclasses of perfect graphs form hierarchies, one problem of interest is to determine the largest class of graphs on which such problems remain polynomially solvable. Therefore, given the NP-completeness of the feedback vertex set problem, one approach is to identify those specially structured problems which can be solved in polynomial time. Research along this line started with the pioneering work of Shamir [76], in which a linear time algorithm was proposed to find a feedback vertex set for a reducible flow graph. Wang, Lloyd, and Soffa [87] developed an $O(|E(G)| \cdot |V(G)|^2)$ algorithm for finding a feedback vertex set. The class of graphs known as *cyclically reducible graphs*, which is shown to be unrelated to the class of quasi-reducible graphs. Although the exact algorithm proposed by Smith and Walford [80] has exponential running time in general, it returns an optimal solution in polynomial time for certain types of graphs. A variant of the algorithm, called the Smith-Walford-one algorithm, selects only candidate sets F of size one and runs in $O(|E(G)| \cdot |V(G)|^2)$ time. The class of graphs for which it finds a feedback vertex set is called Smith-Walford one-reducible.

In the following, consider a set of operations called *contraction operations*. These operations contract the graph while preserving all the important properties relevant to the minimum feedback vertex set. An important property associated with these operations is the so-called Church-Rosser property which implies that the order by which a sequence of operations is performed will not affect the final graph. The basic contraction operations are as follows:

Reduction Procedures

Let (G, w) be a vertex (arc) weighted graph and let V' be a feedback vertex of G , then

IN0(i) - OUT0(i):

if $|out(i)| = 0$ or $|in(i)| = 0$, then $i \in V'$.

reduction: $V(G) = V(G) \setminus \{i\}$;

$E(G) = E(G) \setminus \{(x, y) \mid x = i \text{ or } y = i\}$.

LOOP(i):

if $(i, i) \in E(G)$, then $i \in V'$.

reduction: $V(G) = V(G) \setminus \{i\}$;

$E(G) = E(G) \setminus \{(i, j) \text{ or } (j, i) \mid \forall j \in V(G)\}$.

IN1(i):

if $|in(i)| = 1$ and $(j, i) \in E(G)$,
reduction: $V(G) = V(G) \setminus \{i\}$;
 $out(j) = out(j) \cup out(i)$;
 $E(G) = E(G) \cup \{(j, k) | k \in out(i)\} \setminus \{(i, k) | k \in out(i)\}$.

OUT1(i):

if $|out(i)| = 1$ and $(i, j) \in E(G)$
reduction: $V(G) = V(G) \setminus \{i\}$;
 $in(i) = in(i) \cup in(j)$;
 $E(G) = E(G) \cup \{(k, j) | k \in in(i)\} \setminus \{(k, i) | k \in in(i)\}$.

Levy and Lowe [55] proposed these operations and proved the following properties.

Definition 1 Let $G = (V, E)$ be a directed graph, and let $G' = (V', E')$ be a directed graph resulting from G by repeated applications of the contraction operations until no further contraction is possible. G' is called a contracted graph of G .

Theorem 1 If G contains no parallel edges, then the contracted graph G' of G is unique.

Theorem 2 Let $G = (V, E)$ be a directed graph. (1) If G is contracted into G' by any of the operations $IN0(v)$, $OUT0(v)$, $IN1(v)$, or $OUT1(v)$, and if S' is a minimum feedback vertex set of G' , then S' is a minimum feedback vertex set of G . (2) If G is contracted into G' by $LOOP(v)$ and S' is a minimum cutset of G' , then $S = S' \cup \{v\}$ is a minimum feedback vertex set of G .

It should be noted that for a directed graph with every node having at least two in-edges and two out-edges, the set of contraction rules does not apply and all the above mentioned algorithms will fail to find an optimal feedback vertex set. In other words, this class of algorithms is only optimal for specially structured and very sparse graphs. Nevertheless, this line of work has significant impact in the study of feedback vertex set for the following two reasons. First, a class of graphs of increasing size is computed, where the feedback vertex set of each graph can be found exactly. Second, most proposed heuristics and approximation algorithms use the reduction schemes discussed above.

3.2.1 The feedback vertex set problem on chordal, permutation, and interval graphs

Another line of research on polynomially solvable cases focuses on other special classes, including *chordal* and *interval* graphs, *permutation* graphs, *convex bipartite* graphs, *cocomparability* graphs and on *meshes* and *toroidal meshes*, *butterflies*, and *toroidal butterflies*.

The feedback vertex set on chordal and interval graphs can be viewed as a special instance of the generalized clique cover problem, which is solved in polynomial time on chordal graphs [21, 89] and interval graphs [63]. For permutation graphs, an algorithm due to Brandstädt and Kratsch [7] was improved by Brandstädt [8] to run in $O(|V(G)|^6)$ time. More recently, Liang [57] presented an $O(|V(G)| \cdot |E(G)|)$ algorithm for permutation graphs that can be easily extended to *trapezoid graphs* while keeping the same time complexity. A graph $G = (V, E)$ is called *trapezoid* if and only if it is the intersection graph of some trapezoid diagram, where a *trapezoid diagram* T consists of two parallel lines called top and bottom lines and some trapezoids, each having two corner points on the top line and two on the bottom line. The intersection graph of a trapezoid diagram is the graph whose vertices have a one-to-one correspondence with the trapezoids in T and two vertices in G are adjacent if and only if the corresponding trapezoids in T intersect, i.e. their areas overlap. Both permutation and interval graphs are special trapezoid graphs.

On interval graphs, Lu and Tang [17] developed a linear-time algorithm to solve the minimum weighted feedback vertex set problem using dynamic programming. Interval graphs are special graphs that admit a so called *interval representation* F . An interval graph corresponds to a set of intervals (interval representation) in the real line. Each interval corresponds to a vertex and there is an edge between two vertices if and only if the corresponding intervals intersect. Such a representation of an interval graph can be obtained in $O(|V(G)| + |E(G)|)$ time, assuming sorted endpoints. Lu and Tang observed that a subset V' of $V(G)$ is a feedback vertex set of G if $V(G) \setminus V'$ is a *cycle-free vertex set* (CVS) of G . When G is weighted, V' is a minimum feedback vertex set of G if and only if $V(G) \setminus V'$ is a maximum weighted CVS of G . Consequently, given an interval representation I of a weighted interval graph G with sorted endpoints, and assuming without loss of generality, that no two intervals share a common endpoint, the linear-time algorithm of Lu and Tang finds the maximum weighted CVS of G to solve the minimum weighted feedback vertex set problem of G . At

the same time, it also solves the maximum weighted 2-colorable subgraph problem and the maximum weighted 2-independent set problem, which are equivalent on chordal graphs.

3.2.2 The feedback vertex set problem on cocomparability graphs

Coorg and Rangan [20] present an $O(|V(G)|^4)$ time and $O(|V(G)|^4)$ space exact algorithm for cocomparability graphs, which are a superclass of permutation graphs. In more detail, a graph $G(V, E)$ is said to be a cocomparability graph if and only if its complement graph is *transitively orientable*, i.e. its edges can be oriented to get a directed graph $\tilde{G} = (V, \tilde{E})$ such that

$$(i, j), (j, k) \in \tilde{E} \implies (i, k) \in \tilde{E}.$$

The authors proved that to solve the minimum feedback vertex set problem on cocomparability graphs is equivalent to finding a minimum *cycle-free* set of the given graph. The key idea is that a *cycle-free* subgraph of G is a collection of trees, because it does not contain any cycle. Therefore, a *cycle-free* subgraph is bipartite and admits a *planar embedding*, i.e. a mapping of its n vertices to a set of n points on the plane and a mapping from its m edges to a set of m line segments on the plane such that:

- there exists a line segment $[p_1, p_2]$ if and only if $(v_1, v_2) \in E(G)$, where v_1 and v_2 are mapped to p_1 and p_2 , respectively;
- any two line segments $[p_1, p_2]$ and $[p_3, p_4]$ intersect only at their endpoints.

A natural way to embed a cycle-free subgraph on the plane is the *canonical ordering* of the vertices of the transitively oriented graph G'' obtained from G through its complement $G' = (V', E')$ which can be done in $O(n^2)$ time using topological sort. An *ordering* of the vertices of G'' is *canonical* if and only if

$$(i, j) \in E(G'') \implies i < j.$$

For G'' such an ordering always exists, because G'' is a directed acyclic graph. Once a planar embedding is obtained, the algorithm of Coorg and Rangan conceptually triangulates the planar graph so that the cycle-free subgraph can be considered as a set of triangles, each having three vertices of G . In more detail, the cycle-free subgraph is incrementally built, starting from the empty set and adding one triangle at time. At each step it maintains

the incremented subgraph cycle-free. The authors prove that the cycle-free property can be maintained by checking only the last inserted triangle. They model this process by constructing a directed graph H , whose vertices correspond to the triangles and an edge (u, v) exists whenever the algorithm can safely add triangle v to the set immediately after u has been added. A directed path of length k in H corresponds to a cycle-free subgraph of G having $k + 3$ vertices and the minimum feedback vertex set problem is reduced to that of finding the longest path in a directed graph, which can be easily solved. In fact, to find the longest path in H takes time proportional to the number of the edges in H , by using, for example, depth-first search. Since there are at most $O(n)$ neighbors for each vertex in H , the number of edges in H is bounded by $O(n^4)$ and since to construct H requires $O(n^4)$ time, the complexity of the algorithm due to Coorg and Rangan remains $O(n^4)$.

More recently, Liang and Chang [13] developed a polynomial time algorithm, that by exploring the structural properties of a cocomparability graph uses dynamic programming to get a minimum feedback vertex set in $O(|V(G)|^2 |E(G)|)$ time.

3.2.3 The feedback vertex set problem on meshes and butterflies

A recent line of research on polynomially solvable cases focuses on special undirected graphs having bounded degree and that are widely used as connection networks, namely *meshes* and *toroidal meshes*, *butterflies* and *toroidal butterflies*.

Definition 2 *An $m \times n$ mesh is a graph $M_{m,n} = (V, E)$, where $V = \{v_{ij} \mid 0 \leq i \leq m-1, 0 \leq j \leq n-1\}$ and $E = \{(v_{ij}, v_{i(j+1)}), (v_{ij}, v_{(i+1)j}) \mid i = 0, \dots, n-2, j = 0, \dots, m-2\}$.*

Definition 3 *A toroidal $m \times n$ mesh is a graph $TM_{mn} = (V, E)$ obtained from a mesh $M_{(m+1)(n+1)}$ identifying the vertices v_{0j} with v_{mj} , $0 \leq j \leq n$ and v_{j0} with v_{jn} , $0 \leq j \leq m$.*

In a mesh $M_{m,n}$, Luccio [61] obtained a trivial lower bound on the size of the minimum feedback vertex set V' equal to $((m-1)(n-1) + 1)/4$, by observing that every submesh M_{22} is a cycle of 4 vertices to be broken and one vertex of V' breaks at most 4 such cycles. Luccio proved the stronger lower bounds $|V'| \geq ((m-1)(n-1) + 1)/3$ in a mesh M_{mn} and $(mn + 2)/3$ in a toroidal mesh TM_{mn} . In [61], upper bounds for meshes of size $(2^r +$

$1) \times (2^r + 1)$ and toroidal meshes $TM_{2^r 2^r}$ are derived. These bounds either match the lower bounds or are very close to them. The proofs of the upper bounds are constructive, in the sense that an algorithm that derives them also finds a minimum feedback vertex set V' for the given graph, simply by investigating their topological characteristics. In more detail, in the case of meshes $M_{(2^r+1)(2^r+1)}$, V' is formed by patterns of vertices lying on the diagonal lines that constitute the boundaries of diamonds of varying sizes $2^j + 1$, $0 \leq j \leq r - 2$, each having exactly one vertex not belonging to V' . By applying this algorithm to $M_{(n+1)(n-1)}$ and removing the $(n+1)^{th}$ row and the $(r+1)^{th}$ column, on which no vertex of V' lies, one can obtain a minimum feedback vertex set for $TM_{2^r 2^r}$. This algorithm can be still applied to arbitrary values of m and n . Actually, Luccio showed that to obtain a minimum feedback vertex set for a mesh M_{mn} , $2^{r-1} + 1 < m \leq 2^r + 1$, $2^{s-1} + 1 < n \leq 2^s + 1$, $r, s \geq 0$, $t = \max\{r, s\}$, it is enough to apply the proposed algorithm to build V' for $M_{(2^t+1)(2^t+1)}$, and then the required minimum feedback vertex set for M_{mn} is the portion of V' lying in the upper left submesh M_{mn} . To get a minimum feedback vertex set V'' for TM_{mn} it is possible to do the same, with the only difference that now additional vertices are needed to be inserted along the external boundary of V'' in order to break possible cycles between the first and the last row and the first and the last column, thus increasing the upper bound by an additional linear term.

For *butterfly* and *toroidal butterfly* graphs, Luccio [61] found upper bounds to the size of a minimum feedback vertex set.

Definition 4 A k -dimensional butterfly is a graph $B_k = (V, E)$ having $(k+1)2^k$ vertices organized in $k+1$ levels of 2^k vertices each. Vertex v_{ij} denotes the j^{th} vertex at level i , $0 \leq i \leq k$, $0 \leq j \leq 2^k - 1$, and for $i > 0$ it is connected with the two vertices $v_{(i-1)j}$ and $v_{(i-1)j_i}$, where j_i is the integer whose binary representation differs from that of j in only the i^{th} least significant bit.

Definition 5 A toroidal butterfly TB_k is a k -dimensional butterfly in which the k^{th} level coincides with level 0.

In a k -dimensional butterfly, the vertices in every two adjacent levels form 2^{k-1} cycles with two vertices in the upper level and two in the lower level. The lower bound of the size on a minimum feedback vertex set \bar{V} for

this graph is

$$|\bar{V}| \geq 2^{k-1} \lfloor \frac{(k+1)}{2} \rfloor,$$

because each vertex at level i , $1 \leq i \leq k-1$, belongs to exactly two cycles: one between level $i-1$ and i and one between i and $i+1$. The upper bound is

$$|\bar{V}| \leq (2^{k-2} + 2^{k-4} + 2^{k-5} + 1) k,$$

found by applying an algorithm that for each vertex $v_{(i-1)j}$ adds to \bar{V} either v_{ij} or v_{ij} , connected with $v_{(i-1)j}$ at the next level.

Similar results to those obtained for butterflies B_k can also be obtained for toroidal butterflies TB_k . In fact, TB_k has $k2^k$ vertices and the vertices belonging to the $(k-1)^{th}$ level are connected to those of level 0. Therefore, such kinds of graphs contain 2^k cycles wrapped around each column. To break all of them, it is necessary to remove at least

$$\max\{(2^{k-1}) \lfloor \frac{k+1}{2} \rfloor, 2^k\}$$

vertices, where 2^k is significant only for $k \leq 2$. To find a minimum feedback vertex set \bar{V} , the same algorithm as for B_k can be applied, by including in \bar{V} all vertices belonging to the $(k-1)^{th}$ level.

3.2.4 The feedback vertex set problem on cube connected cycle networks

Luccio [61] solved the minimum feedback vertex set problem also for another important bounded degree network, the k -dimensional cube connected cycle (CCC_k), that has k levels of 2^k vertices each. The only difference between TB_k and CCC_k is that any vertex v_{ij} is now connected with $v_{(i-1)j}$ and $v_{i(j+1)}$, $0 \leq i \leq k-1$, $0 \leq j \leq 2^k - 1$, and the operations on i are computed modulo k . Observing that the vertices in every two adjacent levels form 2^{k-2} cycles of length 8 (4 in the upper level and 4 in the lower level) and that each vertex belongs exactly to two such cycles, at least $2^{k-2} \lfloor (k+1)2 \rfloor$ vertices have to be removed in order to break all such cycles. Moreover, because CCC_k also contains 2^k cycles wrapped around each column, a lower bound on the minimum number of vertices to be removed is given by

$$\max\{2^k, 2^{k-2} \lfloor \frac{(k+1)}{2} \rfloor\},$$

where 2^k is insignificant for $k \leq 6$. The upper bound found by Luccio is $2^{k-1} \lfloor \frac{(k+1)}{2} \rfloor$, obtained by building a minimum feedback vertex set using a variant of the algorithm that finds a minimum feedback vertex set for TB_k . In this case, for each level $i = 1, \dots, k$, incrementing i by 2, 2^{k-1} vertices are inserted in the feedback set to break all cycles with a vertex at the lower level $i - 1$. The remaining vertices belonging to level $k - 1$ are inserted in the feedback set.

3.2.5 The feedback vertex set problem on convex bipartite graphs

Liang and Chang [13] solved in polynomial time the feedback vertex set problem in a special kind of bipartite graph, called the *convex bipartite graph*, whose definition is given next.

Definition 6 A bipartite graph $G = (A, B, E)$ with two distinct sets of vertices A and B is convex if there exists a total ordering on A such that for any vertex $b \in B$ the set of vertices of A connected to b forms an interval in this ordering.

Liang and Chang [13] proposed a polynomial time algorithm having time complexity $O(|A|^3 + |A|^2|E|)$. Their algorithm assumes a convex bipartite graph $G = (A, B, E)$ is given by specifying the total ordering on $A = \{a_1, a_2, \dots, a_n\}$ with $a_1 < a_2 < \dots < a_n$. The algorithm is based on dynamic programming techniques and the special structure of the graph.

3.3 Approximation algorithms and provable bounds

The feedback vertex (arc) set problem has found applications in many fields, including deadlock prevention [87], program verification [76], and Bayesian inference [2]. Therefore, it is natural that in the past few years there have been intensive efforts on approximation algorithms for these kinds of problems, for the cases that are not known to be polynomially solvable.

To quantify the quality of an approximation, several criteria have been established and certain classes of approximation schemes have been defined. The family of approximation algorithms that guarantees the best approximate solution is the so called *fully polynomial approximation scheme* (FAS). It is a family of algorithms $\{A_\epsilon\}$ that for a maximization problem finds an approximate solution at least $(1 - \epsilon)$ times the optimal solution in time polynomial in the length of the input and in $1/\epsilon$. In case of a minimization problem, the algorithm is guaranteed to find an approximate solution at

least $(1 + \epsilon)$ times the value of the optimal solution. The next family of “good” approximation algorithms is the *polynomial approximation scheme* (PAS). It contains approximation algorithms that produce a solution at least $(1 + \epsilon)$ times the optimal solution in time polynomial in the length of the input for fixed ϵ .

When an approximation algorithm does not have any of the aforementioned characteristics, it can still be “evaluated” as well. In fact, to test the quality of the approximate solution there are special quantities called *performance ratios*, defined as follows. Suppose A is an approximation algorithm that finds a feedback vertex set V_A (arc set E_A) for any given vertex (arc) weighted graph (G, w) . Denoting the sum of weights of vertices (arcs) in V_A (E_A) by $w(V_A)$ and the weight of a minimum feedback vertex (arc) set for (G, w) by $\mu(G, w)$, then the *performance ratio of A for (G, w)* is defined by

$$R_A(G, w) = w(V_A)/\mu(G, w).$$

When $\mu(G, w) = 0$, $R_A(G, w) = 1$ if $w(V_A) = 0$ and $R_A(G, w) = \infty$ if $w(V_A) > 0$.

In other words, the performance ratio of an approximation algorithm is the worst-case ratio between the weight of the algorithm’s output and the weight of an optimal solution.

The *performance ratio $r_A(n, w)$ of A for w* is the supremum of $R_A(G, w)$ over all graphs G with n vertices and for the same weight function w . If w is the constant function ($w(i) = 1$), $r_A(n, 1)$ is called the *unweighted performance ratio of A* . The *performance ratio $r_A(n)$ of A* is the supremum of $r_A(n, w)$ over all weight functions w .

Many approximation algorithms for feedback set problems have been proposed in the last two decades. We consider approximation algorithms for undirected and directed graphs next.

3.3.1 Undirected graphs

A $2 \log_2 |V(G)|$ -approximation algorithm for the unweighted minimum feedback vertex set problem on undirected graphs is contained in a lemma due to Erdős and Posa [26]. This result was improved by Monien and Schulz [64] to obtain a performance ratio of $O(\sqrt{\log |V(G)|})$.

Bar-Yehuda, Geiger, Naor, and Roth [2] gave an approximation algorithm for the unweighted undirected case having ratio less than or equal to 4 and two approximation algorithms for the weighted undirected case having ratios $4 \log_2 |V(G)|$ and $2\Delta^2(G)$, respectively. A slight generalization of the

unweighted case was used to reduce the complexity of a Bayesian inference procedure as will be discussed in Section 5. In their algorithm, it is assumed that the set of vertices $V(G)$ is partitioned into a nonempty set $A(G)$ of *allowed* vertices and a possibly empty set $B(G)$ of *blackout* vertices. For a *valid graph* G a feedback vertex set exists if and only if every cycle in G contains at least one allowed vertex. Note that if $B(G) = \emptyset$, then the problem reduces to the classical unweighted feedback vertex set. Before giving a description of their algorithm, the following definitions are needed.

Definition 7 A 2-3-subgraph of a valid graph G is a subgraph H of G such that the degree in H of every vertex in $V(G)$ is either 2 or 3. A maximal 2-3-subgraph of G is a 2-3-subgraph of G that is not a subgraph of any other 2-3-subgraph.

A maximal 2-3-subgraph of G always exists if G is a valid graph that is not a forest.

Definition 8 A linkpoint v in a 2-3-subgraph H is said to be a critical linkpoint if it is an allowed vertex and there is a cycle Γ in G such that

$$V(\Gamma) \cap V(H) \cap E(G) = \{v\}.$$

In this case Γ is called a witness cycle of v .

Definition 9 A cycle in a valid graph G is branchpoint-free if it does not pass through any allowed branchpoint, i.e. a branchpoint-free cycle is made of only blackout vertices and allowed linkpoints of G .

The algorithm in [2] takes as input a 2-3-subgraph of the given valid graph G and gives as output a feedback vertex set of G , containing the set X of all critical linkpoints, the set Y of allowed branchpoints in the maximal 2-3-subgraph of G , and the vertices of a set W covering all branchpoint-free cycles of H not covered by X . The set X can be found by applying depth-first search on G and the set Y by applying breadth-first search. The complexity of the algorithm is linear in the number of edges of G . To speedup the algorithm, Bar-Yeruda et al. show how to preprocess the input valid graph by applying the corresponding undirected versions of the reduction transformations given in Subsection 3.2, by being careful during the process in not generating any blackout vertex cycle that violates the validity property of the reduced graph.

In more detail, in the undirected case, the reduction procedures described in Subsection 3.2 degenerate to the following. A *reduction graph* G' of a graph G is a graph obtained from G by a sequence of the following transformations:

- Delete an endpoint and its incident edges.
- Connect two neighbors of a linkpoint v (other than a self-looped singleton) by a new edge and remove v from the graph with its incident edges.

It was proved in [2] that any valid reduction graph G' of a graph G is such that $\mu(G', w) = \mu(G, w)$. Moreover, if the reduction graph G^* of G is *minimal*, i.e. it is valid and any proper reduction graph G' of G^* is not valid, it is shown that G^* does not contain any blackout linkpoints and that any of its feedback vertex sets contains all allowed vertices of G^* . In this case, if it is possible to obtain a valid graph G^* without any blackout linkpoints and any endpoints, as the reduction transformations do, then for every feedback vertex set V' of G^* containing all linkpoints of G , we have that

$$|V(G)| \leq (\Delta(G) + 1)|V'|,$$

where $\Delta(G)$ is the largest degree among all vertices in G .

In [2], the weighted feedback vertex set problem for undirected graphs was also solved by proposing two approximation algorithms having performance ratios $4 \log_2 |V(G)|$ and $2\Delta^2(G)$, respectively. Both of the algorithms use reduction graph transformations. The first algorithm finds at each iteration a minimal weighted reduction graph G^* of the input graph (G, w) , i.e. a minimal reduction graph such that any of its reduction graphs G' is equal to G^* . Such a graph is necessarily *branchy*, that is, it does not contain any endpoints and its set of linkpoints induces an independent set (i.e. each linkpoint is either an isolated self-loop or it is connected to two branchpoints). Note that transforming a graph into a branchy graph takes $O(|E(G)|)$ time. The algorithm then proceeds to find a cycle Γ in the minimal weighted reduction graph with the smallest number of vertices of length less than or equal to $4 \log_2 |V(G)|$. Next, the algorithm subtracts from the weight of each vertex in $V(\Gamma)$ the minimum among the weights of vertices in $V(\Gamma)$. The vertices whose weights become zero are added to the building feedback vertex set V' and deleted from the graph. These steps are iterated until the graph is exhausted. It has been proved that this algorithm can

be applied even on planar graphs achieving a performance ratio less than or equal to 10. The second algorithm for the weighted feedback vertex set problem in its undirected version is based on the property that every feedback vertex set V' of a branchy graph G^* contains a number of vertices less than or equal to $2\Delta^2(G)|V'|$. This is a greedy algorithm that achieves a performance ratio less than or equal to $2\Delta^2(G)$ for any graph G . It starts each iteration i by finding the minimal weighted reduction graph (H_i, w_{H_i}) of the graph $(H_{i-1}, w_{H_{i-1}})$, where $(H_0, w_{H_0}) = (G, w)$ and by computing α_i , the minimum weight among those associated with the vertices of H_i . It then proceeds by adding to the feedback vertex set V' all the vertices in H_i having weight α_i and removing them from H_i . These steps are iterated until the reduced graph H_i becomes a forest.

For the feedback vertex set problem in general undirected graphs, two slightly different 2-approximation algorithms are described in Becker and Geiger [3] and by Bafna, Berman, and Fujito [1]. These algorithms improve the approximation algorithms of Bar-Yeruda et al. [2], who also give a reduction procedure from the *loop cutset problem* to the minimum weighted feedback vertex set problem. The proposed algorithms also can find a loop cutset which, under specific conditions explained in more detail in Section 5, is guaranteed in the worst case to contain less than four times the number of variables contained in a minimum loop cutset. Subsequently, Becker and Geiger [4] applied the same reduction procedure from the loop cutset problem to the minimum weighted feedback vertex set problem of Bar-Yeruda et al. [2], but their result is independent of any condition and is guaranteed in the worst case to contain less than twice the number of variables contained in a minimum loop cutset. Becker and Geiger [4] propose two approximation algorithms for finding the minimum feedback vertex set V' in a vertex-weighted undirected graph (G, w) . The simplest of them is a *greedy algorithm* that starts with a graph G' , after removing from the original graph G all vertices $i \in V(G)$ with degree $\Delta_G(i)$ equal to 0 or to 1 and then, it repeatedly chooses to insert a vertex v in the feedback vertex set V' , if the ratio between v 's weight $w(v)$ and its degree $\Delta_{G'}(v)$ in the current reduced graph G' is minimal across all vertices in G' . When v is selected, it is removed from G' along with all its incident edges together with all vertices of degree equal to 0 or to 1 after those removals. This step is iterated until the current reduced graph is exhausted. It is proved that the performance ratio of this algorithm is bounded by $2 \log \Delta(G) + 1$, where $\Delta(G) = \max_{v \in V(G)} \Delta_G(v)$ is the degree of graph G . The second algorithm is a modified greedy algo-

rithm having performance ratio bounded by the constant 2. It consists of two phases:

1. The first phase proceeds as in the greedy algorithm, but now, when a vertex v is chosen to be removed from the graph with all its incident edges and to be inserted in the building feedback vertex set V' , the ratio $r(v)$ between its weight and its degree is saved. After the removal from the current graph of all vertices having degree 0 or 1, along with their incident edges, for every removed edge (u_1, u_2) the ratio $r(v)$ is subtracted from the weight of its endpoints u_1 and u_2 .
2. The second phase, on the other hand, is completely new. It starts after exhausting the current reduced graph, i.e. when a feedback vertex set V' is already found, and tries to remove from V' redundant vertices v_i by checking if every cycle in G that passes through v_i intersects with $V' \setminus \{v_i\}$.

By using a Fibonacci heap, the complexity of the algorithm is $O(m + n \log n)$, where $m = |E(G)|$ and $n = |V(G)|$. A vertex is identified and retrieved from such a heap $|V(G)|$ times, with each operation taking $O(\log |V(G)|)$ time. The vertex weights are decreased $|E(G)|$ times at a constant amortized cost each. Moreover, the complexity of the second phase does not increase the total complexity of the algorithm.

In [18], Chudak, Goemans, Hochbaum, and Williamson showed how the algorithms due to Becker and Geiger [3] and Bafna, Berman, and Fujito [1] can be explained in terms of the primal-dual method for approximation algorithms that are used to obtain approximation algorithms for network design problems. The primal-dual method starts with an integer programming formulation of the problem under consideration. It then simultaneously builds a feasible integral solution and a feasible solution to the dual of the linear programming relaxation. If it can be shown that the value of these two solutions is within a factor of α , then an α -approximation algorithm is found. The *integrality gap* of an integer program is the worst-case ratio between the optimum value of the integer program and the optimum value of its linear relaxation. Therefore, by applying the primal-dual method it is possible to proof that the integrality gap of the integer program under consideration is bounded. In fact, Chudak et al., after giving a new integer programming formulation of the feedback vertex set problem, provided a proof that its integrality gap is at most 2. They also gave the proofs of some key inequalities needed to prove the performance guarantee of their

new 2-approximation algorithm, which is a simplification of the algorithm due to Bafna et al. [1].

Theorem 3 *Let V' denote any feedback vertex set of a graph $G = (V, E)$, $E \neq \emptyset$, let τ denote the cardinality of the smallest feedback vertex set for G , and let $E(S)$ denote the subset of edges that have both endpoints in $S \subseteq V(G)$, $b(S) = |E(S)| - |S| + 1$. Then*

$$\sum_{v \in V'} [\Delta_G(v) - 1] \geq b(V(G)) \quad (1)$$

$$\sum_{v \in V'} \Delta_G(v) \geq b(V(G)) + \tau. \quad (2)$$

If every vertex in G has degree at least two, and V'_M is any minimal feedback vertex set (i.e. $\forall v \in V'_M$, $V'_M \setminus \{v\}$ is not a feedback vertex set), then

$$\sum_{v \in V'_M} \Delta_G(v) \leq 2(b(V(G)) + \tau) - 2. \quad (3)$$

Definition 10 *A feedback vertex set V' is almost minimal if there is at most one vertex $v \in V'$ such that $V' \setminus \{v\}$ is a feedback vertex set.*

Definition 11 *A cycle is semidisjoint if it contains at most one vertex of degree greater than 2.*

Theorem 4 *If every vertex has degree at least 2 and the graph does not contain semidisjoint cycles or is itself a cycle, and V'_{AM} is any minimal feedback vertex set, then*

$$\sum_{v \in V'_{AM}} [\Delta_G(v) - 1] \leq 2b(V(G)) - 1. \quad (4)$$

Note that the inequalities 2 and 3 imply

$$\sum_{v \in V'_M} \Delta_G(v) \leq 2 \sum_{v \in V'} \Delta_G(v) - 2, \quad (5)$$

while inequalities 1 and 4 imply

$$\sum_{v \in V'_{AM}} [\Delta_G(v) - 1] \leq 2 \sum_{v \in V'} [\Delta_G(v) - 1] - 1. \quad (6)$$

The condition under which the inequality 4 holds is satisfied if the graph is 2-vertex-connected. As Corollary of inequality 1, Chudak et al. proved the following result:

Theorem 5 *Let V' be any feedback vertex set. Then, for any $S \subseteq V(G)$ such that $E(S) \neq \emptyset$, we have that*

$$\sum_{v \in V' \cap S} [\Delta_S(v) - 1] \geq |E(S)| - |S| + 1 = b(S). \quad (7)$$

Inequalities 3 and 4 are needed to prove the performance guarantee of the proposed approximation algorithm, while 1 and 2 are used in the new integer programming formulation. The standard cycle formulation of the problem is given in Subsection 3.1. Even, Naor, Schieber, and Zosin [27] showed that the integrality gap of that integer program is $\Omega(\log n)$. The new integer programming formulation given by Chudak et al. [18] is as follows:

$$(IP) \left\{ \begin{array}{l} \min \sum_{v \in V(G)} w(v) x_v \\ \text{s.t. } \sum_{v \in S} (\Delta_S(v) - 1) x_v \geq b(S) \quad S \subseteq V(G) : E(S) \neq \emptyset \\ x_v \in \{0, 1\} \quad v \in V(G). \end{array} \right.$$

The linear programming relaxation is:

$$(LP) \left\{ \begin{array}{l} \min \sum_{v \in V(G)} w(v) x_v \\ \text{s.t. } \sum_{v \in S} (\Delta_S(v) - 1) x_v \geq b(S) \quad S \subseteq V(G) : E(S) \neq \emptyset \\ x_v \geq 0 \quad v \in V(G). \end{array} \right.$$

and its dual is:

$$(D) \left\{ \begin{array}{l} \max \sum_S b(S) y_S \\ \text{s.t. } \sum_{S: v \in S} (\Delta_S(v) - 1) y_S \leq w_v \quad v \in V(G) \\ y_S \geq 0 \quad S \subseteq V(G) : E(S) \neq \emptyset. \end{array} \right.$$

The algorithm proposed by Chudak et al. constructs a feasible solution of the linear programming relaxation (LP) and it is essentially the algorithm proposed by Bafna et al. In [1], Bafna et al. developed an algorithm that

starts with a feedback vertex set $V' = \emptyset$, the feasible dual solution $y = 0$, and the original graph $G^* = (V^*, E^*) = (V, E)$. Given a set V' , if it is not a feedback vertex set, there must exist a cycle in G^* . The algorithm first recursively removes from G^* all vertices having degree equal to one and their incident edges and then chooses some set S corresponding to a violated constraint of (IP). S is called the *violated set*. To choose S the algorithm uses a subroutine called VIOLATION, that first looks for a semidisjoint cycle in G^* . If it finds such a cycle, it lets S correspond to the vertices of the cycle and returns S . Otherwise, it returns $S = V^*$. The algorithm then increases as much as possible the dual variable y_S , until some dual inequality becomes tight for some vertex in S , that is added to V' and removed from G^* together with its incident edges. The algorithm continues repeating these steps until V' is a feedback vertex set. It then goes through the vertices in V' in the reverse of the order in which they were added and removes any extraneous vertices. The simpler 2-approximation algorithm of Chudak et al. uses a different VIOLATION subroutine, that avoids searching for semidisjoint cycles by a different choice of S . Given a decomposition of G^* into its 2-vertex-connected components, their algorithm sets S to be an endblock, so that S contains at most one cutvertex.

Another 2-approximation algorithm is due to Vazirani [86], who also found a lower bound on the optimal solution for special vertex weight functions. Before describing his algorithm, the following definitions are needed.

Definition 12 Let $GF[2]^i$ be the set of all binary vectors having i components. Then the characteristic vector of a cycle C in G is a vector in $GF[2]^m$, $m = |E(G)|$, whose components are equal to 1, if they correspond to edges in C , 0 otherwise.

Definition 13 The cycle space of G is defined as the subspace of $GF[2]^m$, spanned by the characteristic vectors of all cycles in G .

Definition 14 The cyclomatic number $cyc(G)$ is the dimension of the cycle space and it is given by

$$cyc(G) = |E(G)| - |V(G)| + k(G),$$

where $k(G)$ denotes the number of connected components of G .

The removal of an edge i from G decreases its cyclomatic number by 1, unless i is a *bridge*, i.e. an edge whose removal increases the number

of connected components. Similarly, the decrease in the cyclomatic number caused by removing a vertex v equals the maximum number of edges incident to v that can be successively removed so that none of them is a bridge at the moment of its removal.

Definition 15 Let $\delta_G(v)$ be the decrease in cyclomatic number caused by removing the vertex v from G . A weight function is cyclomatic if it assigns to each vertex v the weight $c \cdot \delta_G(v)$, for some $c > 0$.

Since the removal of a feedback vertex set $V' = \{v_1, \dots, v_f\}$ decreases the cyclomatic number of G down to 0, then

$$cyc(G) = \sum_{i=1}^f \delta_{G_{i-1}}(v(i)),$$

where G_0 is the original graph and, for $i > 0$, $G_i = G \setminus \{v_1, \dots, v_i\}$.

Since a non-bridge edge in an induced subgraph cannot be a bridge in the larger graph, for each vertex v the following inequalities hold

$$\delta_{G_i}(v) \leq \delta_G(v),$$

$$cyc(G) \leq \sum_{v \in V'} \delta_G(v).$$

Therefore, if the weight function defined on the vertices of G is cyclomatic, then $c \cdot cyc(G)$ is a lower bound on the optimal solution and a straightforward factor 2 algorithm for cyclomatic weights functions is given by the following lemmas:

Lemma 1 If V' is a minimal feedback vertex set of G , then

$$\sum_{v \in V'} \delta_G(v) \leq 2 \cdot cyc(G).$$

Lemma 2 Let w be a cyclomatic weight function defined on $V(G)$ and let V' be a minimum feedback vertex set of G . Then

$$w(V') \leq \mu(G, w).$$

In [86], Vazirani solves the feedback vertex set problem also in the case of arbitrary weights. The proposed 2-approximation algorithm consists of two phases: a *decomposition* phase and an *extension* phase. During the

decomposition phase, the algorithm decomposes the original graph G into a sequence of induced subgraphs, $G_k \subset G_{k-1} \subset \dots \subset G_0 = G$, where G_k is acyclic. On the vertices of each graph G_i , $i < k$, a weight function is defined such that the sum of the weights of a vertex among these subgraphs is equal to its original weight, i.e.

$$\sum_{i:v \in V(G_i)} w_i(v) = w(v).$$

When the decomposition phase terminates, V'_k is a minimal feedback vertex set of G_k . The algorithm proceeds then executing the extension phase. For each G_i , $i < k$, it computes a minimum feedback vertex set V'_i by adding to V'_{i+1} a minimal set of vertices from $V(G_i) - V(G_{i-1})$. The set V'_0 is output as a feedback vertex set for G .

Regarding the maximum size of a feedback vertex set V' in cubic graphs, Speckenmeyer [81] proved that for a connected undirected cubic graph G of order n and girth g (*girth* is the length of a shortest cycle in G)

$$|V'| \leq \frac{(g+1)}{(4g-2)} + \frac{(g-1)}{(2g-1)}.$$

Bondy, Hopkins, and Staton [5] proved that for a connected cubic graph G of girth at least 4

$$|V'| \leq \lceil 1/3 |V(G)| \rceil.$$

An improvement was made by Zheng and Lu, who in [92] found that if the girth of G is at least 4 and if $|V(G)| \neq 8$, then

$$|V'| \leq \lfloor 1/3 |V(G)| \rfloor$$

and the bound is sharp. These results were improved by Liu and Zhao [58], including the result due to Zheng and Lu, resulting in

$$|V'| \leq \frac{g}{4(g-1)} n + \frac{(g-3)}{(2g-2)}$$

for a large class of cubic graphs of order $n \geq 4$ and girth g .

For the subset feedback vertex problem, Even, Naor, Schieber, and Zosin [27] showed that it can be approximated in polynomial time by a factor of $\min\{2 \Delta(G), 8 \log(|V'| + 1), O(\log \tau^*)\}$, where τ^* denotes the value of the optimal fractional solution. In [27] the authors also proposed a technique, called *bootstrapping*, that enhances the $O(\log |V'|)$ to a factor of $O(\log \tau^*/\beta)$,

where β denotes the minimum weight of a vertex. The bootstrapping technique iteratively uses a graph partitioning algorithm. The output of each iteration is by itself a subset feedback vertex set and is used as part of the input of the next iteration. After $O(\log |V'|)$ iterations, the algorithm produces as output a subset feedback vertex set having weight at most $O(\tau^* \log \tau^*)$. Even, Nor, and Zosin [29] improved this result proposing a 8-approximation algorithm. The main tool that they used in developing their approximation algorithm and its analysis is a new version of multicommodity flow, called *relaxed multicommodity flow*, a hybrid of multicommodity flow and multiterminal flow. In multicommodity flow, the arc capacity constraints apply to the total flow of all the commodities, while in multiterminal flow, the arc capacity constraints apply to each commodity separately. A relaxed multicommodity flow is a multiterminal flow with additional constraints, called *intercommodity* constraints. For each arc, the authors considered the maximum flow, among all the commodities, which is shipped along it. They required that for each vertex $v \in V(G)$ the sum of the maximum flows shipped along its incident arcs is bounded by four times the capacity of v . By considering the multicommodity flow, the vertices for which the intercommodity constraints are tight play an important role from the point of view of the connectivity of the graph. They are called *intersaturated* vertices. The main result of Even et al. is a theorem that bounds the weight of the vertices that must be intersaturated, so as to realize a given demand vector by the sum of demands.

3.3.2 Directed graphs

In general, problems on undirected graphs are relatively easier to handle than problems on directed graphs, since more graph theory can be utilized. Not surprisingly, the approximation results obtained so far for the undirected version are stronger than those for the directed version. In fact, none of the algorithms referred to in the previous subsections apply to the feedback vertex set problem in directed graphs and, in contrast with the undirected version, no analytical results are known for the directed case.

A very recent direction of research on approximation algorithms in the directed version focuses on the complete equivalence among all feedback set (and/or feedback subset) problems and among these and the directed minimum capacity multicut problem in circular networks. An exhaustive description of the procedures that reduce any feedback set problem to any other or any of them to the directed minimum capacity multicut problem

and vice versa will be given in Subsection 4.2. These reduction procedures are formalized and used by Even, Naor, Schieber, and Sudan [28] to obtain an approximation algorithm for the subset feedback arc set problem of a weighted directed graph $G = (V, E)$, where the interesting cycles to be *hit* are contained in a set of special vertices $X \subseteq V(G)$, where $|X| = k$. The weight of the feedback arc set found by their approximation algorithm is $O(\tau^* \log^2 |X|)$, where τ^* is the weight of an optimal fractional feedback set. More detail is provided in Subsection 4.3. Nevertheless, their approach can be used to solve any other feedback set problem as well as the directed minimum capacity multicut problem.

Even et al. [28] also proposed an algorithm for approximating the minimum weighted subset vertex set problem in the weighted and directed case, leading to a result that holds for any other feedback set problem as well. Their approach is an algorithmic adaptation of a theoretical result due to Seymour [75], who proved that the integrality gap in the case of the unweighted feedback vertex set problem can be at most $O(\log \tau^* \log \log \tau^*)$, where τ^* is defined as above. Even et al. observe that all existence arguments contained in the proof of Seymour's statement can be made constructive and thus, with some additional operations, an algorithm for the unweighted feedback vertex set problem having an approximation factor of $O(\log \tau^* \log \log \tau^*)$ can be obtained. Further modifications of the algorithm lead to a polynomial time approximation scheme applicable to the weighted problem. In $O(|E(G)| \cdot |V(G)|^2)$ time the algorithm finds a feedback vertex set having weight $O(\min\{\tau^* \log \tau^* \log \log \tau^*, \tau^* \log |V(G)| \log \log |V(G)|\})$. All the observations contained in [28] improve the $O(\log^2 |V(G)|)$ -approximation algorithm for this case due to Leighton and Rao [53].

In the case of directed planar graphs, Stamm [83] presented an $O(|V(G)| \log |V(G)|)$ -approximation algorithm, whose performance guarantee is bounded by the maximum degree of the graph and an $O(|V(G)|^2)$ time approximation algorithm with performance guarantee no more than the number of cyclic faces in the planar embedding of the graph minus 1.

Cai, Deng, and Zang [10] obtained a 2.5-approximation algorithm for the minimum feedback vertex set problem on tournaments, improving the previously known algorithm with performance guarantee of 3 by Speckenmeyer [82]. Let H be the triangle-vertex incidence matrix of a tournament T and let e be the all-one vector. In [10], necessary and sufficient conditions are established for the linear system $\{x \mid Hx \geq e, x \geq 0\}$ to be a *totally dual integral system* (TDI).

Definition 16 A rational linear system $\{x \mid Ax \geq b, x \geq 0\}$ is called totally dual integral, if the optimization problem $\max\{y^T b \mid y^T A \leq c^T, y \geq 0\}$ has an integral optimum solution y for every integral vector c for which the maximum is finite.

It has been shown that any rational polyhedron P has a TDI System $P = \{x : Ax \leq b\}$ representation with A integral, and that, if P is full-dimension, there is a unique minimal TDI System $P = \{x : Ax \leq b\}$ with A and b integral if and only if P is integral. In [11] the authors have extended this approach to the feedback vertex set problems and the cycle packing problem in *bipartite tournaments*, where a bipartite tournament is an orientation of a complete bipartite graph. For the aforementioned problems they have found strongly polynomial time algorithms, which are a consequence of a min-max relaxation on packing and covering directed cycles.

3.4 Exact algorithms

In contrast to the numerous approximation schemes that have been studied, relatively few exact algorithms for the feedback vertex set problem have been proposed. To our knowledge, the first algorithm to find an exact minimal cardinality FVS is due to Smith and Walford [80]. Although it solves the problem in an arbitrary directed graph in exponential running time, it returns an optimal solution in polynomial time for certain types of graphs. Before giving its description, the following definitions are needed.

Definition 17 A maximal strongly connected component of a graph is abbreviated as MSC.

Definition 18 A set of vertices S of graph G with the property that each loop of G contains at least one element of S is called a complete set. The number of elements in the set S is referred to as the set measure $|S|$.

Definition 19 A complete set S of minimal size is called an optimum set and its measure is defined as the index of G denoted by $I(G)$.

The algorithm proposed by Smith and Walford is based on the following idea of partition implication. Given a graph G , an arbitrary subset of vertices F may imply a 2-subgraph partition (G_F, G_R) such that

1. Each vertex of G_F is contained in at least one loop in G and only in loops of G also containing an element of F . G_R contains all remaining vertices of G ;

2. If G is strongly connected, then all elements of any possible set F will be in G_F ;
3. By removing from G all vertices in $G_F \cap F$ all loops in G containing a vertex of G_F are eliminated from G_F ;
4. Loops in G_F and loops in G_R are independent, i.e. they do not contain any common vertex.

The authors proved the following results used by their algorithm.

Theorem 6 *If S_1, S_2, \dots, S_k are optimal feedback vertex set for partitions P_1, P_2, \dots, P_k of graph G , and if the set S given by $S = \bigcup_{i=1}^k S_i$ is complete for G , it is also optimal for G .*

Theorem 7 *If $I(G_F) = |F|$, then F is a subset of an optimal feedback vertex set of G .*

Theorem 8 *Each element of an optimum set S must belong to at least one loop of G not containing any other element of S .*

The running time of the Smith-Walford algorithm reported in the following depends on the cardinality of the partitions of the graph.

INPUT: a digraph $G(V, E)$;

OUTPUT: a minimum feedback vertex set for G .

- 1) Find the MSC subgraphs of the graph under test, store them on push c
- 2) Pop top MSC from push down stack. If the stack was empty, then exit;
- 3) Generate next trial F set for MSC;
- 4) Calculate implied partitions G_F and G_R ;
- 5) If $I(G_F) = |F|$, then go to 3);
- 6) Make F part of minimal feedback vertex set;
- 7) Break G_R subset into its MSC subgraphs;
- 8) Store them on the push down stack and go to 2).

Later exact algorithms of enumerative nature often used the graph reduction procedures to speed up the process. One study by Cheng [16] essentially used direct enumeration plus reduction and reported satisfactory computational results for a set of partial scan design test problems. Orenstein, Kohavi, and Pomeranz [65] proposed a somewhat more involved exact

enumerative procedure based on graph reduction and efficient graph partitioning methods. In addition to the reduction procedures due to Levy et al., Orenstein et al. developed a further type of operation designed *double reduction*, which involves edges deletion:

DOUBLE(v,w): If the vertices v and w form a 2-edged directed cycle in the graph G , then except for (v, w) and (w, v) , remove from G all edges included in directed cycles going through both v and w .

If v and w satisfy the hypotheses of the DOUBLE reduction, then at least one of them must be included in the feedback vertex set and all cycles passing through both v and w will be covered. Therefore, taking into account only the smallest cycle involving v and w will have no influence on the optimum feedback vertex set solution. The algorithm proposed by Orenstein et al. has been designed for identifying a minimum feedback vertex set in a digital circuit. It is based on the following recursive steps:

1. Building the *topological* graph associated with the input sequential circuit;

Definition 20 A digraph $G = (V, E)$ is called a topological graph for a circuit S if each vertex in G represents a flip-flop, and a directed edge (u, v) exists if and only if there is a path from the flip-flop u to the flip-flop v through combinatorial logic. A topological graph does not contain parallel edges.

2. Reduction operations;
3. Partition: this step is executed after no more reduction operations can be applied. The graph G is partitioned into subgraphs, which are solved separately and recursively by using this 3-step procedure.

The authors proposed four different partition techniques:

- (a) The partition into MSC, already proposed in [80];
- (b) The partition into two independent subgraphs. The *locality* of the circuit is such that the circuit can be divided into small subcircuits with limited communication among them;
- (c) An *extended* partition which uses *loop* locality. A digital circuit generally contains many non-nested loops. This property causes the generation of two-edged loops after reduction procedures are performed on the corresponding topological graph;

- (d) Vertex elimination following a Branch-and-Bound procedure is applied when no other partition are possible.

All partitions above described can be performed in polynomial time by using variations of depth-first search and/or breadth-first search.

4. Merging: in this step the solutions computed in step 2 are merged.

The algorithm proposed by Orenstein et al. is efficient in random graphs, even if in cliques or graphs that are *almost* cliques it has an exponential behavior, since the reduction and partition techniques cannot be applied.

Somewhat surprising, the exact algorithm for feedback vertex set based on mathematical programming formulation is quite few. Recently, Funke and Reinelt [31] considered a special variant of feedback problems, namely the problem of finding a maximum weight node induced acyclic subdigraph. They discussed valid and facet defining inequalities for the associated polytope and developed a polyhedral-based exact algorithm presenting computational results obtained by applying a branch-and-cut algorithm.

3.5 Practical heuristics

Although the approximation algorithms guarantee a solution of a certain quality, for many practical real world cases, heuristic methods can lead to better solutions in a reasonable amount of CPU time. As Grötschel and Lovász [38] pointed out, fast construction heuristics combined with local improvement techniques tailored for special applications have been the “workhorse” of combinatorial optimization in practice. Over the years, a number of metaheuristics, including genetic algorithms, simulated annealing, greedy randomized adaptive search procedures (GRASP), Lagrangean relaxation, and others were developed with successful computational performance on a wide range of combinatorial optimization problems. Interestingly, however, feedback vertex set problems seem to be an exception. So far, relatively few practical heuristics have been developed for this family of problems, even fewer have reported computational results. Furthermore, most of the heuristics that seem to be quite successful computationally are greedy type heuristics or generalized greedy type heuristics (e.g. GRASP).

Almost all the efficient heuristics developed so far employ the solution-preserved reduction rules studied by Levy and Lowe [55], which were presented in Subsections 3.2 and 3.3.1. It has been observed in practice that this group of heuristics greatly reduces the cardinality of the graph not only

at the beginning of the algorithm, but also dynamically during the execution of node deletion type heuristics. A recent line of research on heuristic approaches is due to Pardalos, Qian, and Resende [68] where three variants of the so called *Greedy Randomized Adaptive Search Procedure* (GRASP) meta-heuristic are proposed for finding approximate solutions of large instances of the feedback vertex set problem in a digraph. GRASP is a multistart method characterized by two phases: a construction phase and a local search phase, also known as a local improvement phase. During the construction phase a feasible solution is iteratively constructed. One element at time is randomly chosen from a *Restricted Candidate List* (RCL), whose elements are sorted according to some greedy criterion, and is added to the building feedback vertex set and removed from the graph with all its incident arcs. Since the computed solution, in general, may not be locally optimal with respect to the adopted neighborhood definition, the local search phase tries to improve it. These two phases are iterated and the best solution found is kept as an approximation of the optimal solution.

To improve the efficiency of the method, Pardalos et al. incorporated in each iteration of their algorithm solution-preserving graph reduction techniques in their directed version and that can be used also to check if a digraph is acyclic, returning an empty reduced graph in case of positive answer.

The authors employed the following three greedy functions used to select the node with the maximum $G(i)$ values:

1. $G_A(i) = \text{in}(i) + \text{out}(i);$
2. $G_B(i) = \text{in}(i) * \text{out}(i);$
3. $G_C(i) = \max\{\text{in}(i), \text{out}(i)\}.$

Greedy function G_A assigns equal weight to in- and out-degrees. G_B favors the balance between in- and out-degrees. G_C only considers the largest value of the degrees. As demonstrated in Pardalos et al., G_B produced the best computational results. GRASP was tested on two randomly generated problem sets, finding the optimal solutions to all the problems in the first set, where the optimal values are known (computed by Funke and Reinelt [31]). Furthermore, this GRASP dominates the pure greedy heuristics in all the test instances with comparable running time. Fortran subroutines for finding approximate solutions of the directed feedback vertex set problem suing GRASP are given in Festa, Pardalos, and Resende [30].

4 The feedback arc set problem

Several versions of the feedback vertex set problem have been discussed in Section 3. It is possible to consider their *arc* counterparts. More specifically, given a graph $G = (V, E)$ and a nonnegative weight function $w : E(G) \rightarrow R^+$ defined on the arcs of G , find a minimum-weight subset of arcs $E' \subseteq E(G)$ that meets every cycle in a given collection C of cycles in (G, w) . As in the vertex case, this leads to the *minimum feedback arc set problem* (MWFAS) in both directed and undirected graphs, the *minimum weighted graph bipartization problem* via arc removals, and so on.

4.1 Mathematical model of the feedback arc set problem

Given an arc weighted graph (G, w) , $G = (V, E)$ and the set C of all cycles in G , the minimum weighted feedback arc set problem can be formulated as the following integer programming problem:

$$(MFAS) \quad \begin{cases} \min \sum_{e \in E(G)} w(e) x_e \\ \text{s.t. } \sum_{\substack{e \in \Gamma \\ e \in \Gamma}} x_e \geq 1 \quad \forall \Gamma \in C \\ x_e \in \{0, 1\} \quad \forall e \in E(G). \end{cases}$$

In its relaxation, the constraints $x_e \in \{0, 1\}, \forall e \in E(G)$ are replaced by $x_e \geq 0, \forall e \in E(G)$, obtaining a fractional feedback arc set.

As with the feedback vertex set problem, the feedback arc set problem is a *covering* problem and its (linear programming) dual is called a *packing* problem. In the case of the feedback arc set problem this means assigning a dual variable to all interesting cycles to be hit in the given graph, such that for each arc the sum of the variables corresponding to the interesting cycles passing through that arc is at most the weight of the arc itself.

4.2 Relation between the feedback vertex set and the feedback arc set problems

Feedback arc set problems tend to be easier than their vertex counterparts, especially for planar graphs. In the directed case feedback vertex and feedback arc set problems are each reducible to one another. In all reductions, there is a one-to-one correspondence between feasible solutions and their corresponding costs. Therefore, an approximate solution to one problem

can be translated to an approximate solution of the other problem reducible to this problem. Because most of the reduction procedures can be performed in linear time, these problems can be viewed as different representations of the same problem. Hence, as feedback vertex sets are reduced into feedback arc sets with the same weight and vice versa, all of these problems are equally hard to approximate.

Even, Naor, Schieber, and Sudan [28] showed how to perform reductions among feedback set problems and feedback subset problems and vice versa, preserving feasible solutions and their costs. In the following, we report some of these reduction procedures, where FVS and FAS denote the feedback vertex set problem and the feedback arc set problem, respectively, and BLACKOUT-FVS denotes an extension of the general feedback vertex set problem already referred to in the Subsection 3.3.1 (see also [2]). Recall that in the blackout feedback vertex set problem an additional subset of *blackout* vertices $B \subseteq V(G)$ is given and that the objective is to compute a minimum feedback vertex set that does not contain vertices of B .

[FAS \Rightarrow FVS] Given the graph $G = (V, E)$, construct its *directed line-graph* $G' = (V', E')$ as follows:

1. Set $V'(G') = E(G)$;
2. An arc in $E'(G')$ connects the vertices $(v_1 \rightarrow v_2) \in V'(G')$ and $(v_3 \rightarrow v_4) \in V'(G')$ if and only if $v_2 = v_3$.

In the weighted version, a correspondence among the weights of the arcs of G and the weights of the “vertices” of the corresponding graph G' is required as follows:

3. The weight of the “vertex” $(v_1 \rightarrow v_2) \in V'(G')$ is equal to the weight of the arc $(v_1 \rightarrow v_2) \in E(G)$.

A subset of arcs $F \subseteq E(G)$ is a feedback arc set for G if and only if it is a feedback vertex set for G' .

[FVS \Rightarrow FAS] Given the graph $G = (V, E)$, construct a graph $G' = (V', E')$ as follows:

1. For every $v \in V(G)$ insert in $V'(G')$ two vertices v_1 and v_2 ;

2. For every v_1 and v_2 inserted in $V'(G')$ after splitting a vertex $v \in V(G)$ insert in $E'(G')$ an arc $(v_1 \rightarrow v_2)$, all the arcs that enter v in G connecting them to v_1 in G' , and all the arcs that emanate from v in G emanating them from v_2 in G' .

In the case of weights we also have:

3. For every arc $e' = [(v_1 \rightarrow v_2) \mid v \in V(G)]$, set $w(e') = w(v)$. All other arcs in $E'(G')$ have infinite weight.

This establishes a one-to-one correspondence between the finite weighted feedback arc sets of the new graph G' and the feedback vertex sets of the original graph G .

[BLACKOUT-FVS \Rightarrow FVS] A very simple reduction procedure could consist of assigning infinite weight to each blackout vertex. However, in some cases a more formal reduction among these two similar problems could be needed. What can be done is to bypass each blackout vertex in B . For each blackout vertex $v \in B$:

1. Connect arcs between every two vertices that have a path of length two, connecting them, where v is the middle vertex;
2. Remove the blackout vertex v from the graph.

A subset of vertices $F \subseteq V(G)$ is a feedback vertex set that does not contain any blackout vertices from B if and only if it is a feedback vertex set of the graph obtained by the reduction.

Note that only the reduction **[BLACKOUT-FVS \Rightarrow FVS]** requires more than linear time to be performed.

4.3 State of the art of feedback arc set problems

In the literature of feedback set problems most of the proposed algorithms are designed to solve the problem in vertex-weighted graphs. One of the pioneering papers on feedback arc set problems is due to Ramachandran [73], where it is proved that finding a minimum feedback arc set in an arc-weighted reducible flow graph is as difficult as finding a minimum cut in a flow network. The proposed algorithm has complexity $O(m n^2 \log(\frac{n^2}{m}))$, where $m = |E(G)|$ and $n = |V(G)|$. The algorithm was adapted to solve the

problem in the vertex-weighted case. Shamir's linear time algorithm [76], used for the unit-weighted case, cannot be applied to solve the arc-weighted problem, because any reduction between arc and vertex set problems does not preserve the reducibility property.

Given a directed graph $G = (V, E)$, a *dijoin* $E' \subseteq E(G)$ is a set of arcs such that the graph $G' = (V, B)$, $B = E \cup \{(v, u) \mid (u, v) \in E'\}$ is strongly connected. Given nonnegative weights w_e , $e \in E(G)$, the minimum-weight dijoin problem is to find the dijoin with minimum weight. The feedback arc set problem in planar digraphs is reducible to the problem of finding a minimum-weight dijoin in the dual graph, which is solvable in polynomial time [37]. Stamm [83] proposed a simple 2-approximation algorithm for the minimum weight dijoin problem by superposing two arborescences. It is interesting to observe that, when translated to the dual graph, all these problems lead to problems of hitting certain cutsets of the dual graph, problems which can be approximated within a ratio of 2 by the primal-dual method. Goemans and Williamson [36] proposed a primal-dual algorithm that finds a $\frac{9}{4}$ -approximate solution to feedback sets problems in planar graphs.

The first approximation algorithm for the feedback arc set problem was given by Leighton and Rao [53]. Their approximation factor is $O(\log^2 n)$ in the unweighted case, where n is the number of vertices of the input graph. This bound was obtained by using a $O(\log n)$ approximation algorithm for a directed separator that splits the graph into two approximately equally-sized components. This separator can be found by approximating special cuts called *quotient cuts*. This result was improved by Seymour [75], who gave a $O(\log n \log \log n)$ -approximation algorithm that solves the linear relaxation of the feedback arc set mathematical model and then interprets the optimal fractional solution x^* as a length function defined on the arcs. Systematically, in a recursive fashion, it uses this length function to delete from the graph G all arcs between S and \bar{S} . Note that the linear program can be solved in polynomial time by using the ellipsoid or an interior point algorithm. Hence, the quality of the bound in this approach depends on the way the graph is partitioned. Seymour in [75] proved the following lemma:

Lemma 3 *For a given strongly connected digraph $G = (V, E)$, suppose there exists a feasible solution x to the feedback arc set problem. If ϕ is the value of the optimal fractional solution x^* , then there exists a partition (S, \bar{S}) such that, for some ϵ , $0 < \epsilon < 1$, the following conditions hold: If $\delta^+(S) = \{(u, v) \mid (u, v) \in E(G), u \in S, v \in \bar{S}\}$ and $\delta^-(S) = \{(v, u) \mid (v, u) \in E(G), u \in S, v \in \bar{S}\}$, then*

$$\sum_{e \in E(S)} w(e) x(e) \leq \epsilon \phi \quad (8)$$

$$\sum_{e \in E(\bar{S})} w(e) x(e) \leq (1 - \epsilon) \phi \quad (9)$$

and either

$$\sum_{e \in \delta^+(S)} w(e) \leq 20\epsilon\phi \log\left(\frac{1}{\epsilon}\right) \log \log \phi \quad (10)$$

or

$$\sum_{e \in \delta^-(S)} w(e) \leq 20\epsilon\phi \log\left(\frac{1}{\epsilon}\right) \log \log \phi. \quad (11)$$

Furthermore, the partition (S, \bar{S}) can be found in polynomial time.

The previous lemma admits a constructive proof as has been shown by Even, Naor, Schieber, and Sudan [28]. The algorithm proposed by Even et al. finds a feedback arc set having weight $O(\tau^* \log^2 |X|)$, where X is a special set of vertices defining the cycles to be hit and τ^* is the weight of an optimal fractional feedback set. They had the idea of reducing the problem to the directed minimum capacity multicut problem in circular networks and of adapting the undirected *sphere growing* technique described in [34] to directed circular networks. They decomposed the graph in the following way. A fractional and optimal solution to the directed feedback set problem induces a distance metric on the set of arcs (on the set of vertices) $E(G)$. Their approximation algorithm arbitrarily picks a vertex $v \in X$ and solves the shortest path tree problem rooted at v with respect to the metric induced by the fractional solution. The procedure that finds the shortest path tree defines layers with respect to the source v . Each layer is a directed cut that partitions the graph into two parts. The next step of the approximation algorithm is to choose a directed cut and to add the cut to the feedback set constructed so far. The algorithm continues recursively in each part and ends when the graph does not contain any interesting cycles. The key of the algorithm is the choice of the criterion to select the directed cut that

partitions the graph. Even et al. decided to relate the weight of the cut to the cost of the fractional solution.

More recently, Even, Naor, Schieber, and Zosin [27] showed that, for any weight function defined on the arcs, the subset feedback arc set problem can be approximated in polynomial time by a factor of two. The approximation algorithm consists of successive computations of minimum cuts. Its approximation factor is estimated by considering the capacities of minimum cuts as flow paths. When new minimum cuts are computed, previous flow paths are updated according to the decomposition of the graph induced by an optimal solution.

5 Applications

Feedback set problems were originally formulated in the area of combinatorial circuit design, where cycles can potentially cause a problem called a “racing condition”, where some circuit node may receive new inputs before it stabilizes [43]. To avoid such a condition, a (clocked) register is placed at each cycle in the circuit. However, the delay in the circuit speed is proportional to the number of registers placed along a path. Therefore, the objective is to minimize the number of nodes (registers) to be placed so that the total delay can be minimized.

Another application of the feedback set problem is in deadlock prevention in computer systems [59, 78]. Consider an operating system which schedules different processes with requests on different resources, which need to use exclusively before being released by the process. A directed graph modeling these resource requirements is to have a node i for each process i and a directed arc $e(i, j)$ denotes process i requests a resource already allocated to process j . Therefore, if there is a directed cycle in such a graph, a deadlock occurs and every process in the cycle will wait for the requested resource and never releases the resources already allocated to it. To break such cycles, one can remove some processes from the graph and put them in a waiting queue. It is clear that the objective is to minimize the number of removed processes.

In recent years, there have been intensive research efforts in the VLSI testing community on the feedback vertex set problem due to the following application [9, 49, 52]. A circuit can be modeled by a directed graph whose vertices represent gates computing Boolean functions and the directed arcs represent wires that connect gates. In this case, finding a minimum feed-

back arc set helps to reduce the hardware overhead required for testing the circuit by using scan-design techniques. One problem with this architecture is the cost of additional hardware. To reduce this overhead, partial scan methods have been proposed, which only scan a subset of flip-flops in the tested circuit. This way, the hardware overhead of the scan-design can be significantly reduced.

Luccio [61] describes applications of the feedback vertex set problem in synchronous systems [69, 70]. Typically, a synchronous system can be modeled by a network, whose vertices can be colored white or black and at each step a vertex changes its color according to the majority of its neighbors implying that it is receiving the correct information. In a “monotone” synchronous system only white vertices can change their color. It can be easily seen that in a toroidal mesh a feedback vertex set of black vertices causes all vertices to become black after some steps and that a minimum feedback vertex set is an initial configuration of minimum cardinality that stabilizes the system.

Two notable applications of the unweighted feedback vertex set problem in artificial intelligence are the *constraint satisfaction problem* and *Bayesian inference*. The application in the *constraint satisfaction problem* is due to Dechter [23] and Dechter and Pearl [22]. A set of variable $\{x_1, x_2, \dots, x_n\}$ is given, where each variable x_i belongs to a finite domain D_i . For every $i < j$ a constraint subset $R_{ij} \subseteq D_i \times D_j$ is constructed, defining pairs of values allowable for the pair of variables (x_i, x_j) . The objective is to find an n -tuple of values (v_1, v_2, \dots, v_n) to be assigned to $\{x_1, x_2, \dots, x_n\}$, such that all the constraints R_{ij} are satisfied. With each instance of the problem an undirected graph G can be associated whose vertices are the variables $\{x_1, x_2, \dots, x_n\}$ and whose arc set contains the arc (x_i, x_j) if and only if $R_{ij} \subset D_i \times D_j$. G is called a *constraint network* and represents a *constraint satisfaction problem*. It can be easily solved by applying a backtracking method, that repeatedly assigns values to the variables in an order previously defined and backtracks whenever reaching a dead end. This exponential method can be improved, leading to another exponential technique, but in the size of a feedback vertex set of the constraint network. It first finds a feedback vertex set of the constraint network and then arranges the variables so that variables in the feedback vertex set precede all other variables. The backtracking procedure is applied, and as soon as values of the variables in the feedback vertex set are determined, a polynomial time algorithm solves the constraint satisfaction problem in the remaining forest. In case of success a solution is found. Otherwise, a new backtracking phase occurs.

The application of the feedback vertex set problem to Bayesian inference is due to Bar-Yeruda, Geiger, Naor, and Roth [2]. They reduced the weighted loop cutset problem to the weighted blackout-feedback vertex set problem in a directed graph G and gave a $2\Delta^2(G)$ -approximation algorithm for solving any of these two problems, where $\Delta(G)$ is the degree of G . This algorithm reduced the computational complexity of Bayesian inference. Given a probability distribution $P(u_1, u_2, \dots, u_n)$, where u_i belongs to a finite domain D_i , a directed acyclic graph G is called a *Bayesian network* of P if there is a one-to-one correspondence between $\{u_1, u_2, \dots, u_n\}$ and $V(G)$ such that each u_i is associated with the vertex i for which the following holds:

$$P(u_1, u_2, \dots, u_n) = \prod_{i=1}^n P(u_i | u_{i_1}, u_{i_2}, \dots, u_{i_{j_i}}),$$

where i_1, i_2, \dots, i_{j_i} are the tail vertices of the edges in G whose head is i . The *updating problem* is that of finding for each value of i the probability $P(u_i | (v_1 = \mu_1), (v_2 = \mu_2), \dots, (v_l = \mu_l))$ given that the values $\{\mu_1, \mu_2, \dots, \mu_l\}$ are assigned to some variables $\{v_1, v_2, \dots, v_l\}$ among $\{u_1, u_2, \dots, u_n\}$.

Pearl [71] solved this problem as follows. A *trail* t in a Bayesian network G is a subgraph whose underlying graph is a simple path. A vertex b is a sink with respect to a trail t if there exist two consecutive arcs $(a \rightarrow b)$ and $(b \rightarrow c)$ on t . The arc t is said to be “active” by a set of vertices Z if every sink with respect to t either belongs to Z or has a descendant in Z , and every other vertex of t does not belong to Z . Pearl’s algorithm first selects a set of vertices S such that any pairs of vertices in G are connected by at most one active trail $t \in S \cup T$, where Z is any subset of vertices. Then, for each combination of value assignments to the variables belonging to S , a procedure given in [50] is applied. This solves the updating problem by viewing each vertex as a processor repeatedly sending messages to each of its adjacent vertices. When equilibrium is reached, each vertex i contains the conditional probability distribution $P(u_i | (v_1 = \mu_1), (v_2 = \mu_2), \dots, (v_l = \mu_l))$. At the end all the obtained results are combined. This technique, called the *conditioning method*, is exponential in the size of S . Bar-Yeruda et al. showed that S is a loop cutset of the Bayesian network and hence is computable by applying their algorithms.

The approximation algorithms for the weighted feedback vertex set problem have applications in areas of computer science other than areas of artificial intelligence. The leading inference Bayesian algorithm is the *clique tree* algorithm [51] and Shachter et al. [77] have shown that the weight of

the largest clique is bounded by the weight of the union of the loop cutset and the largest parent set of a vertex in a Bayesian network.

Even, Naor, Schieber, and Sudan [28] showed that on a special network called *circular network* any feedback vertex set problem is equivalent to another important NP-hard combinatorial optimization problem, called the *directed multicut problem*, defined by Hu [41] as follows. Given a capacitated network and a set of k source-sink pairs, find a minimum capacity set of edges whose removal disconnects all the source-sink pairs. The relation between feedback set problems and multicut problems was pointed out by Leighton and Rao [53]. Even et al. [28] described a simple procedure that reduces an instance of the feedback arc subset problem to an instance of the minimum multicut problem in circular networks, which are networks such that for each source-sink pair (s_i, t_i) an infinite capacity edge $t_i \rightarrow s_i$ is defined. The approximation algorithms for the feedback set problems proposed in [28], and discussed in Subsection 3.3.2, have been developed by the authors to improve the state of the art of ratios of approximation algorithms for the directed multicut problem.

In a very recent paper by Pachter and Kim [66], a connection between the feedback arc set problem and the *forcing problem* in square grids has been established. Given a graph G admitting a perfect matching M , the *forcing number* of M is the smallest number of arcs in a subset $S \subset M$, such that S is in no other perfect matching. A subset S having this property is said to force M . The concept of forcing arises in combinatorial chemistry [47, 48] leading to extensive study of forcing in hexagonal systems [15, 39, 56]. Pachter et al. [66] solved the forcing problem in special square grid graphs denoted by $R_n = P_{2n} \times P_{2n}$, where \times is the Cartesian graph product and P_{2n} is the path on $2n$ vertices. They used a result of Lucchesi and Younger [60] stating that, for a finite planar directed graph, a minimum feedback set has cardinality equal to that of a maximum disjoint collection of directed cycles. A directed graph G has the *cycle-packing property* if the maximum size of a collection of edge disjoint cycles equals the minimum size of a feedback set. This property still holds for an undirected graph if every orientation of the edges results in a directed graph with the cycle-packing property. Starting from a perfect matching M of a bipartite graph G having the cycle-packing property, Pachter et al. constructed a directed graph $D(M)$ having the same vertex set of G partitioned into two sets A and B and whose arc set contains an arc e directed from A to B if $e \in M$, from B to A otherwise. They proved that there is a one-to-one correspondence between alternating cycles in M and directed cycles in $D(M)$ and that for every feedback set in $D(M)$ there

is a forcing set in M of the same cardinality and vice versa.

Some recent papers due to Isaak [42] and Charon et al. [14] examined tournaments problems as a generalization of the feedback arc set problem for digraphs. A (round-robin) *tournament* $T = (X, U)$ is a complete asymmetric graph such that for every $x, y \in X$ there is a unique arc (x, y) or (y, x) . A weighted tournament is a tournament for which a weight function w is defined on U with values from the positive set of natural numbers. A classical NP-hard tournament problem is the following: Given a weighted tournament T , find a minimum weighted set of arcs of T such that reversing these arcs makes T transitive. Consider any digraph $D = (X, V)$ as tournament $T = (X, U)$, with $V \subset U$ weighted by w . The weights w are defined on U by $w(u) = 1$ if $u \in V$ and $w(u) = 0$ if $u \in U \setminus V$. Any optimal solution of the feedback arc set problem on D gives an optimal solution to the tournament problem on D and vice versa.

6 Future directions

Despite the large body of work on feedback vertex set, many interesting problems remain to be answered.

Recent advances in approximation algorithms seem to push the boundary closer to the limit. For undirected feedback vertex set, the worst case bound 2 cannot be improved unless the vertex cover problem can approximated within a bound less than 2, which has been conjectured by Hochbaum [40] to be NP-complete. The picture is still less clear for the directed case, where despite the arduous efforts of several researchers, the best known approximation bound is still $O(\log n \log \log n)$, and no approximation algorithm with constant ratio bound has been reported. Yannakakis [88] conjectured that there is a gap between the approximation of the directed and the undirected cases for feedback vertex set. This remains the biggest open question in the approximation of feedback vertex set problems.

Traditionally, the major tool used to attack feedback vertex set has been graph theory, whereas the application of mathematical programming is relatively limited. Recently, Goemans and Williams use a primal-dual formulation to model and establish uniform worst case bounds for a wide range of node-deletion problems. They established a bound of $9/4$. However, it is still inferior to the best known bound, and Goemans and Williamson posed the open question of whether the bound by a primal-dual method can match the best bound of 2 for undirected graphs. Funke and Reinelt [31] developed

a polyhedral based integer programming algorithm to exactly solve the feedback vertex set problem. Given the power of mathematical programming to other combinatorial optimization problems, it appears that much work needs to be done along this direction, both with regard to approximation and exact algorithms.

On the specially-structured polynomially solvable cases, Levy and Lowe's result [55] seems to push the study along this line to the limit, at least for flow type graphs. Recent applications in partial scan design report successful applications of the even somewhat brute force exact enumeration methods. All of these methods are used in conjunction with problem reduction techniques, which is quite unique to feedback vertex set. It would be interesting to further investigate the impact of these reduction techniques on the computational complexity of exact or approximate algorithms. In addition, it may also be possible to identify new classes of polynomially solvable feedback set problems (e.g. in VLSI design and circuit testing).

In contrast to the approximation literature, computational studies of feedback vertex set problems seem to be still in their embryonic stage. No modern metaheuristics, except the GRASP procedure recently developed by Pardalos, Qian, and Resende [68] have ever been applied to the feedback vertex set problem. The dimensions of the general problem that can be handled are still quite limited. It seems that this area of computational research has the greatest potential for progress and impact in the coming years. It is also worth noting that, since detecting cycles is a relatively expensive operation, the local search of feedback vertex set appears to be even more difficult than other notorious combinatorial problems like the traveling salesperson or set covering problems. The design of efficient local search procedures will also be a key to the effective computational procedure for feedback vertex set.

Acknowledgment

The research of Paola Festa was supported in part by the Italian Ministry of Scientific Research (MURST) - Research Project MOST 97. The research of Panos Pardalos was supported in part by DIMACS and by National Science Foundation Grants DMI-9622200 and BIR-9505919.

References

- [1] V. Bafna, P. Berman, and T. Fujito, Constant ratio approximations of the weighted feedback vertex set problem for undirected graphs, in *ISAAC95, Algorithms and Computation*, J. Staples, P. Eades, N. Katoh and A. Moffat Eds., Lecture Notes in Computer Science Vol.1004, Springer-Verlag (1995) pp. 142-151.
- [2] R. Bar-Yehuda, D. Geiger, J. Naor, and R.M. Roth, Approximation algorithms for the vertex feedback set problem with applications to constraint satisfaction and Bayesian inference, A preliminary version of this paper appeared in the *Proc. of the 5th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 344-354, (1994) and subsequently published in *SIAM J. Comput.* Vol.27 No.4 (1998) pp. 942-959.
- [3] A. Becker, and D. Geiger, Approximation algorithm for the loop cut-set problem, in *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence* Morgan Kaufman (1994) pp. 60-68.
- [4] A. Becker, and D. Geiger, Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem, *Artificial Intelligence* Vol.83 (1996) pp. 167-188.
- [5] J. A. Bondy, G. Hopkins, and W. Staton, Lower bounds for induced forests in cubic graphs, *Canad. Math. Bull.* Vol.30 (1987) pp. 193-199.
- [6] D.P. Bovet, S. de Agostino, and R. Petreschi, Parallelism and the feedback vertex set problem, *Information Processing Letters* Vol.28 (1988) pp.81-85.
- [7] A. Brandstädt, and D. Kratsch, On the restriction of some NP-complete graph problems to permutation graphs, in *Proc. of Fundamentals of Computing Theory*, Lecture Notes in Comp. Sci. Vol.199 (L. Budach, Ed. Springer-Verlag, Berlin, 1985) pp. 53-62.
- [8] A. Brandstädt, On improved time bounds for permutation graph problems, in *Proc. of the 18th Workshop on Graph-theoretic concepts in computer science*, (Wiesbaden-Naurod, 1992) Springer-Verlag LNCS 657 (1993) pp. 1-10.

- [9] M.A. Breuer, and R. Gupta, BALLAST: A methodology for partial scan design, in *Proc. of the 19th Int. Symposium on Fault-Tolerant Computing* (1989) pp. 118-125.
- [10] M. Cai, X. Deng, and W. Zang, A TDI system and its application to approximation algorithm, in *Proc. of the 39th Annual Symposium on Foundations of Computer Science*, Palo Alto, California, November 8-11, (1998).
- [11] M. Cai, X. Deng, and W. Zang, A min-max theorem on feedback vertex sets, to appear in *Integer Programming and Combinatorial Optimization: Proceedings 7th International IPCO Conference, Lecture Notes in Computer Science* Springer-Verlag (1999).
- [12] S. Chakradhar, A. Balakrishnan, and V. Agrawal, An exact algorithm for selecting partial scan flip-flops, Manuscript (1994).
- [13] M.S. Chang, Y.D. Liang, Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs, *Acta Informatica* Vol.34 (1997) pp. 337-346.
- [14] I. Charon, A. Guenoche, O. Hudry, and F. Wairgard, New results on the computation of median orders, *Discr. Math.* Vol.165/166 (1997) pp. 139-153.
- [15] R. Chen, X. Guo, and F. Zhang, The z -transformation graphs of perfect matchings of hexagonal system, *Discr. Math.* Vol.72 (1988) pp. 405-415.
- [16] K.T. Cheng, and V.D. Agrawal, A partial scan method for sequential circuits with feedback, *IEEE Transactions on Computers* Vol.39 No.4 (1990) pp. 544-548.
- [17] Chin Lung Lu, and Chuan Yi Tang, A linear-time algorithm for the weighted feedback vertex problem on interval graphs, *Information Processing Letters* Vol.61 (1997) pp. 107-111.
- [18] F.A. Chudak, M.X. Goemans, D. Hochbaum, and D.P. Williamson, A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs, *Operations Research Letters* Vol.22 (1998) pp.111-118. Vol.4 (1979) pp. 233-235.
- [19] V. Chvátal, A greedy heuristic for the set covering problem, *Mathematics Of Operations Research* Vol.4 (1979) pp. 233-235.

- [20] S.R. Coorg, and C.P. Rangan, Feedback vertex set on cocomparability graphs, *Networks* Vol.26 (1995) pp. 101-111.
- [21] D.G. Corneil, and J. Fonlupt, The complexity of generalized clique covering, *Discr. Appl. Math.* Vol.22 (1988) pp. 109-118.
- [22] R. Dechter, and J. Pearl, The cycle cutset method for improving search performance in AI, in *Proc. of the 3rd IEEE on AI Applications* Orlando FL (1987).
- [23] R. Dechter, Enhancement schemes for constraint processing: Back-jumping, learning, and cutset decomposition, *Artif. Intell.* Vol.41 (1990) pp. 273-312.
- [24] J. Donald, J. Elwin, R. Hager, and P. Salamon, A bad example for the minimum feedback vertex set problem, *IEEE Transactions on Circuits and Systems* Vol.32 (1995) pp. 491-493.
- [25] R.G. Downey, and M.R. Fellows, Fixed-parameter tractability and completeness I: Basic results, *SIAM Journal on Computing* Vol.24 (1995) pp. 873-921.
- [26] P. Erdős, and L. Posa, On the maximal number of disjoint circuits of a graph, *Pubbl. Math. Debrecen* Vol.9 (1962) pp. 3-12.
- [27] G. Even, S. Naor, B. Schieber, and L. Zosin, Approximating minimum subset feedback sets in undirected graphs, with applications, in the *Proc. of the 4th Israel Symposium on Theory of Computing and Systems* (1996) pp. 78-88.
- [28] G. Even, S. Naor, B. Schieber, and M. Sudan, Approximating minimum feedback sets and multicut in directed graphs, *Algorithmica* Vol.20 (1998) pp. 151-174.
- [29] G. Even, J.S. Naor, and L. Zosin An 8-Approximation Algorithm for the Subset Feedback Vertex Problem, 37th Symp. on Foundations of Comp. Sci. (FOCS) (1996) pp. 310-319.
- [30] P. Festa, P.M. Pardalos, and M.G.C. Resende, Fortran subroutines for approximate solution of feedback set problems using GRASP, Manuscript, AT&T Labs Research, Florham Park, NJ (1999).

- [31] M. Funke, and G. Reinelt, A polyhedral approach to the feedback vertex set problem, Manuscript (1996).
- [32] M.R. Garey, and D.S. Johnson, Computers And Intractability –A Guide to the Theory of NP-Completeness, *W. H. Freeman*, San Francisco, (1979).
- [33] M.R. Garey, and R.E. Tarjan, A linear-time algorithm for finding all feedback vertices, *Information Processing Letters* Vol.7 (1978) pp. 274-276.
- [34] N. Garg, V.V. Vazirani, and M. Yannakakis, Approximate max-flow min-(multi) cut theorems and their applications, *SIAM Journal on Computing* Vol.25 No.2 (1996) pp. 235-251.
- [35] F. Gavril, Some NP-complete problems on graphs, in *Proceedings of the 11th Conference on Information Science and Systems*, John Hopkins Univ. Baltimore, Md., (1977) pp. 91-95.
- [36] M.X. Goemans, and D.P. Williamson, Primal-dual approximation algorithms for feedback problems in planar graphs, in *Proceedings of the 5th MPS Conference on Integer Programming and Combinatorial Optimization (IPCO)* (1996) pp.147-161.
- [37] M. Grötschel, L. Lovász, and A. Schrijver, Geometric algorithms and combinatorial optimization, Springer-Verlag, Berlin (1988) pp. 253-254.
- [38] M. Grötschel, and L. Lovász, Combinatorial optimization: A survey, *DIMACS Technical Report 93-29* DIMACS Rutgers University (1993).
- [39] F. Harary, D.J. Klein, and T.P. Zivkovic, Graphical properties of polyhexes: Perfect matching vector and forcing, *J. Math. Chem.* Vol.6 (1991) pp. 295-306.
- [40] D. Hochbaum, Approximation algorithms for set covering and vertex cover problem, *SIAM Journal on Computing* Vol.11 No.3 (1982) pp. 555-556.
- [41] T.C. Hu, Multi-commodity network flows, *Operations Research* Vol.11 (1963) pp. 344-360.
- [42] G. Isaak, Tournaments as feedback arc sets, *Electronic Journal of Combinatorics* Vol.20 No.2 (1995) pp. 1-19.

- [43] D.B. Johnson, Finding all the elementary circuits of a directed graph, *SIAM J. Computing*, Vol. 4, No.1 (1975) pp. 77-84.
- [44] D.S. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer and System Science* Vol.9 (1974) pp. 256-278.
- [45] R.M. Karp, Reducibility among combinatorial problems, *Complexity Of Computer Computations* R.E. Miller and J.W. Thatcher, Eds., New York: Plenum Press (1972) pp. 85-103.
- [46] A.K. Kevorkian, General topological results on the construction of a minimum essential set of a directed graph, *IEEE Trans. Circuits and Systems* Vol.27 (1980) pp. 293-304.
- [47] D.J. Klein, and M. Randić, Innate degree of freedom of a graph, *J. Computat. Chem.* Vol.8 (1987) pp. 516-521.
- [48] D.J. Klein, T.P. Zivković, and R. Valenti, Topological long-range order for resonating-valance-bond structures, *Phys. Rev. B* Vol.43A (1991) pp. 723-727.
- [49] A. Kunzmann, and H.J. Wunderlich, An analytical approach to the partial scan problem, *J. of Electronic Testing: Theory and Applications* Vol.1 (1990) pp. 163-174.
- [50] H. Kim, and J. Perl, A computational model for combined causal and diagnostic reasoning in inference systems, in *Proc. of the 8th IJCAI*, Morgan-Kaufmann, San Mateo, CA, (1983) pp. 190-193.
- [51] S.L. Lauritzen, and D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems (with discussion), *J. Roy. Stat. Soc. Ser.B* Vol.50 (1988) pp. 157-224.
- [52] D. Lee, and S. Reedy, On determining scan flip-flops in partial scan designs, in *Proceedings Of International Conference on Computer Aided Design* (1990) pp. 322-325.
- [53] T. Leighton, and S. Rao, An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms, in *Proceedings of the 29th Annual Symposium on Fundations of Computer Science*, (1988) pp. 422-431.

- [54] A. Lempel, and I. Cederbaum, Minimum feedback arc and vertex sets of a directed graph, *IEEE Transactions on circuit theory* CT-13 (1966) pp. 399-403.
- [55] H. Levy, and L. Lowe, A contraction algorithm for finding small cycle cutsets, *Journal Of Algorithm* Vol.9 (1988) pp. 470-493.
- [56] X. Li, and F. Zhang, Hexagonal systems with forcing edges, *Discr. Math.* Vol.140 (1995) pp. 253-263.
- [57] Y.D. Liang, On the feedback vertex set problem in permutation graphs, *Information Processing Letters*, Vol. 52 (1994) pp. 123-129.
- [58] J. Liu, and C. Zhao, A new bound on the feedback vertex sets in cubic graphs, *Discrete Mathematics* Vol.148 (1996) pp. 119-131.
- [59] E.L. Lloyd, M.L. Soffa, and C.C. Wang, On locating minimum feedback vertex sets, *Journal of Computer and System Sciences* Vol.37 (1988) pp. 292-311.
- [60] C.L. Lucchesi, and D.H. Younger, A minimax theorem for directed graphs, *J. London Math. Soc.* Vol.17 (1978) pp. 369-374.
- [61] F.L. Luccio, Almost exact minimum feedback vertex set in meshes and butterflies, *Information Processing Letters* Vol.66 (1998, pp. 59-64.
- [62] C. Lund, and M. Yannakakis, On the hardness of approximating minimization problems, *Proceedings Of the 25th ACM Symp. On Theory Of Computing* (1993) pp. 286-293.
- [63] M.V. Marathe, C.Pandu Rangan, and R. Ravi, Efficient algorithms for generalized clique covering on interval graphs, *Discr. Appl. Math.* Vol.39 (1992) pp. 87-93.
- [64] B. Monien, and R. Schultz, Four approximation algorithms for the feedback vertex set problems, in *Proc. of the 7th Conference on Graph Theoretic Concepts of Computer Science*, Hanser-Verlag, München (1981) pp. 315-326.
- [65] T. Orenstein, Z. Kohavi, and I. Pomeranz, An optimal algorithm for cycle breaking in directed graphs, *J. of Electronic Testing: Theory and Applications* Vol.7 (1995) pp. 71-81.

- [66] L. Pachter, and P. Kim, Forcing matchings on square grids, *Discr. Math* Vol.190 (1998) pp. 287-294.
- [67] C. Papadimitriou, and M. Yannakakis, Optimization, approximation and complexity classes, in *Proc. of the 20th Annual ACM Symp. on Theory of Computing* (1988) pp. 251-277.
- [68] P.M. Pardalos, T. Qian, and M.G.C. Resende, A greedy randomized adaptive search procedure for feedback vertex set, *J. Comb. Opt.* Vol.2 (1999) pp.399-412.
- [69] D. Peleg, Local majority voting, small coalitions, and controlling monopolies in graphs: A review, in *Proc. of the 3rd Colloquium on Structural Information and Communication Complexity* (1996) pp. 152-169.
- [70] D. Peleg, Size bounds for dynamic monopolies, in *Proc. of the 4th Colloquium on Structural Information and Communication Complexity* (1997) Carleton Univ. Press, Ottawa, pp. 165-175.
- [71] J. Perl, Fusion, propagation and structuring in belief networks, *Artif. Intell.* Vol.29 (1986) pp. 241-288.
- [72] T. Qian, Y. Ye, and P.M. Pardalos, A Pseudo- ϵ approximation algorithm for feedback vertex set, *Recent Advances in Global Optimization*, Floudas, C.A. and Pardalos, P.M., Eds., Kluwer Academic Publishing (1995) pp. 341-351.
- [73] V. Ramachandran, Finding a minimum feedback arc set in reducible flow graphs, *Journal of Algorithms* Vol.9 (1988) pp. 299-313.
- [74] B. Rosen, Robust linear algorithms for cutsets, *Journal of Algorithms* Vol.3 (1982) pp. 205-217.
- [75] P.D. Seymour, Packing directed circuits fractionally, *Combinatorica* Vol.15 (1995) pp. 281-288.
- [76] A. Shamir, A linear time algorithm for finding minimum cutsets in reduced graphs, *SIAM Journal On Computing* Vol.8 No.4 (1979) pp. 645-655.
- [77] R.D. Shatcher, S.K. Andersen, and P. Szolovits, Global conditioning for probabilistic inference in belief networks, in *Proc of the 10th Conferences on Uncertainty in AI*, Seattle, WA (1994) pp. 514-522.

- [78] A.C. Shaw, The logical design of operating systems, Prentice-Hall, Englewood Cliffs, NJ, (1974).
- [79] D.A. Simovici, and G. Grigoras, Even initial feedback vertex set problem is NP-complete, *Information Processing Letters* Vol.8 (1979) pp. 64-66.
- [80] G.W. Smith, and R.B. Walford, The identification of a minimal feedback vertex set of a directed graph, *IEEE Transactions on Circuits and Systems* Vol.CAS-22 No.1, (1975) pp. 9-14.
- [81] E. Speckenmeyer, On feedback vertex sets and nonseparating independent sets in cubic graphs, *Journal of Graph Theory* Vol.12 (1988) pp. 405-412.
- [82] E. Speckenmeyer, On feedback problems in digraphs, in *Lecture Notes in Computer Science*, Springer-Verlag Vol.411 (1989) pp. 218-231.
- [83] H. Stamm, On feedback problems in a planar digraph, in R. Möhring ed. *Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science*, Springer-Verlag (1990) Vol. 484 pp. 79-89.
- [84] R.E. Tarjan, Depth first search and linear graph algorithms, *SIAM Journal on Computing* Vol.1 (1972) pp. 146-160.
- [85] S. Ueno, Y. Kajitani, and S. Gotoh, On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three, *Discrete Mathematics* Vol.72 (1988) pp. 355-360.
- [86] V. Vazirani, Approximation Algorithms, Manuscript, College of Computing, Georgia Institute of Technology.
- [87] C. Wang, E. Lloyd, and M. Soffa, Feedback vertex sets and cyclically reducible graphs, *Journal of the Association for Computing Machinery* Vol.32 No.2 (1985) pp. 296-313.
- [88] M. Yannakakis, Node and edge-deletion NP-complete problems, in *Proceedings of the 10th Annual ACM Symposium on Theory of Computing* (1978) pp. 253-264.
- [89] M. Yannakakis, and F. Gavril, The maximum k -colorable subgraph problem for chordal graphs, *Info. Process. Lett.* Vol.24 (1987) pp. 133-137.

- [90] M. Yannakakis, Some open problems in approximation, in *Proc. of the second Italian Conference on Algorithm and Complexity, CIAC'94* Italy, Feb. (1994) pp. 33-39.
- [91] D.H. Younger, Minimum feedback arc set for a directed graph, *IEEE Transactions on Circuit Theory* Vol. CT-10 (1963) pp. 238-245.
- [92] M. Zheng, and X. Lu, On the maximum induced forests of a connected cubic graph without triangles, *Discr. Math.* Vol.85 (1990) pp. 89-96.

Neural Networks Approaches for Combinatorial Optimization Problems

Theodore B. Trafalis
School of Industrial Engineering
University of Oklahoma
202 West Boyd, Room 124, Norman, OK 73019
E-mail: trafal@mailhost.ecn.ou.edu

Suat Kasap
School of Industrial Engineering
University of Oklahoma
202 West Boyd, Room 124, Norman, OK 73019
E-mail: suat@ou.edu

Contents

1	Introduction	260
2	Neural Networks	262
2.1	An Overview of Neural Networks	263
2.2	Neural Networks and Combinatorial Optimization Problems	264
3	Mean Field Theory	266
3.1	Spin-Glasses	266
3.2	Mean Field Approximation and Mean Field Equations	267
3.3	Mean Field Annealing	272
4	Combinatorial Optimization Problems	274
4.1	The Graph Bi-partitioning Problem	276
4.2	The Graph Partitioning Problem	277
4.3	The Traveling Salesman Problem	278
4.4	The Quadratic Assignment Problem	281
4.5	The Knapsack Problem	282
4.6	Other Combinatorial Optimization Problems	283

1 Introduction

Most of the engineering design problems and applications can be formulated as a nonlinear programming problem in which the objective function is nonlinear and has many local optima in its feasible region. It is desirable to find a local optimum that corresponds to the global optimum. The problem of finding the global optimum is known as the global optimization problem. Most such global optimization problems are difficult to solve. The main difficulties in finding the global optimum are that there are no operationally useful optimality conditions for identifying whether a point is indeed a global optimum, except in cases of special structured problems [33] and so it is computationally intensive to obtain the global optimum. Therefore, it is desirable and sometimes necessary to find a near global optimum in a reasonable time rather than obtaining the global optimum.

The solution methodologies to solve the global optimization problem are classified into two categories as deterministic and stochastic methods respectively. Deterministic methods [22, 33] attempt to create sequences of successive approximations that converge to points that satisfy the local optimality criteria. They are efficient when the starting point belongs to a close neighborhood of a global optimum. Stochastic methods [66] attempt to cover efficiently the entire space so that all the local optima will be identified, and hopefully one of them will be the global optimum. They try to identify either a collection of points or convergent sequences of points whose limits satisfy some property characterizing an optimum solution. The main difference between deterministic and stochastic methods is that the stochastic methods do not strictly improve the objective function. In general, stochastic techniques still do not guarantee an optimal solution. However, they often work exceptionally well in practice and have many advantages including flexibility, ability, and simplicity.

In this article one class of global optimization problems, specifically combinatorial optimization problems will mainly be discussed. The combinatorial optimization problems deal maximizing or minimizing an objective function subject to inequality and/or equality constraints over a set of com-

binatorial or discrete decision variables. A naive way to solve combinatorial optimization problems is to list all feasible solutions, then evaluate the objective function for each solution, and choose the best solution. However, even though it is possible in principle to solve the problem in this way, in practice it is not, because of the large number of feasible solutions to any problem of a reasonable size. Because of the combinatorial structure of these problems time needed to solve them grows exponentially with the size of the problem. Most of the combinatorial optimization problems are NP-complete [55]. For NP-complete problems, there is no algorithm that provides an exact solution in polynomial time.

Over the last three decades, the combinatorial optimization problems are one of the most challenging problems that have received considerable attention. The traditional solution methodologies for combinatorial optimization problems can be categorized into three groups as exact, heuristic, and approximation methods [78]. Exact methods such as simplex and branch and bound guarantee a global optimum, but the time needed to find it grows exponentially with the size of the problem. Heuristic methods such as SA, tabu search, and genetic algorithms are problem-specific based on the success without formal analysis of performance. On the other hand, approximation methods generate (near) global optimal solutions in a reasonable amount of time. NNs based methods are perceived as approximation methods.

In the last two decades, there has been significant interest in formulating combinatorial optimization problems in terms of statistical physics. This has led to the development of powerful optimization techniques such as neural networks (NNs), simulated annealing (SA), and mean field annealing (MFA) for combinatorial optimization problems. NNs have constituted a new direction in solving combinatorial optimization problems since the pioneering work of Hopfield and Tank [32]. The basic idea behind using NNs to solve an optimization problem is to map the optimization problem onto a highly interconnected network of neurons. A particular configuration of neurons being "on" or "off" yields an associated value of the network energy, which is the objective function of the optimization problem. The new problem is to find a configuration of neurons that minimizes the network energy that corresponds to an optimal solution for the optimization problem. Since SA was proposed in 1983 by Kirkpatrick et al. [38], it has been one of the most popular heuristic algorithms for finding (near) global optimal solutions of the combinatorial optimization problems. SA works by searching the set of all possible solutions, reducing the chance of getting stuck in a poor local optimum by accepting bad solutions with a decreasing probability. Exten-

sive review and list of applications of SA were given [13, 77]. By combining many characteristics of SA and NNs, another technique named as mean field annealing (MFA) based on the mean field theory (MFT) of statistical physics is proposed [60, 61] to solve optimization problems. MFA replaces the stochastic nature of SA with a set of deterministic equations named as mean field equations. The mean field equations depend on the energy function of the NN and are solved at each temperature during the annealing process of SA. It has been reported that MFA is much faster than SA while they have the same quality of solutions [5, 23, 60, 61]. MFA advances to the optimal solution in a fundamentally different way than stochastic methods. Stochastic methods search the solution space in random manner. On the other hand, MFA is ruled by a purely deterministic set of equations.

This article reviews the use of NNs for combinatorial optimization problems and provides a survey of most of the NN approaches that have been applied to combinatorial optimization problems. In Section 2, an overview of NNs is given and the mathematical basis of problem formulation for NNs based on an energy function is explained. In Section 3, spin-glass theory, mean field equations and MFA are presented. In Section 4, various combinatorial optimization problems and their corresponding NNs energy functions are studied. Finally, Section 5 is the conclusion.

2 Neural Networks

Modern era of NNs is said to have begun with the introductory work of McCullough and Pitts [49]. They have proposed a general theory of information processing based on networks of neurons. Each one of these neurons can only take the output values 1 or 0 by representing the active and resting states of neurons respectively. In general, NNs are dynamic systems built from a large number of interconnected units or variables that interact with each other in some prescribed way. NNs have come a long way from the early days of McCullough and Pitts. NNs have established themselves as an interdisciplinary subject with rich connections in the neuroscience, psychology, physical sciences, mathematics and engineering. Motivations for using NNs include the improvement in the speed of the operation through massively parallel computation and possible hardware design advantages.

2.1 An Overview of Neural Networks

NN models are algorithms for intellectual tasks such as learning and optimization that are based on the concept of how the human brain works. A NN model is composed of a large number of processing elements called *neurons*. Each neuron is connected to other neurons by links, each with an associated weight. Neurons without links toward them are called *input neurons* and with no link leaving away from them are called *output neurons*. The neurons are represented by state variables. State variables are functions of the weighted-sum of input variables and other state variables. Each neuron performs a simple transformation at the same time in a parallel-distributed manner. The input-output relation of the transformation in a neuron is characterized by an *activation function*. The following sigmoid function is the most used activation function in the NN literature

$$f(x) = \frac{1}{1 + e^{-\alpha x}}, \quad (1)$$

where α is a slope parameter. The combination of input neurons, output neurons, and links between neurons with associated weights constitute the architecture of the NN. Mathematically, a NN model is a directed graph with the following properties [27]:

1. Each neuron (node) is associated with a state variable S and an activation threshold (bias) v .
2. Each link (edge) between neurons is associated with it a weight w .
3. State of the neuron is determined by an activation function $f(S, w, v)$.

The classification of NNs can be done in different ways. NNs can be categorized as deterministic and stochastic according to the nature of activation. Deterministic NNs activate their states by using a deterministic activation function such as the sigmoid function. Stochastic NNs activate their states according to a probability distribution. A typical example of stochastic NNs is the Boltzmann Machine (BM). According to the nature of the connectivity, NNs can be categorized as feed-forward and recurrent NNs. If a directed graph has no closed paths, then it is called a feed-forward NN. Conversely, if a directed graph has closed paths, then it is called a recurrent NN. The popular multilayer perceptron and the Hopfield NNs are typical examples of the feed-forward and recurrent NNs, respectively.

2.2 Neural Networks and Combinatorial Optimization Problems

To solve combinatorial optimization problems using NNs requires a mapping of the problem onto the NNs in such a way that one can identify a solution from the outputs of neurons. The first step in designing NNs for combinatorial optimization problems is to formulate an energy function that maps the problem onto NNs. Indeed, the minimum of the energy function corresponds to the optimal solution of the combinatorial optimization problem. Most of the NN approaches formulate the energy function by incorporating the objective function and constraints of combinatorial optimization problems through functional transformation and numerical weighting. A functional transformation is usually used to convert constraints to a penalty function to penalize violations. Generally, constraint violations enter to the energy function in an explicit way. Numerical weighting is often used to balance constraint violations and the objective function. In general, such an energy function will have the following form:

$$E = \text{cost} + \sum_i A_i (\text{constraint violation})_i, \quad (2)$$

where A_i is a weight parameter and cost is the objective function that is independent from the constraint violations. By minimizing the energy function E , one attempts to minimize the cost while at the same time minimizes the constraint violations.

The second step in designing NNs for combinatorial optimization problems is to derive a dynamic equation (also known as state equation or motion equation). The dynamic equation prescribes the motion of the activation states of the NN. A properly derived dynamic equation can ensure that the state of the NN reaches equilibrium. The equilibrium-state of the NN satisfies the constraints and optimizes the objective function of the problem. Currently, the dynamic equations are derived by letting the time derivative of a state vector to be directly proportional to the negative gradient of an energy function.

The last step is to determine the architecture of the NN in terms of neurons and links between neurons with associated weights based on the dynamic equation in such a way that one can identify a solution from the outputs of the neurons. The success of optimization using NNs lies in the formulation of the energy function and the derivation of the dynamic equation. NNs do not guarantee globally optimal solutions but they compute locally optimal solutions.

NNs have been used to solve many combinatorial optimization problems since the pioneering work of Hopfield and Tank [32]. They have pioneered an approach for solving minimization problems by utilizing the collective computational capabilities of NNs. The objective function and constraints of the problem is mapped onto a quadratic energy function of neuron states. The energy function has the following form:

$$E(S) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N C_{ij} S_i S_j - \sum_{i=1}^N S_i I_i, \quad (3)$$

where $S_i = \{0, 1\}$ represents the output state of neurons, C_{ij} represents the weight of the link between neurons i and j and I_i represents the input bias; the input of neurons is denoted as x_i . The dynamic equations of neurons are defined by

$$\frac{dx_i}{dt} = -x_i + \sum_{j \neq i}^N C_{ij} S_i + I_i, \text{ where } S_i = f_i(x_i). \quad (4)$$

Hopfield and Tank showed that for a NN with symmetric connections and non-negative elements on the diagonal of the C matrix, will always converge by performing the gradient descent to a stable state, which is the local minimum of $E(S)$. The local minimum of $E(S)$ is obtained when the network is iterated from an initial state by updating each neuron asynchronously in accordance with the following updating rule:

$$S_i = \operatorname{sgn} \left(\sum_{j=1}^N C_{ij} S_j + I_i \right). \quad (5)$$

Hopfield and Tank [31] extended the discrete model to an analog model where $0 \leq S_i \leq 1$. The modified energy function is:

$$E(S) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N C_{ij} S_i S_j - \sum_{i=1}^N S_i I_i + \sum_{i=1}^N \frac{1}{R_i} \int_0^{S_i} g_i^{-1}(S) dS, \quad (6)$$

where R_i is the input resistance to unit i , and g_i is the activation function.

NNs to minimize the energy function are composed of either discrete or continuous neurons. The possibilities of using different types of NNs such as discrete-state and discrete-time, continuous-state and discrete-time, and continuous-state and continuous-time networks to solve combinatorial

optimization problems were investigated [50]. One usually prefers to use continuous neurons rather than discrete ones for two reasons. First, a continuous network tends to avoid oscillations between stable states. Second, solutions are much better than those provided by discrete NNs, because the valley of energy landscapes are wider and neuron outputs are not restricted to corners of the hypercube during convergence.

3 Mean Field Theory

NNs are similar to some models of magnetic materials namely the spin-glass models encountered in statistical physics [67]. In the last two decades, spin-glass theories of statistical physics have been extensively studied because of their application to other areas like optimization and NNs [6, 51, 52]. Little [46] pointed out a similarity between a NN and a system of elementary magnetic spins. Later, Hopfield [30] studied how such a NN or a spin system can store and retrieve information. He used the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks. Hopfield pointed out that the energy function describing a recurrent network with symmetric connectivity is identical to the energy function describing the dynamics of the spin-glass model.

3.1 Spin-Glasses

A spin-glass is a set of spins (i.e. magnetic moments) whose spins S_i and S_j interact through random couplings J_{ij} . In a magnetic model, if all J_{ij} are positive, this is called a ferromagnetic model. Conversely, if all J_{ij} are negative, this is an anti-ferromagnetic model, and if all J_{ij} are different in sign, this is a disordered magnetic model. A disordered magnetic model is called a *spin-glass* model. In order to produce such a model, two ingredients are necessary: frustration and disorder. Competing interactions between spins are in conflict with each other leading to frustration. Positions of the spins are random leading to disorder. Spins in a system interact with each other according to the following Hamiltonian:

$$H(S) = - \sum_{i=1}^N \sum_{j=1}^N J_{ij} S_i S_j. \quad (7)$$

There are number of problems outside physics which share some of the essential features - frustration and disorder - which characterize spin-glasses.

Insights and techniques can be borrowed from the spin-glass theory and brought usefully to other problems such as optimization and NNs. The spin-glass theory may be viewed as an optimization problem. Specifically, one wants to find the ground state configuration of spins that minimizes the Hamiltonian $H(S)$ in equation 7. Note that the minimization of this Hamiltonian is generally a NP-complete problem. To find the ground state configuration of spins, the mean field theory that make essential use of the tools of statistical physics have been used. The other area where the spin-glass theory has played an important role is NNs. The analogy to NNs was realized by identifying each spin with a neuron and associating +1 or -1 values of spins with active or resting state of neurons. The ground state configuration of spins corresponds to the stable state of NNs that minimizes the energy function of the network.

A Hopfield NN is a recurrent network whose dynamics are governed by a system of nonlinear ordinary differential equations defined in equation 4 and by an energy function defined as in equation 3. In physics, equation 3 is described as the Hamiltonian of N interacting spins as in equation 7. The NN input represents the field acting on a spin, the threshold act as the external field, and the connection or weight matrix corresponds to the random couplings for interaction between spins. Minimization of network energy function describes the dynamics of the spins aligning with their local fields.

There are two models of spin-glasses; specifically Ising-glass models and Potts-glass models. In Ising-glass models, the spin S_i can only take two values $S_i = \pm 1$. A Potts-glass model is a generalization of the two-state Ising-glass model to an arbitrary number N -states. Spin-glasses were studied extensively in [10, 21, 51]. Time dependent statistics of the Ising model were studied [26]. Some of the NP-complete problems like partitioning and coloring of random graphs were mapped onto the Potts-glass model [37]. An extensive review of Potts-glass models was given [79]. Potts-glass model of recurrent NNs was studied [70].

3.2 Mean Field Approximation and Mean Field Equations

The theory of spin-glasses relies strongly on the mean field theory (MFT). The MFT assumes that each spin moves in the mean field created on each spin by other spins fixed in their mean state. In other words, the value of S_i , the state of a spin at a particular spin site i , depends on its local environment because of interactions with other spins. Therefore, the value of

S_i is approximated by its mean value, $\langle S_i \rangle$ as the name of the theory implies. The $\langle S_i \rangle$ over a distribution of given parameters are expressed in terms of derivatives of an effective energy with respect to auxiliary fields according to the mean field approximation as explained later in this section. The mean field approximation is an analytic manipulation that simplifies the study of a physical system at the equilibrium. The behavior of the physical system at the equilibrium can be described by a set of equations named as mean field equations. Conceptually, the mean field approximation replaces the value of a spin state that occurs in an energy field by its mean value when evaluating the probability distribution of any other spins. The most probable state is the one with the lowest energy. Consequently, finding the most probable state is equivalent to minimizing the energy of the system. The probability of the system existing in any specific state $S = (S_1, S_2, \dots, S_N)$ is given by the following Boltzmann distribution

$$P(S) = \frac{e^{-E(S)/T}}{\sum_S e^{-E(S)/T}}. \quad (8)$$

where the term in the denominator is known as the partition function, Z . The partition function requires summing over all possible state configurations of the spin-glass $S = (S_1, S_2, \dots, S_N)$. Specifically,

$$Z = \sum_S e^{-E(S)/T} = \sum_{S_1=\pm 1} \sum_{S_2=\pm 1} \cdots \sum_{S_N=\pm 1} e^{-E(S)/T}. \quad (9)$$

Note that this partition function refers to the Ising model. Generally, computing the value of Z is not easy because of the combinatorial structure. If summations in equation 9 is given as integrals, the value of Z can be approximated by using saddle point approximation as is explained in the following.

Each discrete sum over $S_i = \pm 1$ in equation 9 can be replaced with an integral over a continuous variable V as,

$$\sum_{S_i=\pm 1} e^{-E(S)/T} = \sum_{S_i=\pm 1} \int_{-\infty}^{+\infty} e^{-E(V)/T} \delta(S - V) dV, \quad (10)$$

by using the sifting property of the delta function. The delta function can be written by introducing the variable U through the inverse Laplace transform

representation and making a simple shifting manipulation. Specifically,

$$\delta(S - V) = \frac{1}{2\pi i} \int_{-i\infty}^{+i\infty} e^{(S-V)U} dU. \quad (11)$$

By substituting equation 11 into equation 10, we obtain

$$\sum_{S_i=\pm 1} e^{-E(S)/T} = \sum_{S_i=\pm 1} \int_{-\infty}^{+\infty} e^{-E(V)/T} dV \frac{1}{2\pi i} \int_{-i\infty}^{+i\infty} e^{(S-V)U} dU \quad (12)$$

The summation over $S_i = \pm 1$ is eliminated since $\sum_{S_i=\pm 1} e^{SU} = 2 \cosh U$. Then

$$\sum_{S=\pm 1} e^{-E(S)/T} = \frac{1}{\pi i} \int_{-\infty}^{+\infty} \int_{-i\infty}^{+i\infty} e^{-E(V)/T} e^{-UV + \ln(\cosh U)} dU dV. \quad (13)$$

By using equation 13 in equation 9 to evaluate multiple summations, we obtain

$$Z = \sum_S e^{-E(S)/T} = \frac{1}{\pi i} \prod_{i=1}^N \int_{-\infty}^{+\infty} \int_{-i\infty}^{+i\infty} e^{-E_{eff}(V,U,T)/T} dU_i dV_i, \quad (14)$$

where the multiplication symbol denotes a multiple integral corresponding to multiple summations, and E_{eff} is the effective energy or free energy of the system, and it can be defined as follows,

$$E_{eff}(V, U, T) = E(V) + T \left(\sum_{j=1}^n U_j V_j - \ln(\cosh(U_j)) \right). \quad (15)$$

Notice that the evaluation of multiple integrals in equation 14 is not easy. The saddle point approximation [12] is used to evaluate these multiple integrals. By the saddle point approximation, the partition function is dominated by the value of $E_{eff}(V, U, T)$ at a saddle point. Therefore, we seek to determine the values of the saddle points of $E_{eff}(V, U, T)$. They are determined by using a gradient descent approach. Specifically,

$$\frac{\partial E_{eff}(V, U, T)}{\partial U_i} = V_i - \tanh U_i = 0 \quad (16)$$

and

$$\frac{\partial E_{eff}(V, U, T)}{\partial V_i} = \frac{\partial E(V)}{T \partial V_i} + U_i = 0. \quad (17)$$

The saddle point approximation to the partition function results into a set of equations, called mean field equations, whose solutions are the desired equilibrium states of the system. The general form of the mean field equations for the Ising model is

$$V_i = \tanh U_i \quad (18)$$

$$U_i = -\frac{\partial E(V)}{T \partial V_i} \quad (19)$$

Note that the discrete variables S_i have been replaced by continuous variables V_i and U_i , and V_i is the mean value of S_i . For the neural network with energy function given by equation 3, the mean field equations can be reduced to

$$V_i = \tanh \left(\frac{1}{T} \sum_{j=1}^N C_{ij} V_j \right), \quad (20)$$

where for simplicity, we have set $I_i = 0$. An iterative solution of equation 20 computes a new V^{k+1} in terms of previous V^k as follows

$$V_i^{k+1} = \tanh \left(\frac{1}{T} \sum_{j=1}^N C_{ij} V_j^k \right). \quad (21)$$

Each spin of the Ising model has only two states either -1 or $+1$. This makes the Ising model inconvenient to implement for optimization problems with multi-state variables. An alternative approach, proposed by Peterson and Soderberg [61], is to use the Potts model. Instead of allowing each spin to be -1 or $+1$ independently, spins are arranged into sites, and only one spin within each site is allowed to be $+1$. Introducing a second index for the spin S_{ij} , the first index denotes the cite i and the second index denotes the spin j within each cite such that

$$\sum_j S_{ij} = 1. \quad (22)$$

In the Potts model, two possible values of each spin is changed to 0 or 1 rather than -1 or $+1$. Thus, for every i , S_{ij} has only one value for j and zero for the remaining values of j . Equation 22 guarantees that each site parameter takes only one value. Note that, by using the Potts model,

some of the soft constraints of the problem are treated as hard constraints namely they hold automatically. This makes the solution space smaller than the Ising model.

The variables in the mean field equations described in equations 18 and 19 have an extra dimension added to each one of them. Each S_{ij} is like an element of a matrix so that a vector S_i has j elements such that each element is either 0 or 1 and the sum of each element is equal to 1. Therefore, mean field equations for the Potts model will be different than the Ising model. The derivation of mean field equations for the Potts model is similar to Ising model. The only difference comes from the definition of the partition function. The summation in the partition function for the Potts model is now constrained to run over all possible system states S satisfying equation 22. For example, $S_i = (1, 0, \dots, 0), (0, 1, \dots, 0), \dots, (0, 0, \dots, 1)$.

The effective energy for the Potts model will be as follows

$$E_{eff} = E(V) + T \left(\sum_{j=1}^n U_j V_j - \ln \sum_S e^{US} \right). \quad (23)$$

The saddle points of $E_{eff}(V, U, T)$ are determined again by using a gradient descent approach. The general form of mean field equations for Potts model is

$$V_j = \frac{\sum_S S e^{US}}{\sum_S e^{US}} \quad (24)$$

$$U_j = -\frac{\partial E(V)}{T \partial V_j}. \quad (25)$$

So far, the saddle points of $E_{eff}(V, U, T)$ are obtained by using a gradient descent approach. Different approaches to find them have been studied. A new approach to find the saddle points of $E_{eff}(V, U, T)$ was presented [84] by proposing the use of an EM (Expectation-Maximization) algorithm. The EM algorithm assumes there are two types of parameters. An E-step estimates the first type of parameters with the fixed second type of parameters. An M-step maximizes to find the second type of parameters with the fixed first type of parameters. The E-step and the M-step alternate until convergence. It was reported that the EM algorithm was effective only when used in conjunction with the annealing [84]. Note that for the gradient descent approach parameters are estimated together, while EM they are estimated in turn. Another approach to find the saddle points of $E_{eff}(V, U, T)$ was

studied [15] by introducing an ascent-descent dynamic system. Two continuous time dynamic systems namely minimax dynamics and descent dynamic were suggested to find the saddle points. The proposed algorithms take into accounts the saddle point structure of the energy function.

Mostly, the mean field equations are solved iteratively as shown in equation 21. Different approaches to solve mean field equations have been studied. A new approach to solve mean field equations namely a temperature tracking algorithm is proposed [82, 83] for problems with a single minima. The temperature tracking algorithm was a version of deterministic annealing by writing down a differential equation to track the unique energy minimum as a function of temperature. For the linear programming problem, it was shown that the temperature tracking algorithm was equivalent to Faybusovich's [20] interior point algorithm. Another approach to solve mean field equations was presented [74, 75] by using gradient descent differential systems whose trajectory was confined within the solution space of the problem. The gradient descent differential system is proven theoretically to always produce a local optimum solution, and the stability of the presented system is analyzed qualitatively through eigenvalue analysis.

Finally, notice that for both models, the effective energy $E_{eff}(V, U, T)$ is smoother than $E(S)$ because of the additional term known as entropy of the system. As a result of this, the probability of being trapped in a local minimum is reduced.

3.3 Mean Field Annealing

The iterative approach as shown in equation 5 guarantees to convergence only to the local minima and only for the case that the starting point is sufficiently close to a global minimum. One way of avoiding this dependency is to use SA. However, in many cases, SA is so time consuming. Another approach is to use MFA. MFA is a variant of SA, and replaces the stochastic nature of SA with a set of deterministic mean field equations that need to be solved iteratively. This deterministic relaxation procedure exhibits fast convergence toward the solution of the optimization problems.

The effective energy function in equation 15 or 23 has the following general structure

$$E_{eff}(V) = E(V) + TF(V), \quad (26)$$

where T and F correspond to the temperature and entropy of the system, respectively. The effective energy decreases as the temperature decreases. At the high temperature, the effective energy is dominated by the convex

entropy term. Therefore the effective energy is a strictly convex, and it has a unique minimum. On the other hand, at the low temperature, the effective energy is nearly equal to the energy of the system. In order to get a good local minimum (near global minimum), MFA is used as follows. First, the mean field equations are solved at high temperature and a unique solution is obtained. Then, after slightly lowering the temperature, the mean field equations are solved again starting from the higher temperature solution. By continuing this procedure, one can get a lower temperature solution that corresponds to a near global minimum of the energy function. By minimizing the energy function of the network using MFA and an appropriate cooling schedule, it is possible to escape unwanted local minima.

MFA algorithm for a minimization problem can be stated as follows:

Step 0: SELECT randomly a new solution, S^0 .

SET starting temperature, T^0 , and iteration counter, $k = 0$.

COMPUTE $E(S^0)$.

Step 1: ITERATION:

REPEAT

REPEAT

UPDATE iteration counter, $k = k + 1$

SOLVE mean field equations

UNTIL saturation of mean field variables

UPDATE Cooling Schedule, $T^k = \alpha T^{k-1}$

UNTIL system freezes

The spin-glass models have received a lot of attention in statistical physics because of their success in explaining systems that pass through a continuity of intermediate states as temperature changes, but suddenly exhibit a discontinuity in some temperature corresponding to a phase transition. The temperature associated with the phase transition is described as the critical temperature of the system. At the critical temperature, the continuous variables start converging to the limits. Likewise, starting the annealing right at the critical temperature causes the system to reach equilibrium without the cooling while starting at very high temperature requires unnecessary computations. As one sees that, the cooling time can be reduced extremely by choosing an initial temperature that is close enough to the critical temperature. Thus, the knowledge of the critical temperature

is remarkably useful for an efficient implementation of the annealing. For some cases, it is possible to compute the critical temperature analytically. The critical temperature is determined by the spectral radius of the Hessian of the energy function. Unfortunately, the dimension of the Hessian of the energy function becomes very large for large-size problems, which can make the computation of eigenvalues expensive. An approximate expression for the critical temperature using the energy function is described [60] by estimating the eigenvalue distribution of linearized mean field equations. That approximation gave very good results. A parallel version of MFA and a new temperature scheduling method named as maximum entropy cooling schedule were proposed [64].

During the annealing process, a sequence of bifurcation for minimum solution occurs. The structure of the bifurcation affects the quality of the annealing solution. Bifurcation structures in MFA was studied [69, 36]. It was shown that the annealing solution in MFA was not unique in general and it was not always an optimal solution. A chaotic Potts spin model based on a Euler difference equation of the Potts model with inhibitory self-loops was proposed [36].

4 Combinatorial Optimization Problems

NNs have very close ties with optimization, and the ties are manifested mainly into two aspects. On one aspect, learning algorithms have been developed based on optimization techniques to train a NN to perform modeling tasks [2, 59, 68, 80]. On the other aspect, NNs have been developed to solve difficult optimization problems like combinatorial optimization problems that are discussed in this section. Many difficult optimization problems can be formulated as the minimization of a quadratic form and thus can be mapped onto NNs. One of the main advantages of NN approaches to classical optimization approaches is the inherently parallel and distributed nature of the dynamic solution procedure. Therefore, NNs are capable to solve large-scale optimization problems in real time [3, 5, 42, 43, 44, 76].

NNs have constituted a new direction in solving combinatorial optimization problems since the pioneering work of Hopfield and Tank [32]. They showed that an energy function can be defined for an analog NN, and the NN always converges by performing the gradient descent to a stable state, which is the local minimum of the energy function. To find the (near) global minimum of the energy function, the Boltzmann Machine (BM) was

proposed [2]. The BM was the first attempt to combine SA and NNs to solve combinatorial optimization problems. However, BM is designed for discrete variables with the disadvantage of being slow. Another approach combining SA and NNs was proposed [45] by adding a noise term to neural dynamics. It was shown that the intensity of the noise term depended on the state of the neuron as well as on a temperature parameter T . By selecting an appropriate temperature schedule $T(t)$, the resulting NN converged to a near global minimum of the combinatorial optimization problem.

A new method to solve combinatorial optimization problems using the MFT and NNs was introduced by Peterson and Anderson [60]. That was the first detailed attempt to use the MFT and NNs for solving combinatorial optimization problems after a brief description by Hopfield and Tank. A problem was mapped onto a NN by using the Ising model and then MFA was used in order to escape from local minima. As an extension of this study, a new method based on the Potts model was presented by Peterson and Soderberg [61]. They reduced the dimension of the solution space one dimension by using Potts model instead Ising model. It was stated that the solution quality and parameter insensitivity were advantages of the new method. A general framework of MFA as an extension of SA from its original formulation as a Markov process was derived, and its relationship with to Hopfield NNs was shown [5].

The mathematical basis of the behavior of the Hopfield NN and the relationship between the Hopfield NN and combinatorial optimization problems were presented [72]. It was shown that the Hopfield NN could be generalized so that it might be applied any objective function. The relationship between MFT models and the Hopfield model was analyzed mathematically by using the theory of dynamical systems [40]. It was proved that the set of asymptotically stable fixed points (or equilibrium) of the asynchronous MFT model coincided with the set of asymptotically stable fixed points of the Hopfield model, and the dynamics of the asynchronous MFT model was equivalent to the Hopfield model. As a result, it was stated that the MFT model is suitable for solving combinatorial optimization problems in place of the Hopfield model.

A systematic approach to design competition based NNs for the combinatorial optimization was presented [17]. The competition based NNs were studied in detail [19]. NNs for combinatorial optimization problems were reviewed in [11, 17, 28, 47, 50, 65, 72, 78].

The following combinatorial optimization problems have been studied using MFT and NNs approaches.

4.1 The Graph Bi-partitioning Problem

The graph bi-partitioning or bisection problem can be defined as follows: Given a set of N nodes with a given connectivity, partition them into 2 sets each with $N/2$ nodes such that the net connectivity (cut-size) is minimal between each set. The graph bi-partitioning problem was studied in [5, 11, 57, 58, 60, 61, 62, 63, 65].

The problem can be mapped onto a Hopfield NN by the following representation. For each vertex i , a binary unit $s_i = +1$ or -1 is assigned, and for each pair of vertices $s_i, s_j, i \neq j$, a value $T_{ij} = 1$ if they are connected, and $T_{ij} = 0$ if they are not connected is assigned. An energy function for this problem can be written as follows [57, 58, 60]:

$$E(s) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N T_{ij} s_i s_j + \frac{\alpha}{2} \left(\sum_{i=1}^N s_i \right)^2 \quad (27)$$

where α is an imbalance parameter. The first term corresponds to the cost function and the second term corresponds to the constraints of the problem that guarantee equal partition. $T_{ij} s_i s_j$ is 0 whenever vertices i and j are not connected at all, positive whenever connected vertices i and j are in the same partition and negative when they are in separate partition.

The mean field equations for this energy function by using the Ising model can be written as follows [57, 58, 60]:

$$V_i = \tanh \left(\sum_{j=1}^N (T_{ij} - \alpha) \frac{V_j}{T} \right) \quad (28)$$

The graph bi-partitioning problem was mapped onto a NN such that a neuron being "on" corresponded to a certain decision by using the Ising model, and then MFA was used in order to escape from local minima [60]. The issues of the solution quality, programming complexity, convergence times and scalability were addressed, and the results were very encouraging. As an extension of this study, a new method that maps the graph bi-partitioning problem onto NNs by using the Potts model was presented [61]. Numerical studies on the graph bi-partitioning problems were performed by comparing the new method with SA. Very good quality solutions were consistently found by using the new method. The behavior of MFA was examined both analytically and experimentally for the graph bi-partitioning problem [5]. The results showed that MFA found same quality solution as SA in a much faster time.

4.2 The Graph Partitioning Problem

The graph partitioning (GP) problem can be defined as follows: Given a set of N nodes with a given connectivity, partition them into K partitions each with N/K nodes such that the net connectivity (cut-size) is minimal between each set. Note that, if $K = 2$, then this problem becomes graph bisection problem. The GP problem was studied in [11, 28, 50, 57, 61, 62, 63, 65, 76].

In order to map the problem onto a NN, second index for the neuron, s_{ia} , where the first index i denotes the node $i = 1, \dots, N$ and the second index a denotes the partition $a = 1, \dots, K$ is introduced. An energy function for this problem can be written as follows [57, 61, 63]:

$$E(s) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N T_{ij} s_i s_j + \frac{\alpha}{2} \left(\sum_{i=1}^N s_i \right)^2 - \frac{\beta}{2} \sum_{i=1}^N s_i^2 \quad (29)$$

where α and β are imbalance parameters and $s_i = (s_{i1}, s_{i2}, \dots, s_{iK})$ in vector notation. The first term corresponds to the cost function and second and third terms correspond to the constraints of the problem that guarantee equal partition into K subsets.

The mean field equations for this energy function by using the Potts model can be written as follows [57, 61, 63]:

$$V_i = \frac{\sum_S S e^{U_i S}}{\sum_S e^{U_i S}} \quad (30)$$

$$U_i = \frac{1}{T} \left(\sum_{j=1}^N (T_{ij} - \alpha) V_j + \beta V_i \right) \quad (31)$$

A new method that maps the GP problem onto NNs by using the Potts model by allowing only one state at every node to be "on" was presented [61]. Numerical studies on the GP problems were performed by comparing the new method with SA. Very good quality solutions were consistently found by using the new method. It was stated that the solution quality and parameter insensitivity were advantages of the new method. The concept of MFA in the context of the graph partitioning problem was presented [76]. It was indicated that the results obtained by MFA was comparable to the results obtained by SA or the Kerninghan-Lin algorithm, but the rate of convergence of MFA was much faster than that of SA.

4.3 The Traveling Salesman Problem

The traveling salesman problem (TSP) can be defined as follows: Given a list of n cities and the distances, d_{ij} , among them, find the shortest possible tour through a set of n cities, visiting each one exactly once. Note that the TSP is a special case of the GP problem where $K = N$. For the n -city TSP, the number of possible tours are $n!$. However, a tour describes an order in which cities are visited. For the n -city TSP, there are $2n$ tours of equal path-length. Therefore, there are $2!/2n$ distinct paths for closed TSP tours. The total of $N = n^2$ neurons required to map the n -city TSP to NNs. TSP is the most studied problem of combinatorial optimization problems by using NN approaches. The TSP was studied in [11, 32, 34, 35, 36, 47, 48, 50, 56, 57, 61, 62, 63, 64, 71, 78, 81].

Hopfield and Tank [32] formulated the traveling salesman problem (TSP) on a highly interconnected NN and made exploratory numerical studies on a 10-city and 30-city TSP. They showed that an energy function can be defined for an analog NN and the NN always converges by performing the gradient descent to a stable state, which is the local minimum of the energy function. They used the following energy function to map the TSP problem to the NNs.

$$\begin{aligned} E(S) = & \frac{A}{2} \sum_{X}^N \sum_{i}^N \sum_{j \neq i}^N S_{Xi} S_{Xj} + \frac{B}{2} \sum_{i}^N \sum_{X}^N \sum_{X \neq Y}^N S_{Xi} S_{Yi} + \\ & \frac{C}{2} \left(\sum_{X}^N \sum_{i}^N S_{Xi} - N \right)^2 + \frac{D}{2} \sum_{X}^N \sum_{X \neq Y}^N \sum_{i}^N d_{XY} S_{Xi} (S_{Y,i+1} + S_{Y,i-1}). \quad (32) \end{aligned}$$

S_{ij} indicates whether city i is assigned to position j in the tour or not. The first three terms characterize the general problem constraints. The first triple sum is zero if and only if each city row X contains no more than "1", (the rest of the entries being zero). The second triple sum is zero if and only if each "position in tour" column contains no more than "1", (the rest of the entries being zero). The third triple sum is zero if and only if there are n entries of "1" in the entire matrix. The last triple sum is the cost term, the length of the path corresponding to a given tour.

A new method that maps the TSP problem onto NNs by using the Potts model by allowing only one state at every node to be "on" was presented [61]. Numerical studies on the TSP problems by comparing with SA were performed. Very good quality solutions were consistently found by using

the new method. It was stated that the solution quality and parameter insensitivity were advantages of the new method.

An energy function for this problem by using the Potts model can be written as follows [56, 57, 61, 62]:

$$E(S) = \frac{A}{2} \sum_i \sum_j \sum_a D_{ij} S_{ia} S_{j(a+1)} - \frac{B}{2} \sum_i \sum_a S_{ia}^2 + \frac{C}{2} \sum_a \left(\sum_i S_{ia} \right)^2 \quad (33)$$

where A , B , and C are the parameters, D_{ij} is the distance between cities i and j , and $a+1$ is defined Modulo N. The first term minimizes the distance, and the second and third terms ensure that each city is visited exactly once. To determine the appropriate values of A , B , and C parameters, a study was performed in [34, 35], and a method was proposed based on "two-layer random field model" using MFA. The method reduced the computation time. In another study, two NN mappings of TSP with absolute distances, D_{ij} and relative distances, $D_{ij}' = D_{ij} - d$ where d is the average distance between every two cities were introduced [48]. Their theoretical comparisons were made to prove the superiority of relative distances over absolute distances using the stability theory.

The mean field equations for this energy function by using the Potts model can be written as follows [56, 57, 61]:

$$V_{ia} = \frac{e^{U_{ia}}}{\sum_b e^{U_{ib}}} \quad (34)$$

$$U_{ia} = \frac{1}{T} \left(- \sum_j D_{ij} (V_{j(a+1)} + V_{j(a-1)}) - \alpha \sum_j V_{ja} + \beta V_{ia} \right) \quad (35)$$

An alternative approach based on the elastic net to solve TSP was developed [14]. Basically an elastic rubber band by mapping from a plane, x to a circle, y such that each city, i on the plane was mapped onto a point, a on the circle is allowed to expand to touch all cities. Note that the number of points on the circle, M can in principle be larger than the number of cities, N . Let y_a denotes the M coordinates at the circle and x_i denotes the N city coordinates. The elastic net algorithm works starting with a small radius circle containing M y_a coordinates with an origin slightly displaced at the center of gravity for the N cities. Then, by changing the values of the y_a ,

the following energy function was minimized [56, 81].

$$E = -\alpha K \sum_i^N \log \sum_a^M e^{-|x_i - y_a|^2 / 2K^2} + \beta \sum_a^M |y_{a+1} - y_a|^2 \quad (36)$$

It was demonstrated that, there was a strong correspondence between the elastic net approach and MFA based on the Potts model, and the elastic net could be derived from a statistical physics system as a saddle point approximation as follows [56, 62, 63, 81].

N city positions in a TSP are denoted by x_i . For each city i , a Potts spin s_{ia} is defined to be 1 if a is matched to city i and 0 otherwise. By changing the values of the y_a and s_{ia} , the following energy function is minimized [62, 63, 81].

$$E(s_{ia}, y_a) = \frac{1}{2T} \sum_i^N \sum_a^M s_{ia} |x_i - y_a|^2 + \frac{\gamma}{2} \sum_a^M |y_{a+1} - y_a|^2 \quad (37)$$

The first term ensures matching of two coordinates. If $x_i = y_a$ for which $s_{ia} = 1$, it is minimized. The second term minimizes the traveling distance. The derivation of mean field equations are given in [62, 63, 81]. During the derivation of mean field equations the following effective energy will be obtained.

$$E_{eff}(y_a) = -T \sum_i^N \log \sum_a^M e^{-|x_i - y_a|^2 / 2T^2} + \frac{\gamma}{2} \sum_a^M |y_{a+1} - y_a|^2 \quad (38)$$

Note that this effective energy is exactly same as the energy function in equation 36 for elastic net. A proof of the connection between the Hopfield and Tank model and elastic net model energy functions by showing how they can be obtained as special cases of a more general energy function based on generalized deformable models with binary matching fields was provided in [81]. A summary of the results from benchmark studies on TSP by using so timeMFA, elastic net, genetic algorithm, SA, and hybrid approaches is given in [56].

A parallel version of MFA and a new temperature scheduling method named as maximum entropy cooling schedule were applied to the TSP and results showed that the parallel method required a shorter time to obtain the same result that the serial method [64]. The MFT approximation was combined with the notions from the area of constrained optimization specifically Lagrange multipliers and adaptive SA, and the resulting method is to TSP

[71]. The results of numerical experiments on TSP showed both increased computational efficiency and better solutions. Bifurcation process for MFA applied to TSP was investigated [69]. Some of bifurcation properties of MFA based on the Potts model applied to TSP was showed [36].

4.4 The Quadratic Assignment Problem

The generalized quadratic assignment problem (QAP) represents a large class of combinatorial optimization problems arising in a variety of planning and designing concepts. The generalized QAP can be defined as minimizing a quadratic cost function for assignment of a number of M objects to N positions where $N \geq M$. Namely,

$$\begin{aligned} \min \quad & \sum_{i=1}^N \sum_{j \neq i} \sum_{k=1}^M \sum_{l \neq k} c_{ij} d_{kl} s_{ik} s_{jl} \\ \text{subject to} \quad & \sum_{i=1}^N s_{ik} = 1, \quad k = 1, \dots, M \\ & \sum_{k=1}^M s_{ik} \leq 1, \quad i = 1, \dots, N \\ & x_{ik}, x_{jl} \in \{0, 1\} \end{aligned} \quad (39)$$

where c_{ij} denotes the cost associated with assigning one unit of goods from position i to position j and d_{kl} denotes the cost associated with assigning one unit of goods from object k to object l . The QAP was studied in [11, 17, 18, 50, 78].

An energy function for this problem can be written as follows [17, 18]:

$$E = \frac{A}{2} \sum_{i=1}^N \sum_{j \neq i} \sum_{k=1}^M \sum_{l \neq k} c_{ij} d_{kl} s_{ik} s_{jl} + \frac{B}{2} \sum_{i=1}^N \sum_{k=1}^M \sum_{l \neq k} s_{ik} s_{il} + \frac{C}{2} \sum_{k=1}^M \left(1 - \sum_{i=1}^N s_{ik} \right)^2 \quad (40)$$

where A , B , and C are the weight parameters. The first term corresponds to the objective function. The second term specifies the constraint that at most one position can be assigned to each object and the third term specifies the constraint that every object must be assigned by exactly one assignment. This term prevents the solution of no assignments.

The mean field equations for this energy function can be written as

follows:

$$V_{ik} = \tanh \left(-\frac{A}{T} \sum_{j \neq i} \sum_{l \neq k} c_{ij} d_{kl} V_{jl} - \frac{B}{T} \sum_{l \neq k} V_{il} - \frac{C}{T} \left(\sum_{j=1}^N V_{jk} - 1 \right) \right) \quad (41)$$

4.5 The Knapsack Problem

The knapsack problem can be defined as filling a knapsack with a subset of given N items i with associated values c_i and sizes a_{ji} such that their total values are maximized subject to a set of M size constraints. Namely,

$$\begin{aligned} \max \quad & \sum_{i=1}^N c_i s_i \\ \text{subject to} \quad & \sum_{i=1}^N a_{ji} s_i \leq b_j, \quad j = 1, \dots, M \end{aligned} \quad (42)$$

where s_i is the binary decision variable representing whether item i goes to knapsack or not.

The knapsack problem was studied in [1, 17, 50, 53, 54, 58, 62, 63]. In [53], a methodology for finding good solutions to the knapsack problem using NNs and MFT was presented. Inequality constraints were taken into account by adding appropriate penalty term to the energy function. Numerical experiments showed very good results against the branch and bound, LP, and SA. An energy function for this problem can be written as follows [53, 58]:

$$E(S) = - \sum_{i=1}^N c_i s_i + \alpha \sum_{j=1}^M G \left(\sum_{i=1}^N a_{ji} s_i - b_j \right) \quad (43)$$

where the first term maximizes total values of items and the second term corresponds to the constraints. Note that G is a penalty function to ensure that the constraints are satisfied. As penalty function $G(x) = xD(x)$ where $D(x)$ is the standard step function was proposed in [53, 54, 58, 62, 63].

To write the mean field equations for this energy function, a special treatment is needed because of the penalty function. As is shown in [53, 58] this can be done by replacing the derivative $\frac{\partial E(V)}{\partial V_i}$ in

$$V_i = \tanh \left(\frac{1}{T} \frac{\partial E(V)}{\partial V_i} \right) \quad (44)$$

by a difference

$$-c_i + \alpha \sum_{k=1}^M \left[G \left(\sum_{j=1}^N a_{kj} V_j - b_k \right) \Big|_{V_i=1} + G \left(\sum_{j=1}^N a_{kj} V_j - b_k \right) \Big|_{V_i=0} \right] \quad (45)$$

In [54], the MFT approach to the knapsack problem was extended to multiple knapsacks and generalized assignment problems. A hybrid algorithm that combines the LP with MFA was proposed to increase the performance of the proposed approach.

4.6 Other Combinatorial Optimization Problems

In addition, the following combinatorial problems have been also studied.

The Assignment Problem: A novel method named as the invisible hand algorithm for solving the assignment problem using MFT based on the Potts model is proposed in [39]. A continuous time dynamical system motivated by statistical physics was proposed to solve the problem and the corresponding convex energy function parameterized by the temperature was constructed. It was proved that for sufficiently low temperatures unique minimum of that energy function corresponds to an optimal solution of an associated assignment problem. The proposed MFT method and the assignment problem were used to demonstrate connections between MFT and other optimization techniques, in particular, barrier functions and interior point methods in [82, 83].

A method using a Hopfield NN to determine the optimum frequency assignment so that the signal interference is minimized in the satellite communication was proposed [41]. In the proposed solution method, the given frequency band was segmented into n frequency bands, and the assignment was derived by rearranging the segments using the two-dimensional NN arrays. An application of MFA algorithm in biophysics to solve the assignment problem was presented in [7] for the assignment of the nuclear magnetic resonance (NMR) spectra of larger (15-21 kDa) proteins to investigate the structure and dynamics of proteins.

The assignment problems are also considered as the matching problems. A general method for modeling matching problems in terms of elastic network and deformable templates with binary matching fields was described, and the winner take all problem as a matching problem was solved [81]. The global constraint that sum to 1 was imposed explicitly during the computation of the partition function like the Potts model. The track finding

problem is an assignment problem of fitting smooth curves to a given set of signal points. To solve this problem, the deformable templates approach was proposed [58, 63]. The assignment problem was treated as a problem of covering the graph vertices by the cycles of a minimum total weight, which do not intersect at the vertices in [50]. The MFT approach was extended to the generalized assignment problem [54]. In another study [17], a competition based NN was proposed to solve the generalized assignment problem.

The Circuit Partitioning: A MFA algorithm based on the Potts glass model for the circuit partitioning problem using the net-cut model was proposed [8]. The performance of the algorithm was experimentally evaluated in comparison with SA and Kernighan-Lin algorithm. The results indicated that MFA was a successful alternative heuristic for the circuit partitioning problem.

The Clustering Problem: A Potts spin-glass model to solve the standard clustering problem with unlabeled data is proposed [4]. Experimental results showed that for difficult clustering problems, the Potts model was far better than the vector quantization. An extension to deformable templates was also given. In another study [29], a maximum entropy framework namely deterministic annealing for pairwise data clustering was developed, and mean field approximation of pairwise clustering was proposed to calculate expectation values for the data assignments. Results of the pairwise data clustering algorithms in analyzing data and images indicated that deterministic annealing yielded substantially better results than conventional clustering concepts based on gradient descent minimization. Two other NN formulations for the clustering problem were given in [50, 73].

The Constraint Satisfaction Problem: Mean field methods for boosting backtrack search was described for solving classical and partial constraint satisfaction problems [9]. The NN approaches to the constraint satisfaction problem were reviewed in [47].

The Maximum Clique Problem: Mathematical models combining the Ising spin-glass model and NNs to solve the maximum clique problem was proposed [43, 44]. A parallel algorithm based on those models was introduced, and by using proposed algorithms, large-scale problems were solved in a reasonable computation time than the other known algorithms. In another study [28], NN approaches using SA and MFT to the maximum clique problem is presented. Another NN formulation for the maximum clique problem was presented [65].

The Scheduling Problems: A simplified scheduling problem where P teachers lecture Q classes in X classrooms at T time slots was mapped onto

a Potts model by using NNs [24]. The problem was to find a solution where all teachers give a lecture to each of the classes using available classrooms and time slots. Teacher-class pairs (P, Q) were mapped onto classroom-time slots (X, T). PQ distinct K -state Potts spins, with $K = XT$ was needed to map this problem. An energy function and the corresponding mean field equations were derived. By using MFA, a very efficient algorithm was resulted. The proposed model was experimented with numerical studies on Swedish high school system, and also gave very good results [25]. A review of this scheduling problem as well as other combinatorial problems were summarized in [62, 63, 57]. In another study [16], on similar problem namely an academic class scheduling problem at the university level, a variety of annealing techniques including SA and MFA to solve this problem was investigated. The best results were obtained by using SA.

A new method to solve the satellite broadcasting scheduling problem was proposed [3]. The problem was mapped onto an NN from which an energy function was derived. By minimizing the energy function using MFA, excellent results were consistently found for various size problems. A mean field approach based on the Potts model for solving resource allocation problems with nontrivial topology was developed and applied to airline crew scheduling problems [42]. Very good results were obtained for various size problems. In another study [17], a competition based NNs was proposed to solve the job scheduling with deadlines.

Other Problems: A number of interesting combinatorial optimization problems such as vertex cover, maximum independent set, number partitioning, maximum matching, set cover, graph morphism, and graph coloring were studied in [28, 50, 65].

5 Conclusions

The use of NNs for the combinatorial optimization problems and the NN approaches that have been applied to combinatorial optimization problems were reviewed. These approaches suggest that NNs are an alternative for solving combinatorial optimization problems as compared to other optimization techniques. Motivations for using NNs include the improvement in the speed of the operation through massively parallel computation, and possible hardware design advantages. One of the main advantages of NN approaches to classical optimization approaches is the inherently parallel and distributed nature of the dynamic solution procedure. Therefore, NNs are capable to

solve large-scale optimization problems in real time. This is essential in many engineering design, control, and optimization problems. Because of the inherent nature of parallel and distributed information processing in NNs, they are promising computational models for solving large-scale optimization problems in real-time.

References

- [1] S. Abe, J. Kawakami, and K. Hirasawa, Solving inequality constrained combinatorial optimization problems by the Hopfield neural networks, *Neural Networks* Vol.5 (1992) pp. 663-670.
- [2] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski, A learning algorithm for Boltzman machines, *Cognitive Science* Vol.9, (1985) pp. 147-169.
- [3] N. Ansari, E.S.H. Hou, and Y. Yu, A new method to optimize the satellite broadcasting schedules using the mean field annealing of a Hopfield neural network, *IEEE Transactions on Neural Networks* Vol.6 No.2 (1995) pp. 470-482.
- [4] M. Bengtsson and P. Roivainen, Using the Potts glass for solving the clustering problem, *International Journal of Neural Systems* Vol.6 No.2 (1995) pp. 119-132.
- [5] G. Bilbro, R. Mann, T.K. Miller, W.E. Snyder, D.E. Van Den Bout, and M. White, Optimization by mean field annealing, in D.S. Touretzky (ed.) *Proceedings of the Annual Conferences on Advances in Neural Information Processing Systems, Volume 1*, (Morgan Kaufmann Publishers, 1988) pp. 91-98.
- [6] A. Bovier and P. Picco, *Mathematical Aspects of Spin Glasses and Neural Networks*, (Birkhäuser, 1998).
- [7] N.E.G. Buchler, E.R.P. Zuiderweg, H. Wang, and R.A. Goldstein, Protein heteronuclear NMR assignments using mean-field simulated annealing, *Journal of Magnetic Resonance* Vol.125 (1997) pp. 34-42.
- [8] T. Bultan and C. Aykanat, Circuit partitioning using mean field annealing, *Neurocomputing* Vol.8 (1995) pp. 171-194.

- [9] B. Cabon, G. Verfaillie, D. Martinez, and P. Bourret, Using mean field methods for boosting backtrack search in constraint satisfaction problems, in W. Wahlster (ed.) *12th European Conference on Artificial Intelligence*, (John Wiley and Sons, Ltd., 1996).
- [10] D. Chowdhury, *Spin Glasses and Other Frustrated Systems*, (Princeton University Press, 1986).
- [11] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, (John Wiley and Sons, 1993).
- [12] R. Courant and D. Hilbert, *Methods of Mathematical Physics*, (Interscience Publishers Inc., 1953).
- [13] K.A. Dowsland, Simulated annealing, in C.R. Reeves (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, (John Wiley and Sons, 1993) pp. 20-69.
- [14] R. Durbin, R. Szeliski, and A.L. Yuille, An analysis of the elastic net approach to the travelling salesman problem, *Neural Computation* Vol.1 (1989) pp. 348-358.
- [15] I.M. Elfadel, Convex potential and their conjugates in analog mean-field optimization, *Neural Computation* Vol.7 (1995) pp. 1079-1104.
- [16] S. Elmohamed, P. Coddington, and G. Fox, A comparison of annealing techniques for academic course scheduling, Northeast Parallel Architectures Center (NPAC) technical report SCCS-777, January 25, 1997.
- [17] L. Fang and T. Li, Design of competition-based neural networks for combinatorial optimization, *International Journal of Neural Systems* Vol.1 No.3 (1990) pp. 221-235.
- [18] L. Fang, W.H. Wilson, and T. Li, Mean field annealing neural net for quadratic assignment, in *International Neural Network Conference, July 9-13, Paris, France*, (Kluwer Academic Publishers, 1990) pp. 282-286.
- [19] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, (Prentice-Hall, 1994).
- [20] L. Faybusovich, Interior point methods and entropy, *IEEE Conference on Decision and Control*, (1991) pp. 2094-2095.

- [21] K.H. Fischer and J.A. Hertz, *Spin Glasses*, (Cambridge University Press, 1991).
- [22] C.A. Floudas and P.M. Pardalos, *Recent Advances in Global Optimization*, (Princeton University Press, 1994).
- [23] D. Geiger and F. Girosi, Coupled Markov random fields and mean field theory, in D.S. Touretzky (ed.) *Proceedings of the Annual Conferences on Advances in Neural Information Processing Systems, Volume 2*, (Morgan Kaufmann Publishers, 1989) pp. 660-667.
- [24] L. Gislen, C. Peterson, and B. Soderberg, Teachers and classes with neural networks, *International Journal of Neural Systems* Vol.1 No.1 (1989) pp. 167-183.
- [25] L. Gislen, C. Peterson, and B. Soderberg, Complex scheduling with Potts neural networks, *Neural Computation* Vol.4 (1992) pp. 805-831.
- [26] R.J. Glauber, Time-dependent statistics of the Ising model, *Journal of Mathematical Physics* Vol.4 No.2 (1963) pp. 294-307.
- [27] S. Haykin, *Neural Networks: A Comprehensive Foundation*, (Macmillan College Publishing Company, 1994).
- [28] L. Herault and J.-J. Niez, Neural networks and combinatorial optimization: a study of NP-complete graph problems, in E. Gelenbe (ed.) *Neural Networks: Advances and Applications*, (Elsevier Science Publishers B. V., 1991) pp. 165-213.
- [29] T. Hofmann and J.M. Buhmann, Pairwise data clustering by deterministic annealing, *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol.19 No.1 (1997) pp. 1-14.
- [30] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, in *Proceedings of the National Academy of Sciences of the U.S.A., Biophysics*, (1982) pp. 2554-2558.
- [31] J.J. Hopfield, Neurons with graded response have collective computational properties like those of two-state neurons, in *Proceedings of the National Academy of Sciences of the U.S.A., Biophysics*, (1984) pp. 3088-3092.

- [32] J.J. Hopfield and D.W. Tank, "Neural" computation of decisions in optimization problems, *Biological Cybernetics* Vol.52 (1985) pp. 141-152.
- [33] R. Horst, P.M. Pardalos, and N.V. Thoai, *Introduction to Global Optimization*, (Kluwer Academic Publishers, 1995).
- [34] H. Igarashi, A solution to combinatorial optimization problems using a two-layer random field model: Mean-field approximation, in *World Congress on Neural Networks, July 11-15, Portland, Oregon, Volume 1*, (Lawrance Erlbaum Associates Inc. Publishers, 1993) pp. 283-286.
- [35] H. Igarashi, A solution for combinatorial optimization problems using a two-layer random field model: Mean-field approximation, *Systems and Computers in Japan* Vol.25 No.8 (1994) pp. 61-71.
- [36] S. Ishii and M.-A. Sato, Chaotic Potts spin model for combinatorial optimization problems, *Neural Networks* Vol.10 No. 5 (1997) pp. 941-963.
- [37] I. Kanter and H. Sompolinsky, Graph optimisation problems and the Potts glass, *Journal of Physics A: Math. Gen.* Vol.20 (1987) pp. L673-L679.
- [38] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, Optimization by simulated annealing, *Science* Vol.220 No.4598 (1983) pp. 671-680.
- [39] J.J. Kosowsky and A.L. Yuille, The invisible hand algorithm: Solving the assignment problem with statistical physics, *Neural Networks* Vol.7 No.3 (1994) pp. 477-490.
- [40] N. Kurita and K.-I. Funahashi, On the Hopfield neural networks and mean field theory, *Neural Networks* Vol.9 No.9 (1996) pp. 1531-1540.
- [41] T. Kurokawa and S. Kozuka, Use of neural networks for the optimum frequency assignment problem, *Electronics and Communications in Japan, Part 1* Vol.77 No.11 (1994) pp. 106-116.
- [42] M. Lagerholm, C. Peterson, and B. Soderberg, Airline crew scheduling with Potts neurons, *Neural Computation* Vol.9 (1997) pp. 1589-1599.
- [43] K.-C. Lee and Y. Takefuji, Maximum clique problems: Part 1, in Y. Takefuji and J. Wang (eds.) *Neural Computing for Optimization and Combinatorics*, (World Scientific, 1996) pp. 31-61.

- [44] K.-C. Lee and Y. Takefuji, Maximum clique problems: Part 2, in Y. Takefuji and J. Wang (eds.) *Neural Computing for Optimization and Combinatorics*, (World Scientific, 1996) pp. 63-77.
- [45] B.C. Levy and M.B. Adams, Global optimization with stochastic neural networks, in *Neural Networks for Optimization and Signal Processing, Proceedings of the First International Conference on Neural Networks, San Diego*, (1987) pp. 681-690.
- [46] W.A. Little, The existence of persistent states in the brain, *Mathematical Biosciences* Vol.19 (1974) pp. 101-120.
- [47] C.-K. Looi, Neural network methods in combinatorial optimization, *Computers in Operations Research* Vol.19 No.3/4 (1992) pp. 191-208.
- [48] S. Matsuda, Set-theoretic comparison of mapping of combinatorial optimization problems to Hopfield neural networks, *Systems and Computers in Japan* Vol.27 No.6 (1996) pp. 45-59.
- [49] W.S. McCullough and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics* Vol.5 (1943) pp. 115-133.
- [50] I.I. Melamed, Neural networks and combinatorial optimization, *Automation and Remote Control* Vol.55 No.11 (1994) pp. 1553-1584.
- [51] M. Mezard, G. Parisi, and M.A. Virasoro, *Spin Glass Theory and Beyond*, (World Scientific, 1987).
- [52] B. Muller and J. Reinhardt, *Neural Networks: An Introduction*, (Springer-Verlag, 1990).
- [53] M. Ohlsson, C. Peterson, and B. Soderberg, Neural networks for optimization problems with inequality constraints: The knapsack problem, *Neural Computation* Vol.5 (1993) pp. 331-339.
- [54] M. Ohlsson and H. Pi, A study of the mean field approach to knapsack problems, *Neural Networks* Vol.10 No.2 (1997) pp. 263-271.
- [55] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, (Prentice-Hall, 1982).

- [56] C. Peterson, Parallel distributed approaches to combinatorial optimization: Benchmark studies on traveling salesman problem, *Neural Computation* Vol.2 (1990) pp. 261-269.
- [57] C. Peterson, Mean field theory neural networks for feature recognition, content addressable memory and optimization, *Connection Science* Vol.3 No. 1 (1991) pp.3-33.
- [58] C. Peterson, Solving optimization problems with mean field methods, *Physica A* Vol.200 (1993) pp. 570-580.
- [59] C. Peterson and J.R. Anderson, A mean field theory learning algorithm for neural networks, *Complex Systems* Vol.1 (1987) pp. 995-1019.
- [60] C. Peterson and J.R. Anderson, Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem, *Complex Systems* Vol.2 (1988) pp. 59-89.
- [61] C. Peterson and B. Soderberg, A new method for mapping optimization problems onto neural networks, *International Journal of Neural systems* Vol.1 No.1 (1989) pp. 3-22.
- [62] C. Peterson and B. Soderberg, Artificial neural networks, in C.R. Reeves (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, (John Wiley and Sons, 1993) pp. 197-242.
- [63] C. Peterson and B. Soderberg, Artificial neural networks, in E. Aarts and J.K. Lenstra (eds.) *Local Search in Combinatorial Optimization*, (John Wiley and Sons, 1997) pp. 177-213.
- [64] F. Qian and H. Hirata, A parallel computation based on mean field theory for combinatorial optimization and Boltzman machines, *Systems and Computers in Japan* Vol.25 No.12 (1994) pp. 86-97.
- [65] J. Ramanujam and P. Sadayappan, Mapping combinatorial optimization problems onto neural networks, *Information Sciences* Vol.82 (1995) pp. 239-255.
- [66] C.R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*, (John Wiley and Sons, 1993).
- [67] L.E. Reichl, *A Modern Course in Statistical Physics*, (University of Texas Press, 1980).

- [68] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning representations by back-propagating errors, *Nature (London)* Vol.323 (1986) pp. 533-536.
- [69] M.-A. Sato and S. Ishii, Bifurcations in mean field theory, *Physical Review E* Vol.53 No.5 (1996) pp. 5153-5168.
- [70] G.M. Shim, D. Kim, and M.Y. Choi, Potts-glass model of layered feed-forward neural networks, *Physical Review A* Vol. 45 No. 2 (1992) pp. 1238-1248.
- [71] P. Stolorz, Merging constrained optimisation with deterministic annealing to "solve" combinatorially hard problems, in J.E. Moody, S.J. Hanson, and R.P. Lippmann (eds.) *Advances in Neural Information Processing Systems*, (Chapman and Hall, 1991) pp. 1025-1032.
- [72] Y. Uesaka, Mathematical basis of neural networks for combinatorial optimization problems, *Optoelectronics - Devices and Technologies* Vol.8 No.1 (1993) pp. 1-9.
- [73] K. Urahama, Mathematical programming formulation for neural combinatorial optimization algorithms, *Electronics and Communications in Japan* Vol.78 No.9 (1995) pp. 67-75.
- [74] K. Urahama and S.-I. Ueno, A gradient system solution to Potts mean field equations and its electronic implementation, *International Journal of Neural Systems* Vol.4 No.1 (1993) pp. 27-34.
- [75] K. Urahama and T. Yamada, Constrained Potts mean field systems and their electronic implementation, *International Journal of Neural Systems* Vol.5 No.3 (1994) pp. 229-239.
- [76] D.E. Van Den Bout and T.K. Miller, Graph partitioning using annealed networks, *IEEE Transactions on Neural Networks* Vol.1 No.2 (1990) pp. 192-203.
- [77] P.J.M. Van Laarhoven, *Theoretical and Computational Aspects of Simulated Annealing*, (Stichting Mathematisch Centrum, 1988).
- [78] J. Wang, Deterministic neural networks for combinatorial optimization, in O.M. Omidvar (ed.) *Progress in Neural Networks*, (Ablex, 1994) pp. 319-340.

- [79] F.Y. Wu, The Potts model, *Reviews of Modern Physics* Vol.54 No.1 (1982) pp. 235-268.
- [80] L. Xu and A.L. Yuille, Robust principal component analysis by self-organizing rules based on statistical physics approach, *IEEE Transactions on Neural Networks* Vol.6 No.1 (1995) pp. 131-143.
- [81] A.L. Yuille, Generalized Deformable models, statistical physics, and matching problems, *Neural Computation* Vol.2 (1990) pp. 1-24.
- [82] A.L. Yuille and J.J. Kosowsky, Statistical physics algorithms that converge, in R.J. Mammone (ed.) *Artificial Neural Networks for Speech and Vision*, (Chapman and Hall, 1993) pp. 19-36.
- [83] A.L. Yuille and J.J. Kosowsky, Statistical physics algorithms that converge, *Neural Computation* Vol.6 (1994) pp. 341-356.
- [84] A.L. Yuille, P. Stolorz, and J. Utans, Statistical physics, mixtures of distributions, and the EM algorithm, *Neural Computation* Vol.6 (1994) pp. 334-340.

Frequency Assignment Problems

Robert A. Murphey
Wright Laboratory
Eglin AFB, FL 32542 USA
E-mail: `murphey@eglin.af.mil`

Panos M. Pardalos
Center for Applied Optimization, ISE Department
University of Florida, Gainesville, FL 32611 USA
E-mail: `pardalos@ufl.edu`

Mauricio G. C. Resende
Information Sciences Research Center
AT&T Labs Research, Florham Park, NJ 07932 USA
E-mail: `mgcr@research.att.com`

Contents

1	Introduction	296
2	The Frequency Assignment Problem	298
2.1	Frequency-Distance Constraints	300
2.2	Frequency Constraints	301
2.2.1	Mechanisms of interference	303
2.3	Integer Constraints	306
2.3.1	General channel constraints	308
2.3.2	F*D integer constraints	308
2.3.3	Channel separation matrix	309
3	Graph Theory	309
3.1	Graphs	310
3.2	Graph Operations	311
3.3	Special Graphs	311
3.4	Colorings and Cliques	312
3.5	Vertex Orderings and Their Implied Graphs	313

4 Graph Formulations	315
4.1 Co-Channel and Adjacent Channel Frequency Assignments	315
4.2 T -Colorings and Bounds on Minimum Order and Span	316
4.3 T -Colorings of Multigraphs	324
4.4 List Colorings and List T -Colorings	326
4.5 Set Colorings and Set T -Colorings	328
4.6 No-Hole Colorings	331
4.6.1 N_r -Colorings	331
4.6.2 N_r^k -Colorings	332
5 Integer Programming Formulations	333
6 Exact Solution Techniques	342
6.1 Backtracking Methods	342
6.2 Set Covering Approach	343
7 Approximate Solution Methods	343
7.1 Sequential Heuristics For Simple Graph Coloring	344
7.2 The Greedy T -coloring Algorithm	347
7.2.1 Greedy T -colorings of Complete Graphs	348
7.2.2 Greedy T -colorings and Vertex Orderings	351
7.3 The Greedy k-Tuple T -coloring Algorithm	353
7.4 Interset Graphs	354
7.5 Simulated Annealing, Tabu Search, and Genetic Algorithms	357
7.5.1 Simulated Annealing	358
7.5.2 Tabu Search	361
7.5.3 Tabu Thresholding	362
7.5.4 Genetic Algorithms	362
7.5.5 Neural Networks	364
7.6 Polyhedral Methods	365
7.6.1 Linear Relaxations	365
7.6.2 Quadratic Relaxations	366
8 Evaluation of Algorithms	367
References	

1 Introduction

The term *frequency assignment* has been used to describe many types of problems which, quite often, have different modeling needs and objectives. These problems include:

1. Planning models for permanent spectrum allocation, licensing, and regulation which maximize utilization of *all* radio spectra [94].
2. Planning models for network design within a given allocation to include; aeronautical mobile, land mobile, maritime mobile, broadcast, land fixed (point-to-point) and satellite.
3. On-line algorithms for dynamically assigning frequencies to users within an established network. Of special interest here are land cellular mobile systems, where an enormous amount of research has been done. A paper by Katzela and Naghshineh [55] contains nearly 100 references to works just in cellular dynamic channel assignment.

All these problems share a common trait, namely the optimal assignment of a limited radio spectrum resource to an ever growing number of users. Moreover, an increasing amount of spectrum is needed per user to transfer what is becoming an increasingly dense amount of information (e.g. graphics and video for personal communications services).

The FAP has been studied by engineers, mathematicians and scientists and yet, in its general form, remains a very difficult problem. The FAP is closely associated with simple graph coloring so most research has centered on graph coloring theories and algorithms. Engineers have focused their energies on fast heuristics for very specialized problem instances that require a solution immediately. As a result, *sequential* assignment methods have been developed that consider transmitters in some sequence and make the most efficient assignment of spectrum at each step.

The optimal use of spectrum regards the assignment that uses the least amount of spectrum for an acceptable level of interference or the one that minimizes the total interference for an allotted portion of spectrum. Optimality is not guaranteed nor often achieved by sequential methods. In response, mathematicians have developed many graph theoretical results by introducing *T-coloring* theory to model many problem instances. A graph model of the FAP usually results when we let transmitters be the vertices of a graph and edges between vertices describe potential interference. Frequencies are assigned to vertices with the objective of either minimizing the spectrum used or minimizing interference. *T-colorings* model interference as a set of integer frequencies (T-set) that are assigned to an edge and hence may not be shared by the pair transmitters that are modeled by the incident vertices. Many conditions on optimality for sequential techniques have been developed via this approach. However, these results often cannot be used directly to solve realistic frequency assignment problems.

Operations research scientists have studied polyhedral models and polyhedral algorithms for some realistic scenarios. Polyhedral models benefit greatly from theoretical graph results as they provide structure and bounds on more realistic problem instances. Polyhedral techniques function by *relaxing* integer models to continuous ones and then applying classical optimization theory to the relaxed problem. Operations researchers have developed many efficient algorithms for solving continuous optimization problems.

To solve realistic problems, all three of the above mentioned perspectives must be considered.

This chapter is organized as follows. In Section 2 the frequency assignment problem is defined and interference mechanisms are modeled. Because the FAP is so closely related to graph coloring, Section 3 presents a terse review of graph theory and terminology. In Section 4 we develop models of the FAP based upon graphs and graph colorings. Theoretical bounds on the “optimal” assignment are determined for many assumptions regarding the graphs and assignment methods. In Section 5, frequency assignment problems are cast as mathematical programs with integer valued variables. Section 6 explores exact solution techniques to the graph and integer program formulations. However, since graph coloring is known to be a hard problem and branch and bound (the principle method of solving integer programs) can degenerate to total enumeration, exact methods are often not practical. Thus Section 7 examines approximate solution techniques including coloring heuristics, metaheuristics like simulated annealing, and polyhedral methods that operate on a linear or quadratic relaxation of an integer formulation. Section 8 discusses the issues encountered when evaluating approximate algorithms including test problem generators and optimality measures.

2 The Frequency Assignment Problem

A frequency assignment problem (FAP) models the task of assigning radio spectrum to a set of transmitters. That is, if V is the set of all transmitters with $v \in V$, $f(v)$ denotes the continuum of frequencies assigned to v by the assignment rule f . A feasible frequency assignment must additionally satisfy certain interference and bandwidth constraints, I and B respectively. An optimal frequency assignment globally minimizes a cost function that depends upon the particular objective of the problem. Two possible objectives and thus two basic problem formulations are widely used:

- minimize a bandwidth function subject to an acceptable level of interference

- minimize the system interference given a fixed frequency allotment

where the bandwidth function is either the span of frequencies (difference between maximum and minimum frequencies) assigned or the order (integer number) of channels assigned. The system interference function is less concise but can be defined as the number of resources denied service, or the bit error rate for the combined network. Intuitively, we understand that these objectives are at odds with one another. The variables we may adjust are frequency, space and time [95]. Naturally, if two resources have the potential to interfere, we can assign them different frequencies. However, if possible, we would like to reuse frequencies by taking advantage of the spatial nature of radio signal propagation which dictates that signal power is a function of distance. Transmitting during time slots when no other transmitter is using the channel is another way to conserve spectrum. A technique that interleaves transmission times is called a time division multiple access (TDMA) process. In a similar vein, uncorrelated pseudo-random noise codes may be used to modulate signals in such a manner so as to permit multiple transmitters to operate at the same frequency. This technique is termed code division multiple access (CDMA).

Our definition of the frequency assignment problem will only consider frequency and space as variables. While TDMA and CDMA features can be designed into the network, they can be considered as a separate issue apart from frequency assignment by adopting a layered model of data networks [9]. In this model, the effects of modulation, radiation elements and so forth are optimized independently within one layer and the results mapped into higher layers where the frequency assignment takes place. In Section 8 we will see that the modulation technique of the so called *physical* layer maps to the *network* layer and determines the interference issues for the frequency assignment problem.

It is important to understand that interference can only be defined with respect to a receiver. That is, any signal, occurring within the same channel that a desired signal is to be received in, is considered to interfere with the reception of the desired signal if it is of sufficient power (as measured at the designated receiver). Consequently, even though frequencies are assigned to transmitters, measurements at receivers define the resulting levels of interference. Interference power is a function of transmitter power, receiver sensitivity, antenna gains, patterns and polarizations, and channel loss. Channel loss is a function primarily of distance, frequency, and weather and is quantified by minimum acceptable signal to noise power ratio, or maximum permissible interference to noise power ratio as measured at a receiver. We

can assume that most of these factors are already determined or beyond our ability to influence. Consequently, we will define interference directly as a function of frequency and distance.

As a final note, sometimes the term channel is used instead of frequency when describing these problems. Technically, there is a difference. Frequency corresponds to a single wavelength of a continuous sinusoid. A continuum of frequencies in some medium (here we will only consider free space) is a channel with bandwidth equal to the difference between the maximum and minimum frequencies defined for the channel. The bandwidth of a channel is a function of the modulating signal and modulation technique applied. Consequently, channel is the more precise term, however, channel and frequency are typically used interchangeably since the center or *carrier* frequency of a channel often represents the entire channel, with the understanding that the center frequency has a defined bandwidth associated with it. *Band* is also often used to refer to a continuum of frequencies and typically refers to a contiguous group of channels.

2.1 Frequency-Distance Constraints

In [94], Zoellner defines a frequency-distance function, F^*D , to be the effective isolation between two users. F^*D is the product of two terms: the difference between two user's physical locations and the difference between their respective frequencies. Subsequently, the authors of [8] and [95] also define problems where interference is mitigated by adjusting the spatial separation between pairs of transmitters and receivers. Frequency-distance interference constraints are often easy to construct; compute the distances between pairs of transmitters and receivers and compare them with a minimum distance required to prevent interference, given that the pairs operate on the same frequency. There are many ways to define frequency-distance interference constraints, each depending upon the problem at hand. Two examples are provided below.

Example 2.1. Broadcast Network. Problems of this type are defined in [3]. In a broadcast scenario, each transmitter broadcasts to a set of subscribing receivers within some defined service area. Assume each transmitter at location v_i , $i = 1 \dots V$ broadcasts uniformly and *onmidirectionally* with service area radius s_i . Define the minimum distance function, $d(v_i, v_j) = c(s_i + s_j)$, where $c \geq 1$ is a safety margin which depends upon the desired quality of service. The actual physical separation between any two transmitters is $D(v_i, v_j) = \|v_i - v_j\|$, $i \neq j$. If $D(v_i, v_j) > d(v_i, v_j)$, then $f(v_i) \neq f(v_j)$.

Example 2.2. Cellular Network Base Station Assignment. As previously stated, cellular networks have been studied at great length. There are actually many frequency assignment problems defined for cellular systems. This particular problem, discussed at length in [31, 37, 39, 38, 53, 55, 58] and many others, concerns the interference-free assignment of sets of frequencies to cellular base stations. In cellular networks, the total service region is partitioned into a predefined number of hexagonal cells, each with a base station located at its center. Mobile users can only communicate with other users via the base station assigned to the cell which they currently occupy. For example, in Figure 1, mobile a , wishing to send a message to mobile b , must send the message to base station v_i which then routes the message to base station v_j and in turn to mobile b . Channels are assigned to mobile users within each cell by the base stations and cannot be reused within a cell. Each base station is allotted channels that it may use. If that number is insufficient to meet the needs of the mobile callers within its cell, it may be able to borrow channels from other cells. In some systems, the number of channels a given cell or base station is assigned may even adapt to the caller density. The assignment of channels to cells is the frequency assignment problem of interest. Clearly, if the signal power of each call is completely contained within a single channel and each cell is allotted a disjoint set of channels, there will be no interference. However, this approach is wasteful. If two cells are spaced sufficiently distant, they should be able to use the same set of channels. This distance, called the reuse distance d , is compared to distances $D(v_i, v_j)$ measured between pairs of base stations, v_i and $v_j, i \neq j$. If $D(v_i, v_j) > d$, then $f(v_i) \neq f(v_j)$.

2.2 Frequency Constraints

As pointed out by Hale in [46], the papers [13, 26, 34] define interference constraints as a function of frequency alone. Constraints of this type are important when physical distances between transmitters are either unknown or uncontrollable, and only frequency may be adjusted. Networks with mobile nodes are a good examples of this.

The effects of interference are often represented in an *interference diagram* such as that shown in Figure 2. An interference diagram shows what frequency choices for a transmitter (horizontal axis) will interfere with a receiver's chosen frequencies (vertical axis). The shaded band in Figure 2 represents frequency combinations that will result in some level of interference. So, as an example, if in Figure 2 we choose $f(v) = 10$, $f(u) \neq 10 \pm 3.5$. Sometimes interference diagrams have a second, thinner band and sometimes

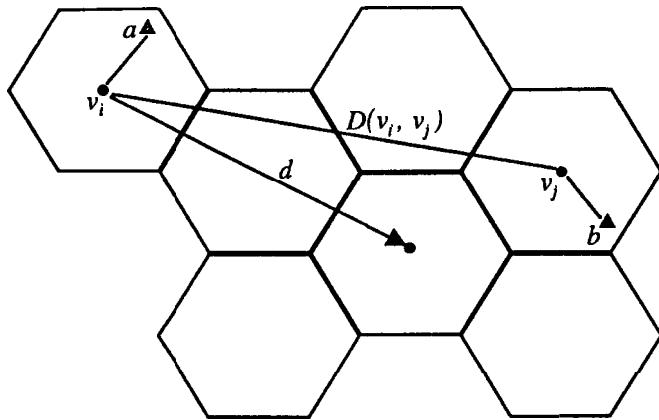


Figure 1: Example Cellular Base Station Assignment Problem

even a third band above and parallel to the existing band [60, 81]. These bands represent the effects of radiated harmonics and spurious noise from the modulation process including intermediate frequencies and intermodulation products. They are often accounted for by expanding the limits of the main band to include them, since attempting to assign a receiver frequency between a transmitter's radiated harmonics is usually not practical and most modern transmitters utilize filters to suppress unwanted harmonics. For the problem instances where harmonics should be considered [65], Section 2.3 develops a general model that can easily account for them. For our purposes, the interference diagram of Figure 2 will represent most problems sufficiently. Note that the width of the band depends upon factors discussed earlier such as physical separation of the transmitter and receiver, frequency, modulation, and so on. However, since in this section we are resigned to being unable to influence these factors, the width of the interference band is solely dependent upon the power spectral density of the transmitter signal present at the receiver.

A natural way to express interference as a constraint is to simply say that $|f(v) - f(u)| > d$, where d is some positive real number that defines the minimum frequency “distance” between v and u . In our example above, $d = 3.5$. However, since our problem definition assigns frequencies to transmitters, our constraints must be on pairs of transmitters. As it turns out, this is not difficult to accomplish.

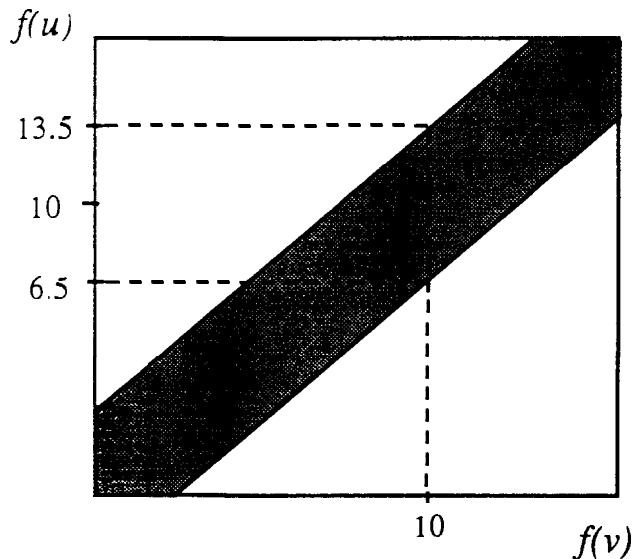


Figure 2: An interference diagram

2.2.1 Mechanisms of interference

As we shall see, all interference processes can be sufficiently explained by a combination of 3 fundamental mechanisms, far-site, co-site, and intermodulating co-site interferences. Through the remainder of this chapter, we shall define \mathbb{R}_+ to be the non-negative real numbers and Z_+ to be the set of nonnegative integers.

Far-Site Interference. Far-site interference occurs when a transmitter interferes with a receiver that is located out of its immediate region, thus preventing the receiver from detecting a desired signal another transmitter is sending it. This mechanism is shown in Figure 3. In this figure (as in Figures 4 and 5), nodes represent physical locations of transmitters and receivers, the solid directed arcs represent desired transmission links, and the directed dashed arc represents potential interference. So, transmitter v_j , while completing a desired link l_{jh} , causes interference to far-sited receiver u_k and in the process, disrupts the desired link l_{ik} from transmitter v_i . As Smith points out in [81], the potential for interference between transmitters v_i and v_j requires that we constrain frequency assignments $f(v_i), f(v_j)$ with respect to receiver u_k . For now consider f to be a mapping of v_i to an interval on the positive real line. Then, we must develop an interference diagram for

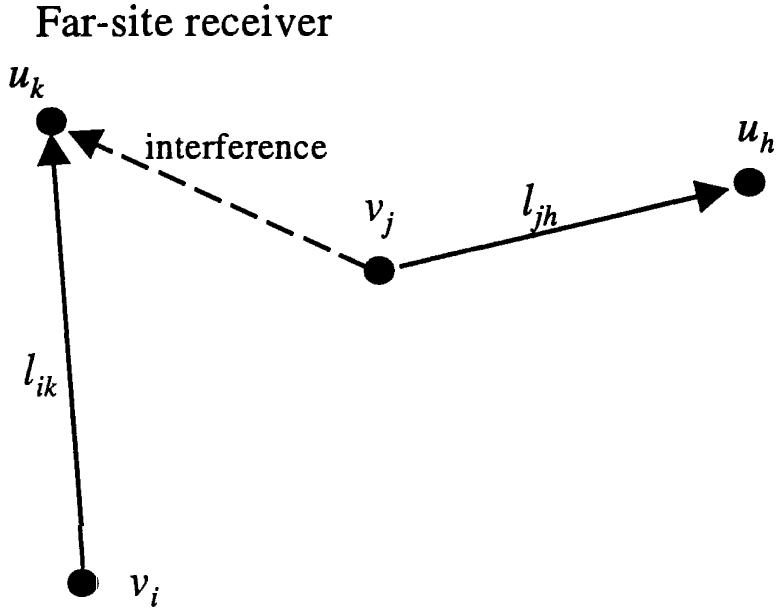


Figure 3: Far-site interference

$[f(v_j), f(u_k)]$ based upon the physical problem, and since $f(v_i) \supseteq f(u_k)$, ¹

$$I(\text{Far}) = \{|f(v_i) - f(v_j)| > d_{i,j,k} : i, j \in V, i \neq j \text{ and } k \in U^i \cup U^j\}$$

where V is the set of all transmitter indices, U^i and U^j are the sets of all receiver indices far-sited from transmitters v_i and v_j respectively, and $d_{ijk} \in \mathbb{R}_+$ is extracted from the interference diagram for $[f(v_j), f(u_k)]$.

Co-Site Interference. Interference to a receiver from a transmitter that is located at the same geographical site is termed co-site interference. Figure 4 illustrates this interference mechanism where transmitter v_k , while closing link l_{kj} , interferes with co-sited receiver u_k . Similar to the far-sited case, we wish to develop a constraint on $f(v_k), f(v_i)$ with respect to the receiver u_k and do so by developing an interference diagram for $[f(v_k), f(u_k)]$ and

¹ Typically, a receiver requires only a subset of frequencies emitted by a transmitter to accurately demodulate the signal. The remaining transmitter frequencies are considered *spurious* and are usually filtered, but cannot be completely eradicated.

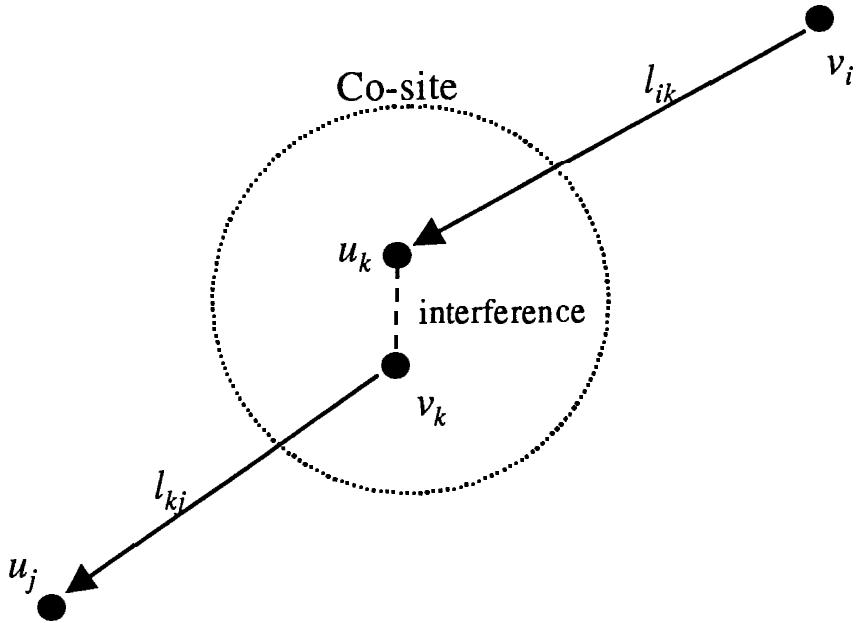


Figure 4: Co-site interference

letting $f(v_i) \supseteq f(u_k)$,

$$I(\text{Co}) = \{|f(v_i) - f(v_k)| > d_{i,k} : i \in V, i \neq k \text{ and } k \in U^c\}$$

where U^c is the set of indices for all co-sited pairs of receivers and transmitters and $d_{i,k} \in \Re_+$ is extracted from the interference diagram for $[f(v_k), f(u_k)]$.

Intermodulating Co-Site Interference. Until now we have considered only one type of interference, namely the unintentional transmissions of two or more transmitters in the same band. We have seen that the interference diagram sufficiently characterizes this type of interference in far-site and co-site mechanizations. In fact, there is third type of interference that concerns us and is caused by the intermodulation of two or more transmitters. Other noise mechanisms such as intentional noise (jamming), or environmental “channel” noise are beyond our ability to control and consequently will not be considered. When a receiver detects two or more dissimilar frequencies, nonlinearities in the receiver permit interfering signal energy to occur at the difference and sum of pairwise combinations of these frequencies. Intermodulation phenomena is thoroughly explained in [20]. Intermodulation is

usually only a problem for co-sited transmitters, so, a prerequisite for consideration of this mechanism is that two or more transmitters be co-sited with a receiver, as shown in Figure 5. Clearly, each transmitter at the co-site, $v_{k(1)}$ and $v_{k(2)}$, must be constrained from interfering with receiver u_k in a pure co-site sense as described in the previous paragraph. However, we must now also consider how the frequencies $f(v_{k(1)}), f(v_{k(2)})$ intermodulate to complete the constraint on $f(u_k)$ and hence $f(v_h)$. In the two previous cases we utilized an interference diagram as the basis for our constraint. For the intermodulation case, the interference diagram is 3-dimensional; depending upon the signal power and frequency of both co-sited transmitters and the nonlinear gain characteristic of the receiver's amplifier. Nonetheless, the result is similar to the 2-dimensional case such that $|2f(v_{k(1)}) - f(v_{k(2)}) - f(u_k)| > d$ and $|f(v_{k(1)}) - 2f(v_{k(2)}) - f(u_k)| > d$ where $d \in \Re_+$. Since $f(v_h) \supseteq f(u_k)$,

$$I(\text{Mod}) = \left\{ \begin{array}{l} |2f(v_{k(1)}) - f(v_{k(2)}) - f(v_h)| > d_{k(x)k(y)h} \\ |f(v_{k(1)}) - 2f(v_{k(2)}) - f(v_h)| > d_{k(x)k(y)h} \end{array} : \begin{array}{l} h \in V, h \neq k, \\ x, y \in V^k, x \neq y, \\ k \in U^c \end{array} \right\}$$

where V^k is the set of indices for all transmitters co-sited at site k and $d_{k(x)k(y)h}$ is derived from the 3-dimensional interference graph for $[f(v_{k(x)}), f(v_{k(y)}), f(u_k)]$. Intermodulation constraints are well explained by Leung in [61] and he provides some good examples.

2.3 Integer Constraints

For each of the interference mechanisms described in Section 2.2.1, the frequency separation between any two transmitters in the network is constrained by a positive scalar, d , which represents the length of a closed and continuous positive real interval (a continuum of frequencies). If we interpret the frequency assignment rule, f as a mapping of v_i to a real number on a closed and continuous positive real interval $\forall i \in V$, then the frequency constrained feasibility problem is to find f such that $I(\text{Far}), I(\text{Co}), I(\text{Mod})$ hold. From a practical standpoint, knowing d is difficult at best, often impossible. An approximate alternative is to partition the spectrum, $B \in \Re_+$, into mutually exclusive subintervals of equal length which we will define to be channels, that is,

$$B = \{C_1, C_2, \dots, C_k, \dots, C_K\} \quad | \quad C_j \cap C_k = \{0\}, j, k = 1 \dots K, j \neq k, \\ C_1 \cup C_2 \cup \dots \cup C_K = B$$

where the index of each channel is a positive integer, assigned ascending with frequency. The interval length may be selected to be the smallest required

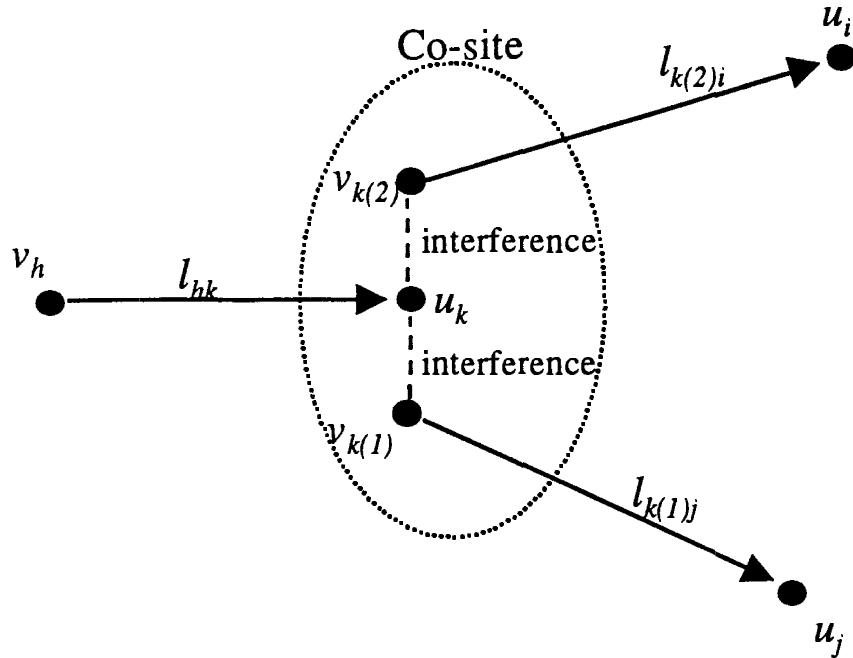


Figure 5: Intermodulating co-site interference

bandwidth of all the transmitters. The feasibility problem then becomes a mapping f of transmitter v_i to a channel, C_k such that $I(\text{Far}), I(\text{Co}), I(\text{Mod})$ hold. However, now $d \in \mathbb{Z}_+$, a positive integer, represents the number of channels separating pairs of transmitters. The three fundamental interference mechanisms are redefined for integrality as follows.

Co-Channel Constraints. A pair of transmitters cannot be assigned the same channel without interfering. Hence, in $I(\text{Far}), I(\text{Co}), I(\text{Mod})$, $d = 0$.

Adjacent Channel Constraints. A pair of transmitters cannot be assigned channels that differ by less than one. Since two channels are adjacent when $d = 1$, in $I(\text{Far}), I(\text{Co}), I(\text{Mod})$, $d = 1$.

k^{th} Adjacent Channel Constraints. A pair of transmitters cannot be assigned channels that differ by less than k . Since two channels are k -adjacent when $d = k$, in $I(\text{Far}), I(\text{Co}), I(\text{Mod})$, $d = k$.

2.3.1 General channel constraints

In a landmark paper, Hale [46] described a unifying theory and terminology for all frequency assignment problems. Hale defines the k^{th} general constraint as $R(k) = [T(k), d(k)]$, where $T(k) \subset Z_+$, $d(k) \in Z_+$, $k = 0, 1, \dots, K$ and $0 = T(0) \subset T(1) \subset \dots \subset T(K)$ and $d(0) > d(1) > \dots > d(K)$. Then

$$\text{if } |f(v_i) - f(v_j)| \notin T(k), \forall v_i, v_j \in V, \rightarrow f \text{ feasible for } V, T(k).$$

Thus, the k^{th} adjacent constraint is a special instance of the general k^{th} constraint where $T(k) = \{0, 1, \dots, k\}$ and $d(k) \in Z_+$. Notice that in general, $T(k)$ need not be a contiguous set of integers; in fact it may be any set of integers that satisfies the nesting property define above. Thus, as we mentioned at the beginning of Section 2.2, problems with significant harmonics can easily be modeled. The following example from [65] illustrates the utility of general channel constraints.

Example 2.3. The UHF Taboo Problem. In Figure 6 the power spectrum for a UHF television transmitter is plotted. Based upon this data, The FCC established interference levels at distances $d(0) = 155$ miles, $d(1) = 75$ miles, $d(2) = 60$ miles, $d(3) = 55$ miles, and $d(4) = 20$ miles. From the figure, the T-sets corresponding to these levels are $T(0) = \{0\}$, $T(1) = \{0, 15\}$, $T(2) = \{0, 7, 14, 15\}$, $T(3) = \{0, 1, 7, 14, 15\}$, and $T(4) = \{0, 1, 2, 3, 4, 5, 7, 8, 14, 15\}$.

2.3.2 F*D integer constraints

Thus far we have developed integer constraints for frequency constrained problems only. Nothing prevents us from extending the concept to frequency-distance (F*D) constrained problems. In Section 2.1 we developed the F*D constraint: if $D(v_i, v_j) > d(i, j)$, then $f(v_i) \neq f(v_j)$, where $d(i, j)$ is the minimum physical distance required between transmitters v_i and v_j and $D(v_i, v_j)$ is the actual distance between the transmitters. In general, define $T(k)$, $k = 0, 1, \dots, K$ such that

$$D(v_i, v_j) > d(i, j) \rightarrow |f(v_i) - f(v_j)| \notin T(k)$$

then, $R(k) = [T(k), d(k)]$ is the k^{th} general F*D constraint. Clearly, the constraint form $R(k)$ works equally well for F*D constrained problems (where $d(i, j)$ is a minimum physical distance) and frequency constrained problems (where $d(k)$ is a minimum frequency separation or distance).

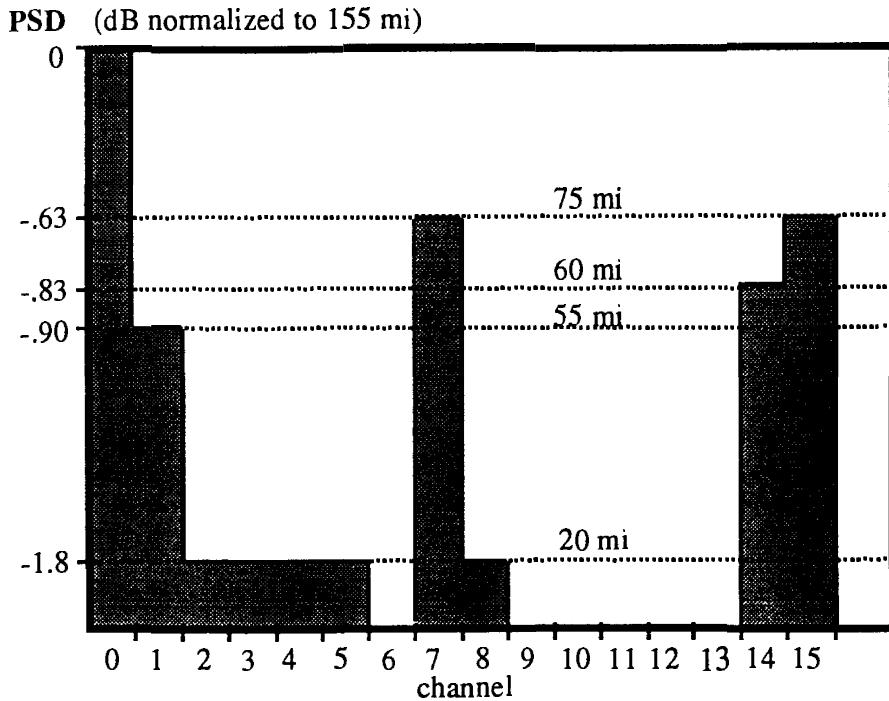


Figure 6: Power Spectrum of UHF Taboo Problem

2.3.3 Channel separation matrix

For some instances, we may not wish to specify d , but instead only provide the infeasible channel separations for pairs of transmitters. For such cases Hale defined an even more general constraint construct, the channel separation matrix, $t \in Z_+^{|V| \times |V|}$ such that,

$$t(i, j) = t(j, i), \quad t(i, i) = \{0\}, \quad i, j \in V, \quad i \neq j$$

consequently,

$$\text{if } |f(v_i) - f(v_j)| \notin t(i, j), \quad \forall v_i, v_j \in V, \rightarrow f \text{ feasible for } V, t.$$

It is trivial to show that $t(i, j)$ generalizes $R(k)$; for if $R(k) = [T(k), d(k)]$, let $t(i, j) = T(k), \forall i \neq j$.

3 Graph Theory

For references on graph theoretical definitions and notation, we use [33, 48, 52].

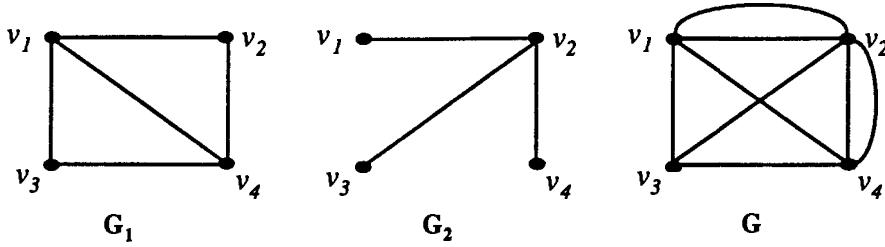


Figure 7: A 2-level Family of Graphs and Equivalent Multigraph

3.1 Graphs

A graph $G(V, E)$ is a pair of finite sets, where the set V contains the *vertices* of the graph and the set E contains distinct unordered pairs of vertices called *edges*. Vertices $(x, y), x, y \in V$ are said to be *adjacent* if they are connected by an edge, that is if $(x, y) \in E$ and the edge (x, y) is said to be *incident* to vertices x and y . The *degree* $d(x)$ of a vertex x is the number of edges incident upon it. The minimum degree of the graph $G(V, E)$ is defined as $\delta(G) = \min_{x \in V(G)} \{d(x)\}$ and the maximum degree is defined as $\Delta(G) = \max_{x \in V(G)} \{d(x)\}$.

The *complement* \bar{G} of a graph $G(V, E)$ has vertex set $V(G)$ and edge $(x, y) \in E(\bar{G})$ if and only if $(x, y) \notin E(G) \forall x \neq y \in V(G)$.

A *multigraph* is a graph without the restriction that the edges be distinct. That is, a pair of vertices may have more than one edge connecting them. If the edge set E of a multigraph is partitioned into K distinct sets, then we have a multigraph that is a family of K graphs G_1, G_2, \dots, G_K on the same vertex set which we represent by $G(V, G_1, G_2, \dots, G_K)$. An example of a family of graphs $G(V, G_1, G_2)$ and the equivalent multigraph G is depicted in Figure 7.

A *subgraph* H of graph G is a graph that has all of its vertices and all of its edges contained in the graph G : $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

An *induced subgraph* H of graph G is a subgraph with vertices $V(H) \subseteq V(G)$ and edges $E(H)$ being exactly those that are incident to all $v \in V(H)$ in the graph G .

Graph $G(V, E)$ is said to be *homomorphic* to graph H if there exists some function $f : (x, y) \in E(G) \rightarrow (f(x), f(y)) \in E(H) \forall x \neq y \in V(G)$. Graph G is said to be *isomorphic* to graph H if G is homomorphic to H and $|V(G)| = |V(H)|$ and $|E(G)| = |E(H)|$, that is, f is a one-to-one mapping.

A *directed* or *oriented* graph $G(V, A)$ is a pair of finite sets, where the set V contains the *vertices* of the graph and the set A contains distinct *ordered* pairs of vertices called *arcs*. Thus if $(v_i, v_j) \in A(G)$ then an arc is drawn as an arrow from vertex v_i to v_j .

3.2 Graph Operations

Consider two disjoint graphs $G(V, E)$ and $H(V, E)$ with $|V(G)| = n$ and $|V(H)| = m$. The Cartesian *product of vertices* of graphs G and H is denoted by $V(G) \times V(H)$ and results in nm vertices, each labeled $\{g_i, h_k\}$, where $g_i \in V(G)$ and $h_k \in V(H)$. The *product of graphs* G and H , denoted by $G \times H$ is defined as the graph with vertices $V(G) \times V(H)$ and edges

$$(\{g_i, h_k\}, \{g_j, h_l\}) \in E(G \times H) \longleftrightarrow k = l \text{ and } (g_i, g_j) \in E(G) \\ \text{or } i = j \text{ and } (h_k, h_l) \in E(H).$$

The *lexicographic product* of G with H , denoted by $G[H]$, is the graph with vertices $V(G) \times V(H)$ and edges

$$(\{g_i, h_k\}, \{g_j, h_l\}) \in E(G[H]) \longleftrightarrow (g_i, g_j) \in E(G) \\ \text{or } i = j \text{ and } (h_k, h_l) \in E(H).$$

3.3 Special Graphs

A *path* is an ordered sequence of distinct vertices and edges in G such that each edge is incident to the vertices immediately before and after it. A graph that is a path of n vertices is denoted P_n . A path that begins and ends at the same vertex is termed a *cycle* or *circuit* and a graph that is a cycle of n vertices is denoted by C_n . A graph that contains no induced subgraph that is a cycle of length 4 or more is called *chordal* or *triangulated*.

A graph is called *connected* if every pair of vertices in G are joined by a path. A path that includes every vertex in V exactly once is called a *Hamiltonian path*. A Hamiltonian path that begins and ends at the same vertex is called a *Hamiltonian cycle* and a graph with a Hamiltonian cycle is called *Hamiltonian*.

An *r -path* is a sequence of n vertices, $r < n$, where $v_i, v_{i+1}, \dots, v_{i+r}$ form a complete induced subgraph for all $i = 1, 2, \dots, m - r$ and no other edges are permitted. Notice that a 1-path is an ordinary path. A *Hamiltonian r -path* of a graph is an r -path that uses all of the vertices in graph G .

A graph with n vertices for which every vertex is adjacent to every other vertex in V is termed *complete* and is denoted by K_n . The complement of a complete graph is called an *independent graph* and denoted by I_n .

A *forbidden difference graph* termed $\tau(V, E)$ was defined by Tesman [85] in relation to the frequency assignment problem: let $V(\tau) = \{0, 1, 2, 3, \dots\}$ and $(x, y) \in E(\tau)$ if and only if $|x - y| \in T$ where T is any finite set of positive integers. Notice that this graph has an infinite number of vertices. A forbidden difference graph with *order* m , denoted by $\tau|_m$ is a finite subgraph of τ induced by the first m vertices that is, $V = \{0, 1, 2, 3, \dots, m\}$ with induced edge set.

The *T-graph* G_T was introduced by Liu [62], also with regard to the frequency assignment problem, and is the complement of the forbidden difference graph. Likewise, the *T-graph* of order m G_T^m is the subgraph of G_T induced by the first m vertices. The utility of the forbidden difference graphs and *T-graphs* will become apparent in Section 4.2 when we discuss graph coloring models for the frequency assignment problem.

3.4 Colorings and Cliques

We will define the *coloring* of a graph $G(V, E)$ to be an assignment of colors to the vertices V such that no two adjacent vertices (i.e. vertices connected by an edge) have the same color. Colors are typically defined to be the set of nonnegative integers. A coloring that assigns c colors to G is termed a *c-coloring*. The chromatic number of a graph G is the minimum number $\chi(G)$ for which a $\chi(G)$ -coloring exists for G . A graph is *c-colorable* if $\chi(G) \leq c$. A graph is *c-chromatic* if $\chi(G) = c$. For coloring a graph G , we will require that G be connected.

A complete subgraph of G is called a *clique* and a clique that will no longer remain complete if any vertex is added to the subgraph is called a *maximal clique* of G . The number of nodes in a maximal clique is called the *clique number*, and is denoted by $w(G)$. When all the vertices adjacent to a vertex x form a clique, then x is called a *simplicial vertex*. The complement of a clique is an *independent subgraph* and the complement of a maximal clique is the *maximal independent subgraph*. The number of vertices in a maximal independent subgraph is denoted by $\alpha(G)$.

Typically, $\chi(G) \geq w(G)$. However, a graph is *weakly γ -perfect* if $\chi(G) = w(G)$ and *perfect* if $\chi(H) = w(H)$ for all induced subgraphs H of G . Weakly γ -perfect graphs include all complete graphs. All weakly γ -perfect graphs are also perfect. Finally, all chordal graphs are perfect.



Figure 8: An Obstruction In A Directed Graph

3.5 Vertex Orderings and Their Implied Graphs

Consider the vertices of a graph $V(G) = \{v_1, v_2, v_3, \dots, v_n\}$. A permutation of the vertices $V_\pi = \{v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(n)}\}$ is called a *vertex ordering*. An ordering is called *natural* if $V_\pi = V$.

An ordering is termed *compatible* if and only if when $i < j < k$ and $(v_i, v_k) \in E(G)$ then $(v_i, v_j) \in E(G)$ and $(v_j, v_k) \in E(G)$.

A vertex ordering is called a *perfect elimination ordering* or *p.e.o.* if and only if $v_{\pi(i)}$, $i = 1, 2, \dots, n$ is a simplicial vertex in the induced subgraph of vertex set $\{v_{\pi(i)}, v_{\pi(i+1)}, \dots, v_{\pi(n)}\}$ (recall that a simplicial vertex has the property that all adjacent vertices form a clique). The *reverse of p.e.o.* is found simply by reversing the order of the p.e.o.

A vertex order is called *perfect* if the span of every induced subgraph H in G is $\chi(H)$ as obtained by the *greedy* algorithm on the vertex order (see Section 7.2 for information on the greedy algorithm).

Suppose a directed graph is created by transforming all the edges of a graph into arcs such that if $i < j$ and $(v_i, v_j) \in E(G)$ then $(v_i, v_j) \in A(G)$. A vertex ordering is called *admissible* if the directed graph thus created has no *obstructions*, where an obstruction is an induced subgraph of the form shown in Figure 8.

Vertex orderings will prove very useful when designing sequential algorithms (Section 7.1). Vertex orderings are also useful as conditions for the existence of some special graphs, which we describe below. Consider the graph G with vertex set $V(G)$ and a function f that maps the vertices of the graph to points in Euclidean j -space: $f(x) \in \Re^j, \forall x \in V(G)$. Define the edge set $E(G)$ such that

$$(x, y) \in E(G) \iff \|f(x) - f(y)\| \leq d,$$

where $d \in \Re_+$ and $\|\cdot\|$ denotes the Euclidean distance operation. The graph G is called a *j-unit sphere graph*. If we let the vertices of the graph be transmitters, $j = 1, 2$ or 3 , $f(x)$ the physical location of transmitter x , and

d a minimum distance between any two transmitters for operation without interference, we see that graph G models the frequency-distance constraints defined in Section 2.1.

When $j = 2$, G is called a *unit disk* graph which, for the FAP, represents transmitters distributed in a plane. When $j = 1$, G is called an *indifference* graph which, for the FAP, represents transmitters distributed on a line. Roberts [73] showed that a graph is an indifference graph, if and only if the vertices of the graph admit an ordering which is *compatible*. Similar characterizations of unit disk graphs remains an open problem. Related work can be found in [28, 29].

Fulkerson and Gross [35] demonstrated that a graph G is *chordal*, as defined in Section 3.3, if and only if the vertices of the graph admit the *reverse of a p.e.o.* Likewise, the graph G^c is the complement of a chordal graph, if and only if there exists a p.e.o. on G .

A graph G is called *perfectly orderable* if the vertices of the graph admit a perfect order. A stronger condition was pointed out by Chvátal [19] that a graph is perfectly orderable if and only if the vertices of the graph admit an *admissible* order.

Chvátal [19] showed that all chordal graphs, complements of chordal graphs and transitively orderable graphs are perfectly orderable graphs. Furthermore, an admissible ordering of a chordal graph is always the reverse of a p.e.o. and an admissible ordering of the complement of a chordal graph is a p.e.o. of the uncomplemented graph. Likewise, Raychaudhuri [71] showed that all indifference graphs are chordal and the reverse of a p.e.o. of an indifference graph is always a compatible ordering. Clearly, perfectly orderable graphs generalize chordal graphs which in turn generalize indifference graphs.

These graphs play an important role in the frequency assignment problem as will be seen in Section 7.2, since a greedy algorithm will always find an optimal frequency assignment (in some sense) if the frequency constraints of the problem can be modeled as one of these graphs.

We will next show how frequency assignment problem constraints can be formulated in graph theoretical terms.

4 Graph Formulations

4.1 Co-Channel and Adjacent Channel Frequency Assignments

Define the vertices V of graph G to be the set of all transmitters in the network of interest. Then define an edge (v_i, v_j) for G if transmitters v_i and v_j cannot use the same channel (are co-channel constrained). The *co-channel* FAP is equivalent to a simple graph coloring,

$$\begin{aligned} \text{Instance: } & G(V, E) \\ \text{find } f: & V \rightarrow \mathbb{Z}_+ \text{ s.t. } (v_i, v_j) \in E \longleftrightarrow f(v_i) \neq f(v_j) \end{aligned}$$

The fact that co-channel frequency assignment is equivalent to a graph coloring problem was first pointed out by Metzger [64]. Metzger's objective was to minimize the number of frequencies (colors). Hence Metzger's co-channel FAP is:

$$\begin{aligned} \text{Instance: } & G(V, E) \\ \text{find } & \chi(G). \end{aligned} \tag{1}$$

Hale [46] expanded upon Metzger's work adding some fundamental results. Hale recognized that there are two figures of merit for any feasible frequency assignment; the discrete number of frequencies used in the assignment, which he called *order*, and the largest frequency assigned minus the smallest frequency assigned, which he called *span*. Hence Metzger's formulation is that of a minimum order co-channel frequency assignment problem. Clearly, the minimum order of a co-channel problem is simply $\chi(G)$. Hale then proved the following results:

Theorem 4.1 (Hale [46]). *For any frequency assignment, $\text{order} \leq \text{span} + 1$.*

Theorem 4.2 (Hale [46]). *For the co-channel constrained assignment, $\text{minimum span} + 1 = \text{minimum order}$.*

This shows that if we have a method of assigning the gaps or “holes” in the spectrum of a minimum order assignment to another system, minimum span assignments will typically waste spectrum. It appears that for adjacent channel assignment problems there exists a tradeoff: we may minimize the order but only at the expense of making the span much larger than optimal. The converse is also true. This tradeoff was first mentioned by Hale [46] and later by Cozzens and Roberts [21]. Hale and later Raychaudhuri [72] and

Roberts [76] point out the need to find what they call *restricted* colorings. However, very little work has been done on this area. Restricted colorings attempt to minimize the span (order) while the order (span) is fixed. A special case of restricted colorings occurs when we attempt to minimize the span (order) for order equal to the minimum order (span equal to the minimum span).

Theorem 4.3 (Hale [46]). *The minimum order of a general channel constrained assignment is bounded below by the minimum span of the co-channel constrained assignment.*

4.2 *T*-Colorings and Bounds on Minimum Order and Span

In [21], Cozzens and Roberts recognized that the general channel constrained frequency assignment problem is equivalent to a special type of coloring for a multigraph. In section 2.3.1 we defined general channel constraints $R(k) = [T(k), d(k)]$, $k = 1, \dots, K$. We may define a multigraph, $G(V, G_0, G_1, \dots, G_K)$ such that for any two distinct vertices $x, y \in V$, there is an edge in G_k between x and y if and only if the absolute difference between the frequencies assigned to x and y are not contained in $T(k)$:

$$(x, y) \in E(G_k) \iff |f(x) - f(y)| \notin T(k).$$

By section 2.3.1, $0 \in T(0) \subset T(1) \subset \dots \subset T(K)$, and we require that the multigraph be *nested*; $G_0 \supset G_1 \supset \dots \supset G_K$.

For any graph $G(V, E)$ and any finite set of integers T , a *T-coloring* of G is

$$\begin{aligned} f : V &\rightarrow \mathbb{Z}_+ \setminus \{0\} \\ (x, y) \in E(G) &\iff |f(x) - f(y)| \notin T \\ \forall x, y \in V, x &\neq y \end{aligned} \tag{2}$$

and the *T-coloring* of a multigraph $G(V, G_0, G_1, \dots, G_K)$ is

$$\begin{aligned} f : V &\rightarrow \mathbb{Z}_+ \setminus \{0\} \\ (x, y) \in E(G_k) &\iff |f(x) - f(y)| \notin T(k) \\ \forall (x, y) \in V, x &\neq y, \quad k = 0, 1, \dots, K. \end{aligned} \tag{3}$$

Clearly, the general channel constrained frequency assignment problem of Section 2.3.1 is a *T-coloring* of a multigraph.

Cozzens and Roberts refer to the minimum order and minimum span for the *T-coloring* of graph G as the *T-order* and *T-span* and denote each

by $\chi_T(G)$ and $spt(G)$ respectively. They also defined a third optimality criteria, the *T-edge span* to be the minimum span taken over only the edges of G (the span is taken over all vertices of G) and is denoted by $esp_T(G)$. The significance of the *T-edge span* will be explained later in this section. Several basic results regarding the 3 optimality criteria were proven by Cozzens and Roberts and are presented below.

Theorem 4.4 (Cozzens and Roberts [21]). *For any graph G and any T -set such that $0 \in T$,*

$$\chi_T(G) = \chi(G).$$

Theorem 4.4 points out the fact that the order of any T -coloring of any graph is just the chromatic number of the graph. Since finding the chromatic number of a graph has been so well studied, [52], emphasis in subsequent T -coloring research has been placed on finding good bounds for the T -span and T -edge span.

Theorem 4.5 (Cozzens and Roberts [21]). *If r is the maximum element in T , then*

$$\chi(G) - 1 \leq esp_T(G) \leq spt(G) \leq (r + 1)[\chi(G) - 1].$$

The upper bound was improved by Tesman [84, 85] as

Theorem 4.6 (Tesman [85]). *Let $t = |T|$. Then*

$$\chi(G) - 1 \leq spt(G) \leq t(\chi(G) - 1).$$

Only when $T = \{0, 1, \dots, r\}$ does Theorem 4.6 become Theorem 4.4. Clearly, Theorem 4.6 offers an improved upper bound.

Cozzens and Roberts also proved the following important result:

Theorem 4.7 (Cozzens and Roberts [21]). *For any graph G and any T -set T ,*

$$spt(K_{w(G)}) \leq esp_T(G) \leq spt(G) \leq spt(K_{\chi(G)}).$$

Now we can see the significance of the T -edge span for frequency assignment problems. For if $esp_T(G)$ can be found, a tighter lower bound on $spt(G)$ might be had. In fact, the T -edge span for the co-channel FAP (i.e. $T = \{0\}$) is equivalent to the *bandwidth* of a graph [68].

When certain assumptions about the structure of the graph G are made, Cozzens and Roberts show that an exact value for the T -span and T -edge span can be found as seen below.

Corollary 4.8 (Cozzens and Roberts [21]). *If G is weakly γ -perfect then*

$$esp_T(G) = sp_T(G) = sp_T(K_{\chi(G)})$$

With Theorem 4.7 and Corollary 4.8 Cozzens and Roberts underline the fundamental need of finding the T -span of complete graphs as a prerequisite to bounding the T -span of a frequency assignment. Indeed, since [21], several papers and Ph.D. theses have identified special T -sets that have closed form solutions for the T -span of a complete graph. These results are presented below.

Cozzens and Roberts [21] presented an expression for the T -span and T -edge span of a complete graph for *consecutive* T -sets.

Theorem 4.9 (Cozzens and Roberts [21]). *If T is a consecutive set, that is $T = \{0, 1, 2, \dots, r\}$ then*

(i) *for any graph G ,*

$$sp_T(K_n) = (r+1)[n-1] \text{ hence, } (r+1)[w(G)-1] \leq esp_T(G) \leq sp_T(G) \leq (r+1)[\chi(G)-1],$$

(ii) *for G weakly γ -perfect,*

$$sp_T(G) = esp_T(G) = (r+1)[\chi(G)-1],$$

(iii) *for G perfect, $sp_T(G)$ and $esp_T(G)$ may be computed in $O(n^2)$ time.*

The proof of part (iii) follows from the fact that $\chi(G)$ is computable for a perfect graph in $O(n^2)$ time [45], that all perfect graphs are also weakly γ -perfect and finally by part (ii). Later we shall see that in general, computing the optimal T -span or T -order of a graph is a hard problem. Cozzens and Roberts [21] generalize Theorem 4.9 to *r-initial* T -sets.

Theorem 4.10 (Cozzens and Roberts [21]). *Suppose T is r-initial, that is, $T = \{0, 1, 2, \dots, r\} \cup S$ where $n(r+1) \notin S$, $n = 1, 2, \dots$*

(i) *For any graph G ,*

$$sp_T(G) = sp_T(K_{\chi(G)}) = (r+1)[\chi(G)-1].$$

(ii) *For G perfect, $sp_T(G)$ and $esp_T(G)$ may be computed in $O(n^2)$ time.*

Raychaudhuri [71, 72] defines *k-multiple of s T-sets* as $T = \{0, s, 2s, \dots, ks\} \cup S$ where $S \subseteq \{s+1, \dots, ks-1\}$. Note that some of the UHF Taboo T-sets in Example 2.3 fit this form as do T-sets with harmonic sidebands, making them quite practical.

Theorem 4.11 (Raychaudhuri [71]). *Suppose T is a k-multiple of s set. Then for any graph G*

$$(i) \ sp_T(G) = sp_T(K_{\chi(G)})$$

(ii)

$$sp_T(K_n) = \begin{cases} sp + skp - sk - 1 & \text{if } n = sp, p \in Z_+ \setminus \{0\} \\ sp + skp + l - 1 & \text{if } n = sp + l, l \in \{1, 2, \dots, s-1\}. \end{cases}$$

$$(iii) \ sp_T(K_n) = sd(k+1) + q - 1, \ m = ds + q, \ d \geq 0, \ q \geq 1.$$

Theorem 4.11 (i) and (ii) provide an exact expression for the T-span by letting $n = \chi(G)$. Expression (iii), introduced by Liu [62], is an alternate way to state (ii).

The previous two theorems find an exact value for the T-span and bounds on the T-edge span. When an exact value for the T-span exists for any graph, we will say that this T-set has Property 1.

Property 1.

$$sp_T(G) = sp_T(K_{\chi(G)}), \text{ for all graphs } G.$$

Liu [62] developed a sufficient condition for determining when a T-set has Property 1.

Theorem 4.12 (Liu [62]). *Consider a T-set T with T-graph G_T . Then T has Property 1 if and only if G_T^n is weakly perfect for all $n \in Z_+ \setminus \{0\}$.*

Liu [62] also developed a method for finding $sp_T(K_n)$ for any T-set:

Theorem 4.13 (Liu [62]). *For any T-set T with T-graph G_T ,*

$$sp_T(K_n) = \min\{\alpha : w(G_T^\alpha) = n\} - 1,$$

where $w(G_T^\alpha)$ is the clique number of the α order T -graph. Based upon these results, Liu developed some T -sets beyond the r-initial and k-multiple of s sets that have Property 1.

Theorem 4.14 (Liu [62]). Define the extended k-multiple of s T -set as: $T = \{0, s, 2s, \dots, ks\} \cup S \cup S'$ where, $S \subseteq \{s+1, \dots, ks-1\}$ and $S' = \{\sigma \in Z_+ : \sigma \geq s(k+2), \sigma \notin [(s(2k+1)+1, s(2k+3)-1] \cup [s(3k+2)+1, s(3k+4)-1] \cup [s(4k+3)+1, s(4k+5)-1] \cup \dots\}$. For any graph G , if T -set T is an extended k-multiple of s set, then T has Property 1 and Theorem 4.11 holds.

Theorem 4.15 (Liu [62]). Let $N_p = \{p, 2p, \dots\}$. If $T = ([0, a+b] \setminus \{a+1\}) \cup S$, where $a = cp$, $c \geq 1$, $p \geq 2$, $b \geq 2$, $i(a+1) \notin N_p$ and $(a+b+1) + i(a+1) \notin N_p$ for all $i = 0, 1, \dots, p-1$ and $S = \{\sigma : \sigma < pa + p + b\}$ such that $N_p \cap [0, pa + p + b - 1] \subseteq S$, $i(a+1) \notin S$, $(a+b+1) + i(a+1) \notin S$, $i = 0, 1, 2, \dots, p-1$, then T has Property 1 and

$$spt_T(K_n) = k(pa + p + b) + (l-1)(a+1), \quad n = kp + l, \quad k \geq 0, \quad 1 \leq l \leq p.$$

Theorem 4.16 (Liu [62]). If $T = \{0, s, 2s, \dots, ks, (k+2)s, (k+3)s, \dots, (3k+2)s\} \cup R$, where $R \subseteq [s+1, ks-1] \cup [(k+2)s+1, (3k+2)s-1]$, then T has Property 1.

Theorem 4.17 (Liu [62]). If $T = \{0\} \cup [s, l]$ then T has Property 1 if and only if $l = ms$ for any $m \in Z_+ \setminus \{0\}$.

Note that a general sufficiency condition that defines for all T -sets those that have Property 1 remains an open problem.

The remaining results introduce special T -sets that do not find an exact value for the T -span. However, they do provide bounds on the T -span and the T -edge span through use of Theorem 4.7 and knowledge of the clique and chromatic numbers of G .

Theorem 4.18 (Cozzens and Wang [24]). Given the complete graph K_n on n vertices, and $T = \{0, 1, 2, \dots, r, p(r+1)\} \cup S$, where p is the smallest integer strictly greater than 1 such that $p(r+1) \in T$, and the following is always true:

For all $l_1, l_2, \beta \in Z_+ \setminus \{0\}$;

if $\beta \leq \frac{n}{p}$ and $l_1, l_2 < p$: $\{(p\beta + l_1 - l_2)(r+1) + \beta\} \notin S$

or if $\beta < \frac{n-p}{p}$ and $l_1 < p$: $\{(p\beta - l_1)(r+1) + \beta - 1\} \notin S$

or if $\beta \leq \frac{n-p}{p}$ and $l_1 < p$: $\{(p\beta + l_1)(r+1) + \beta + 1\} \notin S$

or if $p > l_1 > l_2$: $l_1(r+1) \cup (l_1 - l_2)(r+1) \notin S$

then

$$sp_T(K_n) = \begin{cases} (n-1)(r+1) + (k-1) & \text{if } n = kp \\ (n-1)(r+1) + k & \text{if } n = kp + \lambda, 1 \leq \lambda \leq p. \end{cases}$$

Additional results that depend upon the value of n are in Wang [89]. In a similar manner, Tesman developed the following results.

Theorem 4.19 (Tesman [84]). *If $T = \{0, r, 2r, 2r+1\}$ and $n = ar + j$, $a, r, j \in Z_+$ then*

$$sp_T(K_n) = \begin{cases} 4(a-1) & r=1, j=0 \\ (3a-2)r + (a-2) & a \leq r+1 \text{ and } r \geq 2, j=0 \\ (3a-1)r - 1 & a > r+1 \text{ and } r \geq 2, j=0 \\ 3ar + (j-1) & 1 \leq j \leq r-1. \end{cases}$$

Notice that with $r = 7$, Theorem 4.19 yields the UHF taboo T -set $\{0, 7, 14, 15\}$ of Example 2.3.

Theorem 4.20 (Tesman [84]). *If $T = \{0, r, r+1, \dots, kr+l\}$ and $n = ar + b$, $1 \leq b \leq r$, $a, b, r, l \in Z_+$ then*

$$sp_T(K_n) = (k+1)ar + al + b - 1.$$

Theorem 4.21 (Tesman [84]). *If $T = \{0, r, r+1, 2r+1\}$ and $n = ar + b$, $1 \leq b \leq r$, $a, b, r \in Z_+$ then*

$$sp_T(K_n) = (3r+1)a + b - 1.$$

Some other bounds on $sp_T(G)$ for $T = \{0, 1, 2, \dots, m\} \setminus \{k\}$, $1 \leq k \leq m$, $m \geq 2$ for various values of k and m are given in Eggleton, Erdős, and Skilton [29]. Bonias [11] described a process for finding a T -coloring with either $sp_T(K_n)$ or approximately close to it by expressing the T -span of a complete graph in terms of the T -span of another, smaller complete graph.

Theorem 4.22 (Bonias [11]). *For any T , and $n, m, p \in Z_+$,*

$$sp_T(K_{mp+1}) \geq m \cdot sp_T(K_{p+1}).$$

Theorem 4.23 (Bonias [11]). *For any T , and $n, p \in Z_+$: $n > p$,*

$$sp_T(K_n) \geq sp_T(K_{n-p}) + sp_T(K_{p+1}).$$

Using this same line of thought, Bonias developed a recursion relation for expressing the color assigned to a vertex of a complete graph as a function of the colors already assigned. The recursion is developed as follows. For any T -set T , let D be the complement of T , that is, the *allowed* channel separations. Let $S = \{s_1, s_2, \dots, s_p\}$ such that $s_i \in D$. Let the vertices of K_n be ordered arbitrarily as $v_1, v_2, \dots, v_i, \dots, v_n$ and the color $f(v_1) = 1$. The recursion relation for $i > 1$ is defined as:

$$f(x_i) = f(x_{i-1}) + s_{(i-2) \bmod(p)+1} \quad (4)$$

so that the following sequence results:

$$\begin{aligned} f(x_1) &= 1 \\ f(x_2) &= 1 + s_1 \\ &\vdots \\ f(x_{p+1}) &= 1 + s_1 + \dots + s_p \\ f(x_{p+2}) &= 1 + s_1 + \dots + s_p + s_1 \\ &\vdots \end{aligned}$$

The recursion is used in the following definition.

Definition 4.1. *Let $t_{max} = \max\{T\}$. The sequence S is called a T -admissible sequence and the coloring $F_S = \{1, (1 + s_1), \dots, (1 + s_1 + \dots + s_p), \dots\}$ is called a T -admissible coloring if the sequence S forms a feasible T -coloring of K_n that is exactly F_S by applying the recursion relation, and:*

- (i) *if $\sum_{i=1}^p \leq t_{max}$ then the following must be true for $S^c = \{\sigma_1, \sigma_2, \dots, \sigma_{cp}\}$, where S^c is the sequence obtained by concatenating S with itself c times and $c = \lceil t_{max} + \frac{1}{\sum_{i=1}^p s_i} \rceil$:*

$$\sigma_i + \sum_{j=i+1}^c p\sigma_j + \sum_{k=1}^m \notin T, \text{ for } i = 3, 4, \dots, cp, \text{ and } m = 1, \dots, i-2$$

(ii) else, if $\sum_{i=1}^p > t_{\max}$ then, in addition to the above, the following must also be true:

$$\sigma_i + \sum_{j=i+1}^k \sigma_j \notin T, \text{ for } i = 1, 2, \dots, cp-1, \text{ and } k = 1, 2, \dots, cp.$$

This shows that it is straightforward to check if a sequence is T -admissible. This has significance due to the following theorem.

Theorem 4.24 (Bonias [11]). *For any T -set T and graph K_n , let*

$$S = \{s_1, s_2, \dots, s_p\}$$

be a T -admissible sequence and denote the span of T -admissible coloring F_S by $\text{span}_T(K_n, F_S)$. Let $\Delta s_i = \text{span}_T(K_i, F_S) - spt(K_i)$ for $i = 2, 3, \dots, p+1$. If $\Delta s_{p+1} = 0$, and

(i) if $(n-1) \bmod (p) = 0$, then for all n

$$spt(K_n) = \text{span}_T(K_n, F_S)$$

(ii) else

$$spt(K_n) \geq \text{span}_T(K_n, F_S) - \Delta s_{(n-1) \bmod (p)+1}.$$

Theorem 4.24 says that if we can find a T -admissible sequence with a small value of p such that $\Delta s_{p+1} = 0$, then by using the recursion relation in equation (4), we can find a coloring with the minimum span or at least an upper bound on the minimum span.

Many more results have been obtained for T -colorings [10, 11, 32]. Excellent summaries of T -colorings are given by Roberts in [75, 76].

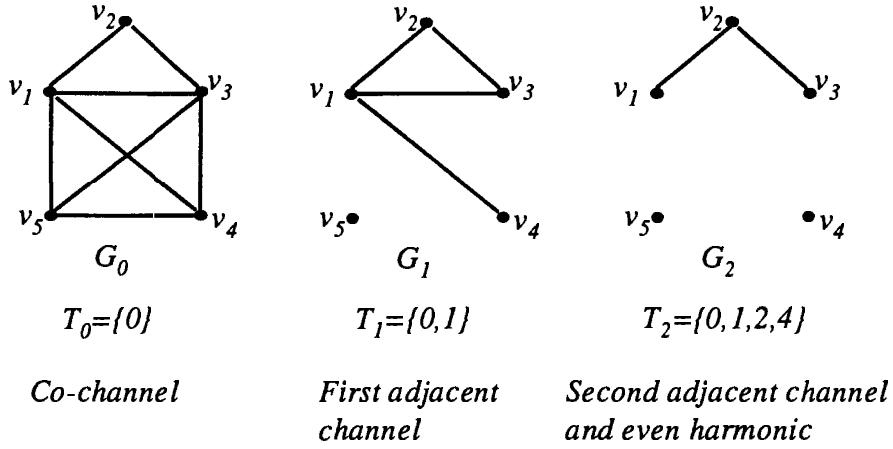


Figure 9: Example Of A Nested Multigraph

4.3 T -Colorings of Multigraphs

The results on T -coloring presented by Cozzens and Roberts [21] will now be extended to the general (and more practical) case introduced at the beginning of Section 4.2 where interference may occur on different “levels”. Here, the transmitters are the vertices of a multigraph and each interference level is represented by a separate set of edges on the common set of vertices.

We require that the $(K + 1)$ -level multigraph $G(V, G_0, G_1, \dots, G_K)$ be nested such that $G_0 \supseteq G_1 \supseteq \dots \supseteq G_K$ and that the T -sets $T(i)$ be reverse nested, that is satisfy $0 \in T(0) \subseteq T(1) \subseteq \dots \subseteq T(K)$. An example of a nested multigraph with associated T -sets is shown in Figure 9. Graph G_0 represents the interference constraints with forbidden channel separations in $T(0)$, graph G_1 represents the constraints with forbidden channel separations in T_1 , and so on.

Henceforth, we will assume that all multigraphs are nested and T -sets satisfy $0 \in T(0) \subseteq T(1) \subseteq \dots \subseteq T(K)$. A T -coloring of the multigraph G is restated from Equation (3) as

$$\begin{aligned}
 f : V &\rightarrow Z_+ \setminus \{0\} \\
 (x, y) \in E(G_k) &\iff |f(x) - f(y)| \notin T(k) \\
 \forall (x, y) \in V, x \neq y, \quad k &= 0, 1, \dots, K.
 \end{aligned}$$

The T -order, T -span, and T -edge span of multigraph G will be denoted

$\chi_T(G)$, $spt(G)$ and $esp_T(G)$ respectively and are respectively the minimum order, span, and edge span assignments that satisfy equation (3).

Cozzens and Wang [24] extended the T -coloring results of Theorems 4.4, 4.5 and 4.10 to the case of multigraphs.

Theorem 4.25 (Cozzens and Wang [24]). *Given multigraph*

$$G(V, G_0, G_1, \dots, G_K)$$

and any $(K + 1)$ -level T -set $T = \{T(0), T(1), \dots, T(K)\}$

$$(i) \quad \chi_T(G) = \chi(G_0)$$

$$(ii) \quad spt(G) \geq esp_T(G) \geq \chi(G_0) - 1$$

$$(iii) \quad \max_{0 \leq l \leq K} \{spt_{(l)}(G_l)\} \leq spt(G) \leq spt_{(K)}(G_0)$$

(iv) if each $T(k)$ is r_k -initial, $k = 0, 1, \dots, K$, then

$$\max_{0 \leq l \leq K} \{(r_l + 1)(\chi(G_l) - 1)\} \leq spt(G) \leq (r_K + 1)(\chi(G_0) - 1)$$

(v) if each $T(k)$ is r_k -initial, $k = 0, 1, \dots, K$ and $\chi(G_j) = \chi(G_0)$, $j = 1, \dots, K$, then

$$spt(G) = spt_{(K)}(G_0) = (r_K + 1)(\chi(G_0) - 1).$$

Hale [46] and others (see Lanfear [60]) have focused on the realistic and more tractable 2-level nested multigraph case, where $K = 1$ and $T(0) = \{0\}, T(1) = \{0, 1\}$. In such instances, the T -sets represent only co-channel and first adjacent channel constraints.

Raychaudhuri [71] developed the following intriguing result for 2-level multigraphs with the restriction that the co-channel graph G_0 be complete.

Theorem 4.26 (Raychaudhuri [71]). *Consider the 2-level nested multigraph*

$G(V, G_0, G_1)$, where G_0 is complete and $T(0) = \{0\}$, $T(1) = \{0, 1\}$. If the weighted graph G' is defined as: $V(G') = V(G)$, $E(G') = E(G_0)$, and every edge $(x, y) \in G'$ has a weight $w(x, y)$ assigned as

$$w(x, y) = \begin{cases} 2 & \text{if } (x, y) \in E(G_1) \\ 1 & \text{else} \end{cases}$$

then

$$spt(G) = \text{length of the shortest Hamiltonian path in } G',$$

where the length of a Hamiltonian path is the sum of the weights of the edges contained in the path.

Notice that if G_0 is complete then no two transmitters can share the same frequency, thus no frequency reuse is permitted. If we relax the unit disk condition to an indifference condition, then we obtain a 2-homogeneous indifference system. The following results apply to 2-homogeneous indifference systems for 2-level nested graphs, extending Theorem 4.26.

Theorem 4.27 (Tesman [85]). *Let multigraph $G(V, G_0, G_1)$ be a 2-homogeneous nested indifference system with $\chi(G_0) = p$, $\chi(G_1) = q$ and $T(0) = \{0\}$, $T(1) = \{0, 1\}$.*

(i) *For any G_0*

$$spt_T(G) = \begin{cases} 2q - 2 & \text{if } p = q \\ 2q, 2q - 1 \text{ or } 2q - 2 & \text{if } q + 1 \leq p \leq 2q - 1 \\ 2q \text{ or } 2q - 1 & \text{if } p = 2q \\ p - 1 & \text{if } p \geq 2q + 1 \end{cases}$$

(ii) *If G_0 is complete, then*

$$spt_T(G) = \begin{cases} 2q - 2 & \text{if } p = q \\ 2q - 1 \text{ or } 2q - 2 & \text{if } q + 1 \leq p \leq 2q - 1 \\ p - 1 & \text{if } p \geq 2q. \end{cases}$$

4.4 List Colorings and List T -Colorings

In many frequency assignment problems the frequencies that may be assigned to a vertex v_i are restricted to a finite set of integers $S(i)$ for $i = 1, 2, \dots, n$ where $n = |V|$. When the FAP is a co-channel assignment only, this situation is equivalent to a *list coloring*. Erdős, Rubin and Taylor [32] define a list coloring of graph as follows.

Definition 4.2. *Consider the graph $G(V, E)$ and finite sets of integers $S(i)$ for $i = 1, 2, \dots, n$, where $n = |V|$. A list coloring is defined as the rule f :*

$$\begin{aligned} V \rightarrow Z_+ \text{ s.t. } (v_i, v_j) \in E \longleftrightarrow & f(v_i) \neq f(v_j) \text{ and} \\ & f(v_i) \in S(i) \text{ for all } v_i \in V(G), \\ & S(i) \in Z_+ \setminus \{0\} \quad i = 1, 2, \dots, n. \end{aligned}$$

Erdős, Rubin and Taylor define a graph to be *k-choosable* if it may be list colored for every choice of $S(i) : |S(i)| = k$, $i = 1, 2, \dots, n$. The *choice number* $ch(G)$ of a graph G is the integer for which G is $ch(G)$ -choosable but not $(ch(G)-1)$ -choosable. Since the same list may be assigned to each vertex as $S(i) = \{1, 2, \dots, k\}$, $i = 1, 2, \dots, n$, k -choosability implies k -colorability as a special case and hence

$$ch(G) \geq \chi(G). \quad (5)$$

For certain cases the equality holds. For instance, when the graph is complete it is easy to see that $ch(G) = \chi(G)$. This is also true when the graph is chordal.

Theorem 4.28 (Erdős, Rubin and Taylor [32]). *For $G(V, E)$ chordal,*

$$ch(G) = \chi(G).$$

Bollobás and Harris [10] developed similar results for edge list colorings and defined an edge choice number. Tesman [84] extended list coloring concepts to T -colorings.

Definition 4.3. *Given any T -set T and list sets $S(i)$, $i = 1, 2, \dots, n$, an S -list T -coloring of graph $G(V, E)$ is the rule f :*

$$\begin{aligned} V \rightarrow Z_+ \text{ s.t. } (v_i, v_j) \in E \longleftrightarrow & |f(v_i) - f(v_j)| \notin T \text{ and} \\ & f(v_i) \in S(i), \text{ for all } v_i \in V(G), \\ & S(i) \in Z_+ \setminus \{0\} \quad i = 1, 2, \dots, n. \end{aligned}$$

Tesman defines a graph to be *T-k-choosable* if it may be S -colored for every choice of $S(i) : |S(i)| = k$, $i = 1, 2, \dots, n$. The *T-choice number* $T-ch(G)$ of a graph G is the integer for which G is $T-ch(G)$ -choosable but not $(T-ch(G) - 1)$ -choosable. Note that

$$T-ch(G) \geq ch(G) \quad (6)$$

and that the equality holds for $T = \{0\}$. Tesman then develops the following useful theorems.

Theorem 4.29 (Tesman [84]). *For all graphs $G(V, E)$ and T -sets T ,*

$$T-ch(G) \leq t(\Delta(G)) + 1,$$

where $\Delta(G)$ is the maximum degree of graph G .

Theorem 4.30 (Tesman [84]). *For chordal graphs $G(V, E)$ and any T -set T ,*

$$T\text{-}ch(G) \leq (2t - 1)(\chi(G) - 1) + 1,$$

where $t = |T|$.

Theorem 4.31 (Tesman [84]). *For complete graphs K_n and T -set*

$$T = \{0, s, 2s, \dots, ks\},$$

$$T\text{-}ch(K_n) = t(n - 1) + 1.$$

Tesman also found results for graphs that are trees and circuits which are described in [84].

4.5 Set Colorings and Set T -Colorings

In Example 2.2, each base station of a cellular network was assigned a set of channels to be used to communicate with users within its cell region without causing interference to neighboring cells. For the co-channel interference model, this situation is equivalent to *set coloring*. Networks that require multiple simultaneous links between some of the vertices or that need extra channels to maintain some guaranteed level of reliability are other examples of FAPs modeled by set colorings. We now define a set coloring.

Definition 4.4. *Consider the graph $G(V, E)$ and let $n = |V|$. A set coloring is defined as the rule R :*

$$\begin{aligned} V \rightarrow Z_+ \text{ s.t. } (v_i, v_j) \in E &\longleftrightarrow R(i) \cap R(j) = \emptyset \text{ where} \\ &R(i) \subset Z_+ \setminus \{0\} \quad i = 1, 2, \dots, n. \end{aligned}$$

where $R(i)$ is the set of colors that are assigned to vertex v_i .

A k -tuple coloring is a set coloring where $|R(i)| = k$ for all $i = 1, 2, \dots, n$. The k -tuple chromatic number, denoted by $\chi_k(G)$, is defined as the minimum number of distinct colors in a k -tuple coloring, that is

$$\chi_k(G) = \min \left\{ \left| \bigcup_{i=1,2,\dots,n} R(i) \right| \right\}.$$

Stahl [83] studies k-tuple colorings in detail and provides the following theorem.

Theorem 4.32 (Stahl [83]). *For any graph $G(V, E)$,*

$$\chi_k(G) = \chi(G[K_k])$$

where $G[K_k]$ is the lexicographic product of G with K_k .

Roberts [74] introduced the term set coloring with respect to frequency assignment and traffic phasing problems. Roberts defines many types of restrictions on general set colorings which he calls λ -colorings. The nature of the restrictions are represented by λ . For example, $\lambda = n$ represents the restriction that all sets $R(i)$ be of cardinality n , $\lambda = c$ means that each set must be a set of consecutive integers, $\lambda = \bar{n}$ implies that both restrictions n and c apply, $\lambda = I$ means that each set is defined over a real interval, and so on. Furthermore, Roberts defines the λ -chromatic number $\chi_\lambda(G)$ to be the λ -coloring of minimum cardinality (if sets are integer) or infimum measure (if sets are real intervals).

Note that the restriction $\lambda = n$ is equivalent to a n -tuple coloring and so $\chi_n(G)$ is simply the k-tuple chromatic number. Also notice that the restriction $\lambda = I$ requires a generalization of Definition 4.4 so that $R(i) \subset \mathbb{R}_+ \setminus \{0\}$ $i = 1, 2, \dots, n$. Roberts obtained good results for the restrictions $\lambda = n$ and $\lambda = \bar{n}$ as shown.

Theorem 4.33 (Roberts [74]). *If G is weakly γ -perfect, then $\chi_n(G) = n\chi(G)$.*

The restriction in Theorem 4.33 is $\lambda = n$, i.e., the sets must be of uniform cardinality, and the proof makes use of Stahl's Theorem 4.32. Roberts also has a result for when the sets are restricted as $\lambda = \bar{n}$ i.e., when they must be of uniform cardinality and each a consecutive set of integers.

Theorem 4.34 (Roberts [74]). *For any G , $\chi_{\bar{n}}(G) = n\chi(G)$.*

Opsut and Roberts [67] continue the work in [74] and develop linear programming models for some of the restrictions.

We have shown that T -colorings can model many realistic frequency assignment problems. For situations that require set assignments, it is natural to consider combining the ideas of set colorings and T -colorings. Tesman [84] defines *set T-colorings* as follows.

Definition 4.5. Consider the graph $G(V, E)$ with $n = |V|$ and T -set T . A set T -coloring is defined as the rule R :

$$\begin{aligned} V \rightarrow Z_+ \text{ s.t. } (v_i, v_j) \in E &\longleftrightarrow |r_l(i) - r_m(j)| \notin T \\ &\text{for all } r_l \in R(i) \text{ and all } r_m \in R(j), \text{ where} \\ &R(i) = \{r_1(i), r_2(i), \dots\} \subset Z_+ \setminus \{0\} \quad i = 1, 2, \dots, n. \end{aligned}$$

and $R(i)$ is the set of colors that are assigned to vertex v_i .

Tesman limits his research to considering only k -tuple sets. He then defines the minimum order $\chi_T^k(G)$ of a k -tuple T -coloring as the smallest number of distinct colors used in all assignments defined by Definition 4.5 with $|R(i)| = k$, and the minimum span $sp_T^k(G)$ as the smallest range of colors likewise used in all assignments.

Theorem 4.35 (Tesman [84, 86]). For all graphs G and all T -sets T

(i)

$$\chi_T^k(G) = \chi_k(G)$$

(ii) Let $m = \max T$. Then

$$sp_T^k(G) \leq (m + 1)(\chi_k(G) - 1),$$

and

$$sp_T^k(G) \leq (m + k)\chi(G) - (m + 1)$$

(iii)

$$sp_T^k(K_{w(G)}) \leq sp_T^k(G) \leq sp_T^k(K_{\chi(G)}).$$

If we further restrict each set to be a consecutive set of integers (which Roberts refers to as $\lambda = \bar{n}$), then a k -tuple set coloring is called a *consecutive k -tuple coloring* and subsequently, a *consecutive k -tuple T -coloring* follows from Definition 4.5. The following theorem extends 4.35.

Theorem 4.36 (Tesman [84, 86]). For all graphs G and consecutive T -sets $T = \{0, 1, \dots, m\}$,

(i) If $\chi(G) = 1$ then

$$sp_T^k(G) = k - 1$$

(ii) Else, if $\chi(G) \geq 2$, then

$$(m+1)(\chi(G)-1) + 2(k-1) \leq sp_T^k(G) \leq (m+k)\chi(G) - (m+1).$$

Theorem 4.36 was also obtained by Füredi, Griggs and Kleitman [36] and they further conjectured that all spans between the bounds in part (ii) may be found given a suitable graph. Füredi, Griggs and Kleitman refer to consecutive 2-tuple T -colorings as *pair labellings* and, in general, t-tuple T -colorings as *t-labellings* [36].

4.6 No-Hole Colorings

Roberts [77] considered colorings of graphs with $T = \{0, 1\}$ such that the colors assigned are a consecutive set of integers. Roberts termed such a coloring a *no-hole* coloring or N-coloring for short, and called the T -set a *2-distant* set since the frequencies assigned to vertices incident to a common edge in G must be different by at least two. In this section we examine existence conditions and optimality issues surrounding no-hole colorings for arbitrarily long consecutive T -sets. Later, a special no-hole set colorings will be reviewed, specifically the k-tuple T -coloring discussed in Section 4.5 where T is a consecutive set of integers.

4.6.1 N_r -Colorings

Troxell (Sakai) [78] and Troxell and Wang [79] generalized Robert's 2-distant no-hole concept to $(r+1)$ -distant T -sets: $T = \{0, 1, \dots, r\}$. Troxell called a coloring that uses all of the colors in T a *no-hole $(r+1)$ -distant coloring* termed N_r -coloring and developed the following results for them:

Theorem 4.37 (Troxell [78]). Consider graph $G(V, E)$ with $|V| = n$ and $(r+1)$ -distant T -set $T = \{0, 1, \dots, r\}$. If G has an N_r -coloring, then G has an N_r -coloring using all colors in the set $\{1, 2, 3, \dots, n\}$.

Theorem 4.38 (Troxell [78]). G has an N_r -coloring if and only if G^c has a Hamiltonian r -path.

We note that for $r = 1$ the results apply to 2-distant sets. Now we may establish conditions on when a hole in the spectrum must exist.

Theorem 4.39. Consider the multigraph $G(V, G_0, G_1, \dots, G_K)$ with $(r+1)$ -distant T -sets $T(k) = \{0, 1, \dots, r_k\}$, $k = 1, 2, \dots, K$, $r_1 < r_2 < \dots < r_K$. Define $\deg_G \stackrel{\Delta}{=} \max_v \left\{ \deg(v) : v \in V(G) \right\}$. If $\deg_{G_k} > n - (r_k + 1)$ for any k , then G is not N_{r_k} -colorable.

Proof. Suppose that a vertex v in G_k has $\deg(v) = n - (r_k + 1) + 1$. If we color v : $f(v) = 1$, then the $n - r_k$ vertices that are adjacent to v must, at best, be assigned colors $r_k + 2, r_k + 3, \dots, n, n + 1$. Consequently, a minimum of $n + 1$ colors are needed which is more colors than vertices. Clearly, a hole must result. \square

A simpler version (not a multigraph) of Theorem 4.39 was presented by Troxell in [78] and with a different proof. However, Theorem 4.39 generalizes Troxell's result. Note that the converse of Theorem 4.39 is not necessarily true. That is, if $\deg_{G_k} \leq n - (r_k + 1)$ for any k , then G still may not be N_{r_k} -colorable.

Suppose we are able to determine for some graph G that a no-hole coloring exists. The natural question which follows is, can we color G such that the span which results is $spt_T(G)$? Since an N_r -coloring implies $T = \{0, 1, \dots, r\}$, we know from Theorem 4.10 that $spt_T(G) = (r + 1)(\chi(G) - 1)$ for any G . However, even though an N_r -coloring may exist, it does not necessarily attain $spt_T(G)$. Furthermore, finding conditions for when a N_r -coloring attains $spt_T(G)$ is an open problem. However, Roberts [77] found that conditions could be established on certain graphs for *near-optimality*. Troxell [78] and Troxell and Wang [79] generalized Robert's results.

A N_r -coloring is said to be *near-optimal* if the span is bounded above by $spt_T(G) + r$. Using this definition, existence and near-optimality conditions for paths, circuits and bipartite graphs were established by Roberts [77], Troxell [78] and Troxell and Wang [79]. Additionally, results for indifference graphs were also found.

Theorem 4.40 (Troxell [78]). *Consider any indifference graph $G(V, E)$ and let $n = |V|$.*

- (i) *If $n = (r + 1)\chi(G) - r$ then G has an N_r -coloring if and only if G has a unique clique of size $\chi(G)$.*
- (ii) *If $n \leq (r + 1)(\chi(G) - 1)$ then G does not have an N_r -coloring. If $n \geq (r + 1)\chi(G)$, then G must have at least one N_r -coloring of which at least one is near-optimal.*

4.6.2 N_r^k -Colorings

In Section 4.5 we discussed frequency assignment problems which require a set of frequencies to be assigned to each vertex. Set-coloring and set

T -coloring were defined and theoretical results presented. If each set was of equal cardinality we spoke of k -tuple colorings and k -tuple T -colorings. Tuple set colorings are now extended to no-hole colorings. A *no-hole k -tuple $(r+1)$ -distant coloring*, denoted by N_r^k -colorings was defined by Troxell [78, 88]. We will use the terms order and span in the manner they were defined in Section 4.5. The following results generalize those of Theorems 4.37, 4.38, 4.39 and 4.40 respectively with the exception that Theorem 4.43 generalizes the similar result by Troxell in [78].

Theorem 4.41 (Troxell [78]). *Consider graph $G(V, E)$ with $|V| = n$ and $(r+1)$ -distant T -set $T = \{0, 1, \dots, r\}$. If G has an N_r^k -coloring, then G has an N_r^k -coloring using all colors in the set $\{1, 2, 3, \dots, kn\}$.*

For the next theorem, recall that $G[H]$ denotes the lexicographic product of G with H .

Theorem 4.42 (Troxell [78]). *Let I_k be the graph with k vertices and no edges. G has an N_r^k -coloring if and only if $(G[I_k])^c$ has a Hamiltonian r -path.*

Theorem 4.43 (Troxell [78]). *Consider a graph G with $n = |V(G)|$ and $(r+1)$ -distant T -set T . If G contains a complete q -partite subgraph H with $m = |V(H)|$ such that*

$$k(n - m) < (q - 1)r,$$

then G is not N_r^k -colorable.

Theorem 4.44 (Troxell [78]). *Consider any indifference graph $G(V, E)$ with $n = |V|$. If $kn \leq (r+k)\chi(G) - (r+1)$ then G does not have an N_r^k -coloring. If $n \geq \chi(G)(\lceil \frac{r}{k} \rceil)$, then G must have at least one N_r^k -coloring of which at least one is near-optimal.*

5 Integer Programming Formulations

In this section we cast frequency assignment problems in the context of readily recognizable integer programs (IP). The typical technique used for solving IPs is branch and bound (BB). It is well known that BB can require total enumeration of the problem so that, for large problems, IPs may not be practical for solution.

However, IPs do reveal structure of the FAP and also permit lower bounds on the optimal solution to be obtained via relaxation techniques (see Sections 7.6.1 and 7.6.2).

Baybars [3] developed an 0-1 integer program (IP) for a minimum order co-channel and adjacent-channel frequency assignment problem. Consider the 2-level multigraph $G(V, G_0, G_1)$ where $V = \{v_1, \dots, v_n\}$ is the set of vertices, G_0 is the co-channel subgraph with $T(0) = \{0\}$ and G_1 is the adjacent channel subgraph with $T(1) = \{0, 1\}$. Let I be the index set of the vertices of the multigraph such that $I = \{1, 2, \dots, n\}$. Define $F = \{1, 2, \dots, m\}$ to be a finite set of discrete colors (frequencies) and define the following binary decision variables:

$$x_{if} = \begin{cases} 1 & \text{if vertex (transmitter) } i \text{ assigned color (frequency) } f \\ 0 & \text{else,} \end{cases} \quad (7)$$

$$y_f = \begin{cases} 1 & \text{if color (frequency) } f \text{ used} \\ 0 & \text{else.} \end{cases} \quad (8)$$

List all of the vertices that are adjacent to v_i on graph G_0 in set $V^0(i)$. Likewise, list all of the vertices that are adjacent to v_i on graph G_1 in set $V^1(i)$. Hence, $(v_i, v_j) \in E(V(G_l)) \rightarrow v_j \in V^l(i), \forall i, j \in I, l = 0, 1$. The minimum order frequency assignment problem with only co-channel and first adjacent channel constraints is equivalent to the following integer program.

IP- 1.

$$\text{minimize } z = \sum_{f \in F} y_f \quad (9)$$

subject to

$$\sum_{f \in F} x_{if} = 1 \quad \forall i \in I \quad (10)$$

$$\sum_{i \in I} x_{if} \leq ny_f \quad \forall f \in F \quad (11)$$

$$\sum_{j \in V^0(i)} x_{jf} \leq |V^0(i)|(1 - x_{if}) \quad \forall i \in I, \forall f \in F \quad (12)$$

$$\sum_{j \in V^1(i)} (x_{j1} + x_{j2}) \leq 2|V^1(i)|(1 - x_{i1}) \quad \forall i \in I \quad (13)$$

$$\sum_{j \in V^1(i)} \sum_{l=f-1}^{f+1} x_{jl} \leq 3|V^1(i)|(1 - x_{if}) \quad \forall i \in I, \forall f \in F \setminus \{1, m\} \quad (14)$$

$$\sum_{j \in V^1(i)} (x_{j,m-1} + x_{jm}) \leq 2|V^1(i)|(1 - x_{im}) \quad \forall i \in I \quad (15)$$

$$x_{if} \in \{0, 1\} \quad \forall i \in I, \forall f \in F \quad (16)$$

$$y_f \in \{0, 1\} \quad \forall f \in F. \quad (17)$$

We make a few comments about IP-1. Equation (10) requires that every vertex be assigned exactly one color. Equation (11) says that if color f is available ($y_f = 1$) then up to n vertices could be assigned color f (subject to the remaining constraints). Equation (12) defines the co-channel constraints by stating that if vertex v_i is assigned color f , then no vertices adjacent to v_i can be assigned color f . Equation (14) defines the adjacent-channel constraints by stating that if vertex v_i is assigned color f , then no vertices adjacent to v_i can be assigned color $f-1$, f or $f+1$. Note that if $V^1(i^*) = \{\emptyset\}$ then the summation in equation (14) skips i^* . Equations (13) and (15) are just special cases of equation (14) for $f = 1$ and m respectively.

Baybar included another constraint in IP-1 that attempts to keep the minimum order assignment from being very large in span. He accomplishes this by forcing the assigned spectrum to be consecutive:

$$y_f \geq y_{f+1}, \quad \forall f \in F \setminus \{m\} \quad (18)$$

A major problem with the IP-1 formulation is that a very good a priori estimate of m is necessary. If m is chosen too small, the problem will be infeasible, while if chosen too large, the computational complexity increases unnecessarily and the span most likely will as well. Use of fast heuristics or Theorems 4.26 and 4.27 to obtain bounds on m may help. If the problem is a list-coloring where the spectrum F is known and inflexible, then estimating m is no longer as critical an issue.

We can modify IP-1 to the general case of $(K + 1)$ -level nested T -colorings. Consider the nested multigraph $G(V, G_0, G_1, \dots, G_k, \dots, G_K)$ and T -sets $T(k) = \{t_1(k), t_2(k), \dots, t_{q_k}(k), \dots, t_{Q_k}(k)\}$, $T(0) \subset T(1) \subset \dots \subset T(k) \subset \dots \subset T(K)$. $V^k(i)$ is the set of all vertices adjacent to vertex v_i on graph G_k , $k = 0, 1, \dots, K$. Then we may replace equations (12) - (15) with the following:

$$\sum_{j \in V^k(i)} \sum_{q_k=1}^{Q_k} (x_{j,(f-t_{q_k}(k))} + x_{jf} + x_{j,(f+t_{q_k}(k))}) \leq 3|V^k(i)|(1 - x_{if}), \quad (19)$$

$$\forall i \in I, \forall f \in F, k = 0, 1, \dots, K$$

such that

$$x_{j,(f-t_{q_k}(k))} = 0 \quad \text{if } t_{q_k}(k) \geq f, \quad (20)$$

$$\forall j \in I, \forall f \in F, k = 0, 1, \dots, K.$$

As part of the *European Cooperation on the Long term in Defense* (EU-CLID) program, six research groups pooled their efforts on the *Combinatorial Algorithms for Military Applications* (CALMA) project. One of the major objectives was to develop and test algorithms for frequency assignment problems. For an overview of the CALMA work, see the report by Tiourine, Hurkens and Lenstra [87].

The CALMA group worked from a common problem definition that did not use a graph coloring model as its foundation. Instead, an IP formulation was developed directly from the frequency “distance” constraint concepts described in Section 2.2. They also added a new concept, that of *mobility*, which is the ease with which a given transmitter can change frequencies given that it already has a frequency assigned to it. Mobility answers the concern raised by Box [13] and Smith [81] that for expensive terrestrial networks, changing the operating frequency of some pre-existing transmitter stations could be quite expensive.

The CALMA researchers addressed three objectives: minimum order, minimum span and minimum interference for otherwise infeasible instances.

The three integer programs that follow are from the paper by Wisse [93]. The foundation for all of the CALMA integer programs is an interference graph $G(V, E)$ with vertices $V = \{v_1, \dots, v_n\}$ representing transmitters, and an edge $(v_i, v_j) \in E$ only if a distance constraint exists between transmitters v_i and v_j . It is important to realize that this interference graph is very different from the graphs defined by Hale [46] and Cozzens and Roberts [21] in Sections 4.1 and 4.2. The graphs described by Hale and Cozzens and Roberts have an edge defined only for a fixed frequency separation or “distance”, whereas the CALMA interference graph has an edge for *any* finite frequency distance.

As before, define the index set of vertex set V to be I and the set of all available frequencies to be $F = \{1, 2, \dots, m\}$. We will use the decision variables that we defined in equations (7) and (8). Let $\text{dom}_i \subseteq F$ be the set of all frequencies available to transmitter v_i and init_i the frequency initially assigned to v_i (if v_i is a new transmitter, $\text{init}_i = \{\emptyset\}$). The network designer can choose one of 5 values for the mobility of a transmitter; $\text{mob}_i \in \{0, 1, 2, 3, 4\}$ where $\text{mob}_i = 0$ means that the frequency of transmitter v_i cannot be changed and $\text{mob}_i = 4$ implies that the frequency of transmitter v_i is easiest to change. As we stated before, $(v_i, v_j) \in E$ implies that a distance constraint exists between transmitters v_i and v_j . Hence, E is the set of all distance constraints. There are two types of distance constraints represented by subsets E^{eq} and E^{in} :

$$E^{eq} = \{(v_i, v_j) : |f - g| = d_{ij}, \text{ where } x_{if} = 1, x_{jg} = 1, f, g \in F\} \quad (21)$$

$$E^{in} = \{(v_i, v_j) : |f - g| > d_{ij}, \text{ where } x_{if} = 1, x_{jg} = 1, f, g \in F\} \quad (22)$$

such that $E^{eq} \cup E^{in} = E$ and $E^{eq} \cap E^{in} = \{\emptyset\}$. Clearly E^{eq} is a set of equality constraints, which prove useful for representing duplex links in the network (the reverse link in a duplex net can always be assigned a frequency that is exactly some fixed distance away from the forward link). E^{in} is the set of inequality constraints and as such has more general applicability; being able to model interference between any transmitter pair. The equality set E^{eq} was included to reduce the number of inequality constraints when duplex links are known to exist, which is common in CALMA problem instances.

There are several ways to define an integer program on this model. One of the CALMA minimum order IP formulation follows. The other possibilities are detailed in Wisse’s report [93].

IP- 2.

$$\text{minimize } z = \sum_{f \in F} y_f \quad (23)$$

subject to

$$\sum_{f \in F} x_{if} = 1 \quad \forall i \in I \quad (24)$$

$$\sum_{i \in I} x_{if} \leq ny_f \quad \forall f \in F \quad (25)$$

$$x_{jf} = x_{i,(f-d_{ij})} + x_{i,(f+d_{ij})} \quad \forall f \in F, \forall (v_i, v_j) \in E^{eq} \quad (26)$$

$$\sum_{g: |f-g| \leq d_{ij}} x_{jg} \leq 1 - x_{if} \quad \forall f \in F, \forall (v_i, v_j) \in E^{in} \quad (27)$$

$$\sum_{i \in I} x_{if} = 0 \quad \forall f \in F \setminus \text{dom}_i \quad (28)$$

$$x_{if} = 1 \quad \forall i : \text{init}_i = f \text{ and } \text{mob}_i = 0 \quad (29)$$

$$x_{if} \in \{0, 1\} \quad \forall i \in I, \forall f \in F \quad (30)$$

$$y_f \in \{0, 1\} \quad \forall f \in F \quad (31)$$

Notice that in IP-2, equations (23), (24), (25), (30) and (31) are equivalent to equations (9), (10), (11), (16), and (17) of IP-1 respectively. Equation (26) enforces the equality constraints by requiring that the paired transmitter v_i receive a frequency exactly d_{ij} above or below that of the subject transmitter v_j . As a result, we realize that for equation (26) to be feasible, the maximum degree of any vertex in the graph $G(V, E^{eq})$ is two. Equation (27) represents the inequality constraints. If $x_{if} = 1$ then x_{jg} must be zero for all frequencies g that interfere with f . If $x_{if} = 0$, then at most one transmitter can use a frequency g that interferes with f . Equation (28) simply denies the use of a frequency not in the transmitter's domain and equation (29) does not allow the frequency of a transmitter with zero mobility to be changed.

For a minimum span problem, we change the objective to minimizing the difference between the largest and smallest frequencies assigned. Without loss of generality, we can assume that in any solution, the smallest frequency will be 1. If it is not, we simply subtract the smallest frequency value plus 1 from each frequency in the solution and obtain this result. Thus, the minimum span objective reduces to minimizing the maximum frequency.

Define a new decision variable μ to be the maximum frequency assigned. The CALMA minimum span IP formulation is given next.

IP- 3.

$$\text{minimize } \mu \quad (32)$$

subject to

$$\sum_{f \in F} x_{if} = 1 \quad \forall i \in I \quad (33)$$

$$x_{jf} = x_{i,(f-d_{ij})} + x_{i,(f+d_{ij})} \quad \forall f \in F, \forall (v_i, v_j) \in E^{eq} \quad (34)$$

$$\sum_{g: |f-g| \leq d_{ij}} x_{jg} \leq 1 - x_{if} \quad \forall f \in F, \forall (v_i, v_j) \in E^{in} \quad (35)$$

$$\sum_{f \in F} f x_{if} \leq \mu \quad \forall i \in I \quad (36)$$

$$\sum_{i \in I} x_{if} = 0 \quad \forall f \in F \setminus dom_i \quad (37)$$

$$x_{if} = 1 \quad \forall i : init_i = f \text{ and } mob_i = 0 \quad (38)$$

$$x_{if} \in \{0, 1\} \quad \forall i \in I, \forall f \in F \quad (39)$$

$$\mu \in F \quad (40)$$

The only difference between IP-2 and IP-3 is that the objective functions are changed. Equation (25) was not included in IP-3 and equations (36) and (40) were added to ensure that all frequencies in a solution must be at or below the maximum frequency μ .

If no feasible minimum order or minimum span assignment can be found,² then the CALMA objective is to minimize either the interference, the number of frequencies reassigned, or both. We will focus on a formulation that minimizes both.

Let b_{mob_i} be the penalty paid when the preassigned frequency is changed, i.e. when $x_{if} = 1$ but $f \neq init_i$ (recall that $mob_i \in \{0, 1, 2, 3, 4\}$). CALMA defines the mobility for any equality constrained transmitter pair to be zero. Thus only inequality constrained transmitters can contribute to the mobility penalty.

²We should point out that by sufficiently increasing m , a feasible solution can always be found. However, if F is a restricted set, this may not be the case.

Define $a_{p_{ij}}$ to be the penalty paid if constraint $(v_i, v_j) \in E^{in}$ with priority $p_{ij} \in \{1, 2, 3, 4\}$ is violated. Partition E^{in} into four subsets:

$$\Pi_p = \{(v_i, v_j) \in E^{in} : p_{ij} = p, p = 1, 2, 3, 4\}$$

Define a decision variable w_{ij} on constraint $(v_i, v_j) \in E^{in}$ to identify when a constraint has been violated:

$$w_{ij} = \begin{cases} 1 & \text{if } |f - g| \leq d_{ij} \text{ and } (v_i, v_j) \in E^{in}, \text{ where } x_{if} = 1, x_{jg} = 1 \\ 0 & \text{otherwise.} \end{cases} \quad (41)$$

The objective is to minimize the total penalty caused by violating constraints and changing frequencies. The IP formulation for infeasible instances follows.

IP- 4.

$$\text{minimize} \sum_{q=1}^4 \sum_{v_i: mob_i=q} b_q (1 - x_{i,init_i}) + \sum_{p=1}^4 \sum_{(v_i, v_j) \in \Pi_p} a_p w_{ij} \quad (42)$$

subject to

$$\sum_{f \in F} x_{if} = 1 \quad \forall i \in I \quad (43)$$

$$x_{jf} = x_{i,(f-d_{ij})} + x_{i,(f+d_{ij})} \quad \forall f \in F, \forall (v_i, v_j) \in E^{eq} \quad (44)$$

$$\sum_{g: |f-g| \leq d_{ij}} x_{jg} \leq 1 - x_{if} + w_{ij} \quad \forall f \in F, \forall (v_i, v_j) \in E^{in} \quad (45)$$

$$\sum_{i \in I} x_{if} = 0 \quad \forall f \in F \setminus dom_i \quad (46)$$

$$x_{if} = 1 \quad \forall i : init_i = f \text{ and } mob_i = 0 \quad (47)$$

$$x_{if} \in \{0, 1\} \quad \forall i \in I, \forall f \in F \quad (48)$$

$$w_{ij} \in \{0, 1\} \quad \forall (v_i, v_j) \in E^{in} \quad (49)$$

Notice that in the first term of the objective function, that when $x_{i,init_i} = 1$, transmitter v_i gets its preassigned frequency and consequently there is no penalty incurred. In the second term, when the IP chooses to violate constraint (v_i, v_j) , then $w_{ij} = 1$ and a penalty of $a_{p_{ij}}$ is incurred. Other

Table 1: Size of Integer Program Formulations

	number of variables	number of constraints
IP-1	$m + nm$	$n + m + 2nm$
IP-1 with eq (19)	$m + nm$	$n + m + Knm$
IP-2	$m + nm$	$n + m + m E $
IP-3	$1 + nm$	$2n + m E $
IP-4	$ E^{in} + nm$	$n + m E $

than the new objective function, IP-4 differs from IP-3 in three respects. Equation (49) has been added, equations (36) and (40) were removed, and equation (35) was modified to be (45). The idea of equation (45) is to permit up to two links to share a frequency, even when this causes interference.

Table 1 compares the number of variables and constraints for the different IP formulations. We see that the CALMA formulation of IP-2 is smaller than the modified version of Baybar's formulation (IP-1 with equation (19)) if $|E| < Kn$, where $|E|$ is the number of edges in the CALMA interference graph and K is the number of interference levels in the multigraph. In general, both formulations are able to represent the same problems.

Many formulations different than those presented here could most certainly be developed, especially if one were to exploit the structure of a very specific problem. Additionally, the constraints in the IPs of this section could probably be strengthened. For instance, equation (13) can be strengthened by the valid inequalities:

$$\sum_{i \in C} x_{if} \leq y_f, \quad (50)$$

where C is a clique of vertices in either the multigraph for IP-1 or the interference graph for IP-2. If C is the maximum clique, then equation (50) is a facet of the convex hull of the feasible solution space and is maximally strong. Indeed, an interesting relationship between the formulation of IP-2 and the minimum cardinality vertex packing problem was pointed out by the Delft and Eindhoven CALMA team [1]. They point out that equation (27) is in fact a valid inequality that strengthens the general vertex packing clique constraints

$$x_{if} + x_{jg} \leq 1, \quad \forall i \in I, \quad \forall (v_i, v_j) \in E.$$

For more on constructing strong valid inequalities, see Nemhauser and Wolsey [66].

6 Exact Solution Techniques

In the simplest case, when $T = \{0\}$ (the co-channel FAP) T -coloring reduces to simple graph coloring. Finding $\chi(G)$ is, in general, an \mathcal{NP} -complete problem [40]. This fact implies that both T -coloring and T -coloring multigraphs are very hard problems. Consequently, we expect that when a special graph or T -set structure does not exist, any exact method will only be useful for solving small problems. However, even though exact methods suffer from long run-times and an inability to handle large problems, they are useful for testing the accuracy of faster heuristics on small instances.

Here we mention two graph coloring approaches: a *backtracking method* that is basically a branch and bound technique that uses a tree search, and a *set covering* approach. Both techniques were developed for graph coloring problems.

It is widely known that integer programs can be solved by a branch and bound strategy. However, in general, one needs good bounds and an efficient branching strategy for this to be effective. Furthermore, Nemhauser and Wolsey show that the complexity of finding all facets of $\text{conv}(S)$ is \mathcal{NP} -complete where $S = P \cap Z_+^n$ and $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$. Hence, finding an optimal solution to an IP is in general also a hard problem. Branch and bound, cutting plane, and other techniques are discussed in-depth in Nemhauser and Wolsey [66].

6.1 Backtracking Methods

Brelaz [14] and Peemöller [69] introduced a tree search branch and bound method for simple (one-level) graph coloring that uses backtracking and an upper bound on the chromatic number obtained by a sequential heuristic (see Section 7.1 for information on sequential heuristics). The technique attempts to find a large clique (the larger the better) of size r in the graph which can then, of course, be colored $1, 2, \dots, r$. At this point usually r is less than the upper bound (if not it is optimal). So, the algorithm tries to color the vertices outside the clique with colors less than the computed upper bound. The algorithm considers these vertices sequentially by some vertex ordering (usually saturation degree [14]) and attempts to color the first vertex in the ordering (which we will call the root vertex) the smallest unused color. If this assignment causes a violation to occur, then the

vertices adjacent to the root vertex which are causing the violation are recolored through backtracking; essentially enumerating a search tree that contains the vertices adjacent to the root vertex. All of the violating adjacent vertices must be recolored, which in turn generates another backtrack step. This process continues until a feasible recoloring is obtained. If a feasible recoloring is not obtained, then the next largest color is assigned to the root. This process is repeated for all uncolored vertices, and subsequent backtracked vertices, until the graph has been completely colored.

Lanfear [60] successfully applied this technique, with modifications to the backtracking criteria, to several two-level frequency assignment problems. Smith [81] also applied a backtracking idea to two-level frequency assignment problems. As we would expect, all of the tree search methods only work well on small problems since backtracking is an exponential time algorithm. The performance of backtracking techniques is investigated in [4, 92].

6.2 Set Covering Approach

Cameron [15] developed an algorithm that solves a minimum set covering problem to test the hypothesis that a graph has a given chromatic number. The test is presented as:

Given a graph $G(V, E)$ with $n = |V|$ and $p \geq \chi(G)$ the hypothesized chromatic number, the graph is p -colorable if and only if the minimum set cover of G has exactly np columns.

When the test is repeated for all chromatic numbers between an upper bound (as found by a heuristic) and a lower bound (the clique number) the resulting algorithm is exact. Unfortunately the storage requirement are intensive. Matrices generated for testing the set covers are large; $p(m+n)+n$ rows by $2np$ columns, which, for a graph with 50 vertices, 50 edges and $\chi(G) = 10$ yields a matrix of 1050 by 500. Furthermore, the algorithm only solves the co-channel problem and there are no known results on the minimum set cover which generalize to multigraphs.

7 Approximate Solution Methods

Since for general problems exact methods cannot solve large problems, we now turn our attention to some approximate techniques.

7.1 Sequential Heuristics For Simple Graph Coloring

Metzger [64] is generally credited with the introduction of sequential assignment methods. He described three fundamental ways with which a graph can be colored:

Frequency Exhaustive: Given an ordering of the vertices, attempt to color each vertex, sequentially, the smallest feasible color. This approach is also called a *greedy* coloring.

Requirement Exhaustive: Given an ordering of the vertices, attempt to color each vertex, sequentially, color 1. Then repeat for color 2 and so on.

Uniform: Given an ordering of the vertices, attempt to color each vertex, sequentially, the color that has been least used.

A critical issue in each of these sequential techniques is the *vertex order* (sequence) which is developed based upon some as yet unspecified rule.

The first sequential heuristic that showed promise for obtaining good solutions was introduced by Matula and Beck [63] and called *smallest last*. This algorithm was applied to simple (co-channel) graph colorings and obtained good solutions. The smallest-last ordering is found according to the pseudo-code in Figure 10. Essentially, *smallest-last* assigns the vertex of smallest degree in V to be v_1 . This vertex is then deleted from the graph and then next smallest degree vertex is found, assigned to be v_2 and deleted. The algorithm proceeds until all vertices have been deleted.

The smallest-last sequential coloring of a graph is constructed by applying the frequency exhaustive greedy algorithm, proceeding sequentially according to the smallest-last vertex order $(v_n, v_{n-1}, \dots, v_1)$. Matula and Beck [63] showed that the worst case complexity of finding a smallest-last order was $O(n + e)$, where $n = |V|$ and $e = |E|$.

Zoellner and Beall [95] discussed a similar sequential heuristic termed *largest-first* which they compare to smallest-last. Largest-first orders the vertices of the graph according to their degree in G : largest to smallest. The frequency exhaustive assignment is then applied to this ordering to obtain a coloring. It is easy to see that a largest-first vertex order can be found in linear time.

Zoellner and Beall [95] compared smallest-last, largest-first, and random vertex orders by applying each to frequency exhaustive and uniform sequential methods. Their results indicate that frequency exhaustive assignments and smallest-last sequences perform best on co-channel graphs. A frequency

```

procedure Smallest-Last( $G(V, E)$ ,  $n = |V|$ )
1    $v_1 \leftarrow \min_{i=1,2,\dots,n} \{ \deg(v_i \in V) \};$ 
2    $V^1 = V - v_1;$ 
3   for  $j = 2, \dots, n$ 
4        $v_j \leftarrow \min_{i=1,\dots,n-j+1} \{ \deg(v_i \in V^{j-1}) \};$ 
5   rof;
6   return  $(v_n, v_{n-1}, \dots, v_1);$ 
end Smallest-Last;

```

Figure 10: Pseudo-Code For Finding Smallest-Last Ordering Algorithm

exhaustive assignment using a smallest-last vertex order obtained solutions that were within 6 to 8 percent of $spt(G)$. They also tested the approaches on 2-level multigraphs with less success, being 10 to 15 percent over $spt(G)$.

In Section 4.4, list colorings were presented as a model for frequency assignment problems with a restriction on the frequencies assignable to each vertex. Suppose a sequential coloring algorithm has been modified to list color a graph. For a small enough list or dense enough graph, we will often find that at some vertex $v_{\pi(i)}$ in the ordering π , there will exist no colors in the list that yield a feasible coloring. Leung addresses this problem in [61] and proposes a *partial backtracking* technique to rectify it.

Backtracking, as described in Section 6.1, is a well known process in *depth-first* integer programming branch and bound methods where if an infeasible solution is found, the algorithm backtracks to the node that has a branch not yet fathomed. Leung's partial backtracking technique only steps back one level and attempts to resolve the conflict that makes the solution infeasible. However, there is no guarantee that it will resolve the conflicts. Therefore, while it improves the performance of a sequential technique on list colorings, it does not ensure a solution.

Leung's partial backtracking is similar to the exact approaches in Section 6.1 but has a different approach for resolving conflicts. Given a vertex $v_{\pi(i)}$ which cannot be assigned a color from its list $S(\pi(i))$, Leung's technique operates by finding a set of vertices which have already been assigned a color but which if assigned an alternate (but feasible) color would permit $v_{\pi(i)}$ to be colored from $S(\pi(i))$. If this set is of minimum cardinality, it is called the *reassignable denying set*. The algorithm then backtracks by reassigning the vertices in the reassignable denying set and assigning $v_{\pi(i)}$ a now

feasible color. If a reassignable denying set is not found, then the algorithm terminates without a solution. Partial backtracking could be modified to be exact by allowing multiple steps backward as described in Section 6.1.

Box [13] also developed an iterative sequential heuristic for list colorings. The basic idea is to color the most difficult vertices first. Box begins by assigning each vertex a difficulty score of zero. The vertices are then ordered by any method (including random) and are sequentially colored from a finite list of colors. Those vertices that cannot be colored are denied a frequency and are assigned a difficulty score uniform randomly from some small predefined range (say 0.15 to 0.45). The algorithm then repeats with a reordering of the vertices by their difficulty scores in decreasing magnitude and a recoloring. The algorithm stops when a feasible coloring is achieved. The idea is that vertices that are difficult to color will eventually “rise to the top of the ordering” and thus be colored first. However, for large problems or infeasible instances the algorithm does not necessarily converge to an optimal solution or improve the assignment from that of a smallest last ordering. Nonetheless, the randomization idea (used by Box to prevent cycling in the vertex order) is intriguing and could be further developed in a sequential heuristic to prevent local entrapment while ensuring global convergence.

In fact, Gamst and Rave [39] used Box’s heuristic in a *decomposition method* which they developed for list-set colorings. The basic premise of the heuristic is that Box’s technique should color small graphs much faster than larger ones. Hence Gamst and Rave partition the interference graph into 3 parts: the *head* which is composed of those vertices that form the maximal clique of G , the *tail* which is those vertices which have degree strictly less than the clique number $w(G)$, and the *body* which is the remaining vertices. The algorithm first colors the head by applying a frequency exhaustive assignment. The body is colored next by applying Box’s iterative method, followed by the tail which is also colored iteratively. The method works fairly well for co-channel problems but is not reliable on adjacent channel problems and tends towards long run-times if the graph is sparse (small $w(G)$).

Brelaz [14] developed the DSATUR algorithm that uses the vertex order specified by the *saturation degree* of the vertices. The saturation degree of a vertex is defined to be the number of different colors that exist on the vertices that are adjacent. The vertex with the highest saturation degree is “most denied” since it has fewer colors to choose from. Hence Brelaz specified an ordering of the vertices by saturation degree from highest to lowest. The graph is then colored by a frequency exhaustive technique according to the ordering.

Eisenblätter [31] applied DSATUR to cellular frequency assignment problems with only co-channels and adjacent channels and added a local search phase to improve the primal solution. The preliminary results were very promising. de Werra and Gay [27] also applied DSATUR to some problems with co-channels and adjacent channels.

Hale [47] expanded upon the work of Metzger and Zoellner and Beall by defining a generalized structure for all sequential coloring algorithms that consists of three fundamental steps:

Step 1: Order the vertices.

Step 2: select the next vertex to color.

Step 3: select the color.

Hale's procedure is general. It covers all types of vertex orderings in step 1 and permits Metzger's three sequential techniques in step 3. It adds step 2 to permit the coloring sequence to adapt during the process. Hale introduced new sequential techniques for step 2 that are adaptive variants of the saturation degree.

7.2 The Greedy T -coloring Algorithm

The greedy T -coloring algorithm, introduced by Cozzens and Roberts [21] and shown in Figure 11, is a frequency exhaustive sequential heuristic that the T -coloring research community has studied in great detail (actually the algorithm presented in [21] is somewhat different than the one shown in Figure 11, but in subsequent papers the Figure 11 definition has prevailed). In essence, the algorithm recursively colors the vertices of the graph with the smallest feasible color proceeding according to some prior defined vertex ordering. While the algorithm always results in a feasible coloring (for an unrestricted set or list of available colors) for any graph, it rarely finds the coloring with span $spt(G)$. However, for a few specific graphs, T -sets and vertex orderings presented in this section, $spt(G)$ can be obtained with the greedy algorithm. This is of great interest to us since the complexity of the greedy T -coloring algorithm is $O(n^2t)$ where $n = |V|$ and $t = |T|$.

Two types of results are presented here: those obtained for coloring complete graphs with the greedy T -coloring algorithm and those for graphs implied by special vertex orderings identified in Section 3.5.

```

procedure Greedy T-coloring(graph  $G(V, E)$ , T-set  $T \subseteq \mathbb{Z}_+$ ,
vertex order  $\{v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(p)}, \dots, v_{\pi(n)}\}$ ;
1   colors =  $\mathbb{Z}_+ \setminus \{0\}$  ;
2   color  $f(v_{\pi(1)}) = 1$  ;
3   do  $p = 2, 3, \dots, n$ ;
4       color  $f(v_{\pi(p)}) = \min\{\text{colors}\}$ 
4           s.t.  $|f(v_{\pi(p)}) - f(v_{\pi(k)})| \notin T \quad \forall k < p$ 
4           and  $(v_{\pi(k)}, v_{\pi(p)}) \in E(G)$ ;
5   od;
6   return coloring  $f$ 
end Greedy T-coloring;

```

Figure 11: Pseudo-Code For The Greedy T -Coloring Algorithm

7.2.1 Greedy T -colorings of Complete Graphs

Theorems 4.7, 4.10 and 4.11 raise the issue that an efficient algorithm that will optimally T -color complete graphs is needed. As a side note, these theorems state their results in terms of the chromatic number of the graph $\chi(G)$, the finding of which is known to be an \mathcal{NP} -hard problem. Hence, all of the results in this section depend very much on the ability to find $\chi(G)$ for an arbitrary graph. Since general graph coloring is so well studied, we will defer discussion on techniques for finding $\chi(G)$ and refer the reader to [52]. Roberts [76] further underscores the importance of this issue with the following theorem.

Theorem 7.1 (Roberts [76]). *For all weakly γ -perfect graphs G , there exists an ordering of the vertices for which the greedy T -coloring algorithm yields a T -coloring of span $spt(G)$ if and only if the greedy T -coloring algorithm applied to the complete graph $K_{\chi(G)}$ yields a coloring of span $spt(K_{\chi(G)})$.*

Therefore, by enumerating all of the T -sets that the greedy T -coloring algorithm will color K_n with optimal T -span (and since $\chi(G)$ can be found in polynomial time for a weakly γ -perfect graph) we have a practical method for finding which γ -perfect graphs are optimally colorable by the greedy T -coloring algorithm.

Observe that since the vertex order of a complete graph is arbitrary, any T -coloring of any complete graph K_n can be completely described by the T -set T and the integer n . Hence, results for this section will specify which

sets T and values of n that the greedy T -coloring gets $spt(K_n)$.

Theorem 7.2 (Raychaudhuri [71]). *For any complete graph K_n with k -multiple of s T -set T , an optimal T -span coloring can always be obtained using the greedy T -coloring algorithm.*

By the first result of Theorem 4.11 which says that for k -multiple of sT -sets $spt(G) = spt(K_{\chi(G)})$, the optimal span for any arbitrary graph with known chromatic number is also obtained. However, is it crucial to note that this observation does *not* imply that the optimal coloring of an arbitrary graph can be obtained since the greedy algorithm is applied to the complete graph $K_{\chi(G)}$.

Theorem 7.3 (Tesman [84]). *For any complete graph K_n with T -set $T = \{0, 1, \dots, s-1\} \cup \{r, r+1, \dots, r+t-1\}$ where $t < s$, the optimal T -span coloring can always be obtained by the greedy T -coloring algorithm.*

Tesman also pointed out that the first two lines of Theorem 4.19, Theorem 4.20 and Theorem 4.21 can all be obtained with the greedy T -coloring algorithm. Other results can be found in Cozzens and Wang [24] and Wang [89].

In *measurement theory* a measurement is said to be *meaningful* if the conclusion drawn from the measurement remains unchanged regardless of the scale or units assigned to the measurement. This definition implies that if a conclusion about some data is dependent upon the scale used to measure the data, the conclusion is not reliable and therefore meaningless. Cozzens and Roberts [23] used measurement theory to conclude that given some T -set T , it is meaningless to say that the greedy T -coloring algorithm obtains the T -span of K_n only for a specific value of n . Conversely, it is meaningful to say that the greedy T -coloring algorithm obtains the T -span of K_n for any (positive integer) value of n . The reasoning behind this conclusion is based on the ability to scale the units of the T -set without changing the results obtained by the greedy T -coloring algorithm.

Define a *ratio scaled T -set* as $sT = \{st | \forall t \in T\}$ where s is a positive integer chosen such that sT is again a set of positive integers. If the T -set is meaningful, then the greedy T -coloring algorithm will find the optimal span regardless of the ratio scale applied to T .

As an example, if $T = \{0, 1, 4, 5\}$, $n = 3$, $s = 2$ then the greedy T -coloring of K_3 uses colors $\{1, 3, 9\}$ which is not the optimal solution (colors $\{1, 4, 7\}$). Alternately, the greedy sT -coloring of K_3 uses colors $\{1, 2, 5\}$ which *is* the optimal solution. However, if we increase the value of n enough we will see

that the greedy sT -coloring is also no longer optimal. Hence, we must know how the greedy T -coloring algorithm responds for all n before we can draw any conclusions about the meaningfulness of T .

This fact is useful as a condition for proving when T is optimally colorable by the greedy algorithm.

Theorem 7.4 (Cozzens and Roberts [23]). *Given a positive integer s and T -set T , the greedy T -coloring algorithm applied to any complete graph K_n obtains a coloring with T -span $spt(K_n)$ if and only if the greedy sT -coloring algorithm applied to any complete graph obtains the sT -span $sp_{sT}(K_n)$ for all $n \in \mathbb{Z}_+ \setminus \{0\}$.*

We say that $T \in \mathcal{G}$ if Theorem 7.4 is true. That is, \mathcal{G} is the set of all T -sets that have the property that the greedy T -coloring algorithm applied to any complete graph K_n obtains a coloring with T -span $spt(K_n)$. We can think of Theorem 7.4 as a condition for $T \in \mathcal{G}$.

Liu [62] developed an alternate condition based upon her results in Theorem 4.13. Liu applied a greedy heuristic called the *recursive* method to construct a clique of size α in the T -graph G_T^α . Let the vertices of the T -graph be numbered as $V(G_T) = \{0, 1, 2, \dots, n\}$. The recursive method constructs a listing of vertices as follows: Let $\pi(1) = 0$ and $\pi(2)$ be the smallest integer in V that is adjacent to $\pi(1)$. Given that the construction at step i is $\{\pi(1), \pi(2), \dots, \pi(i)\}$, select $\pi(i+1)$ by finding the smallest integer in V that forms a clique with vertices $\{\pi(1), \pi(2), \dots, \pi(i)\}$. The procedure repeats until m vertices have been selected. The integers corresponding to $\{\pi(1), \pi(2), \dots, \pi(m)\}$ constitute a clique of size m in the T -graph G_T^α , where $\alpha = \pi(m)$.

Note that α may or may not be minimum for a clique of size m . That is, there may exist a clique of size m in G_T that has $\pi^*(m) < \alpha$. This clique is the maximum clique if $\pi^*(m)$ is minimal for choice of m . In a complementary sense, $\{\pi(1), \pi(2), \dots, \pi(m)\}$ are the colors that the greedy T -coloring algorithm colors the complete graph K_m (or you may say $1 + \{\pi(1), \pi(2), \dots, \pi(m)\}$ are the colors if the first color is 1 instead of 0). By Theorem 4.13 if $\pi(m)$ is minimal for the choice of m , then these colors achieve the T -span. The following theorem results.

Theorem 7.5 (Liu [62]). *$T \in \mathcal{G}$ if and only if the recursive procedure constructs a maximum clique in the T -graph G_T^n for all $n \in \mathbb{Z}_+ \setminus \{0\}$.*

To demonstrate the utility of the theorem, let us use the previous example: $T = \{0, 1, 4, 5\}$. For $m = 3$, the recursive procedure constructs the

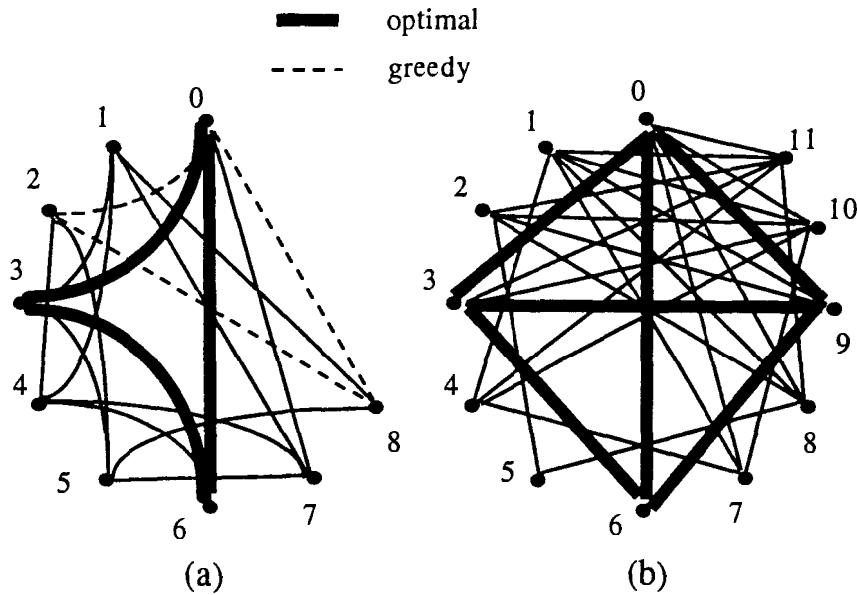


Figure 12: (a) Cliques on T -graph G_T^8 with $T = \{0, 1, 4, 5\}$; (b) Clique on T -graph $G_{T_2}^6$ with $T_2 = \{0, 1, 2, 4, 5\}$

clique $\{0, 2, 8\}$ so $\alpha = 8$. However, $\{0, 3, 6\}$ is also a clique in G_T^8 so $\{0, 2, 8\}$ is not maximal for $n = 3$.

Contrary to this past example, consider T -set $T_2 = \{0, 1, 2, 4, 5\}$. For $m = 3$, the recursive procedure constructs the clique $\{0, 3, 6\}$ which is maximal (there are no other cliques of size 3 in $G_{T_2}^6$). In fact, for all values of n , the recursive procedure constructs maximum cliques. Therefore, by Theorem 7.5, $T_2 \in \mathcal{G}$. Both examples are illustrated in Figure 12.

7.2.2 Greedy T -colorings and Vertex Orderings

Recall from Section 3.5 that the vertices of a graph can be ordered in a certain way, if and only if the graph is of a certain type. Such was found to be the case for compatible orderings and indifference graphs, reverse of a p.e.o. and chordal graphs and admissible orderings and perfectly orderable graphs. Results on greedy T -coloring graphs implied by special vertex orderings are summarized below.

Theorem 7.6 (Cozzens and Roberts [21]). *For any indifference graph $G(V, E)$ and any T -set T , let $n = |V|$ and $t = |T|$.*

- (i) *The greedy T -coloring algorithm colors the graph with $\chi_T(G) = \chi(G)$ colors in $O(n^2t)$ time using a compatible vertex ordering.*
- (ii) *If in addition T is an r -initial set, then the greedy T -coloring algorithm solution has span $spt(G)$ and edge span $espt(G)$ and is found in $O(n^2)$ time.*

However, as defined, indifference graphs only model transmitters that are arranged on a line and therefore these results are of limited use. Raychaudhuri found more general results for the greedy T -coloring algorithm.

Theorem 7.7 (Raychaudhuri [71]). *For any chordal graph $G(V, E)$ and any T -set T , let $n = |V|$ and $t = |T|$.*

- (i) *The greedy T -coloring algorithm colors the graph with $\chi_T(G)$ colors in $O(n^2t)$ time using the reverse of a p.e.o. vertex ordering.*
- (ii) *If in addition, T is either an r -initial or k -multiple of s set, then the greedy T -coloring algorithm solution has span $spt(G)$ and can be found in $O(n^2)$ time.*

Recall from Section 3.5 that for any chordal graph, a reverse of a p.e.o. vertex ordering always exists. So assuming that the ordering can be found, an efficient algorithm exists for T -coloring chordal graphs. Many graphs found in nature are chordal so this result is quite useful. Furthermore, if necessary, we might add edges to a non-chordal graph to make it chordal and thus be able to solve a “graphical relaxation” of a general graph. It would be interesting to know how, if at all, such relaxations bound the optimal solution.

Raychaudhuri generalized the result on chordal graphs a step further in the following theorem.

Theorem 7.8 (Raychaudhuri [72]). *For any perfectly orderable graph $G(V, E)$ and any T -set T , let $n = |V|$ and $t = |T|$.*

- (i) *The greedy T -coloring algorithm colors the graph in $\chi_T(G)$ colors in $O(n^2t)$ time using an admissible vertex ordering.*
- (ii) *If in addition T is either an r -initial or k -multiple of s set, then the greedy T -coloring algorithm solution has span $spt(G)$ and can be found in $O(n^2)$ time.*

Recall from Section 3.5 that for any perfectly orderable graph, an admissible vertex ordering always exists. So, assuming that the ordering can be found, an efficient algorithm exists for T -coloring perfectly orderable graphs. However, finding the admissible ordering is known to be a hard problem [76] limiting (for now) the usefulness of this result.

Notice an important conclusion regarding the previous theorems: by the definitions in Section 3.5, Theorem 7.8 generalizes Theorem 7.7 which in turn generalizes Theorem 7.6.

Cozzens and Wang [24] show that in general, the results of Theorems 7.6, 7.7 and 7.8 do not apply to multigraphs. They define a multigraph

$$G(V, G_0, G_1, \dots, G_K)$$

to be a $(K + 1)$ -homogeneous indifference system if there exists a vertex ordering of V that is compatible for all graphs G_k , $k = 0, 1, \dots, K$. They then state that even if $G(V, G_0, G_1, \dots, G_K)$ is a $(K + 1)$ -homogeneous indifference system span assignments are not guaranteed by the greedy T -coloring algorithm. However, they did prove the following rather restrictive result.

Theorem 7.9 (Cozzens and Wang [24]). *Let $|V| = n$. If each $T(k)$ is r_k -initial, $k = 0, 1, \dots, K$, $\chi(G_j) = \chi(G_0)$, $j = 1, 2, \dots, K$ and G_0 is chordal, then the greedy $T(K)$ -coloring of G_0 obtains $\chi_T(G)$, $spt_T(G)$ and $esp_T(G)$ in $O(n^2)$ time.*

The proof of this result follows from Theorem 7.7 and Theorem 4.25 part (v).

7.3 The Greedy k-Tuple T -coloring Algorithm

Tesman [84] developed an extension to the greedy T -coloring algorithm for k -tuple set coloring problems discussed in Section 4.5. The *greedy k -tuple T -coloring* algorithm is a slight modification to Figure 11 and appears in Figure 13. Note that $R(\pi(i))$ is the set of k colors that are assigned to vertex $v_{\pi(i)}$. Tesman proves the following interesting results which relate the performance of greedy k -tuple T -coloring to that of greedy T -coloring already discussed.

Theorem 7.10 (Tesman [84]). *Given any graph $G(V, E)$ and T -set T , let π be a vertex ordering of $V(G)$ and let the greedy T -coloring of G which follows π have order equal to q . Then the order of the greedy k -tuple T -coloring of G which follows π will be exactly kq .*

```

procedure Greedy k-tuple T-coloring( $G(V, E)$ , T-set  $T \subseteq Z_+$ ,
vertex order  $\{v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(p)}, \dots, v_{\pi(n)}\}$ );
1   colors =  $Z_+ \setminus \{0\}$  ;
2   color  $R(\pi(1)) = \{1, 2, \dots, k\}$  ;
3   do  $p = 2, 3, \dots, n$ ;
4     color  $R(\pi(p)) = \min\{\text{k-tuple of colors}\}$ 
4     s.t.  $|r_l(\pi(p)) - r_m(\pi(k))| \notin T \quad \forall k < p$ 
4     and for all  $r_l \in R(\pi(i))$  and all  $r_m \in R(\pi(j))$ 
4     and  $(v_{\pi(k)}, v_{\pi(p)}) \in E(G)$ ;
5   od;
6   return set coloring  $R$ 
end Greedy k-tuple T-coloring;

```

Figure 13: Pseudo-Code For The Greedy k-tuple T -Coloring Algorithm

Theorem 7.11 (Tesman [84]). *Given a graph $G(V, E)$ that is weakly γ -perfect and T-set T , let π be a vertex ordering of $V(G)$ such that the greedy T -coloring which follows π has order equal to $\chi_T(G) = \chi(G)$.*

- (i) *For any T-set, the order of the greedy k-tuple T-coloring of G which follows π will be exactly $\chi_k(G)$.*
- (ii) *For $T = \{0, 1, \dots, m\}$, the span of the greedy k-tuple T-coloring of G which follows π will be exactly $sp_T^k(G)$.*

Theorem 7.12 (Tesman [84]). *Given a T-set $T = \{0, 1, \dots, m\}$, the greedy k-tuple T-coloring algorithm will always find $\chi_T^k(G)$ and $sp_T^k(G)$ for each of the following:*

- (i) *Graph $G(V, E)$ which is indifferent and a vertex ordering π that is compatible.*
- (ii) *Graph $G(V, E)$ which is chordal and a vertex ordering π that is the reverse of a perfect elimination ordering.*

7.4 Interset Graphs

Inspired by Raychaudhuri's work with Hamiltonian graphs and an algorithm by Carmassi and Tomati [16], Lanfear [60] describes a heuristic for 2-level

coloring that, during its execution, develops an alternative graph he calls the *Interset graph*. The advantage of Lanfear's work is that G_0 need not be complete. The disadvantage is that it is a heuristic and as such its performance cannot be guaranteed.

The Interset graph G^I is constructed as follows: color the co-channel graph G_0 using γ consecutive colors, where γ is a guess at the value of the T -span, $spt(G) + 1$. Observe that this process is equivalent to finding γ independent subgraphs on G_0 such that the vertices of G are partitioned into γ independent sets $\{y_1, y_2, \dots, y_\gamma\}$, $y_i \cap y_j = \{\emptyset\}$ and $V(G) = \cup_{i=1}^\gamma y_i$. Define γ vertices for the interset graph, one corresponding to each set: $V(G^I) \leftrightarrow \{y_1, y_2, \dots, y_\gamma\}$. If all vertices in set y_i are not adjacent to any vertex in set y_j on the adjacent channel graph G_1 , then we define (y_i, y_j) to be an edge on the interset graph.

If G^I has a Hamiltonian path, then G is γ -colorable. However, G could be γ -colorable and not have a Hamiltonian path, so it is important to find a γ -coloring of G_0 that introduces a lot of edges to G^I , thereby increasing the probability of finding a Hamiltonian path (assuming one exists). Furthermore, given a Hamiltonian path, $y_{\pi(1)}, y_{\pi(2)}, \dots, y_{\pi(\gamma)}$, we may color G with order γ by sequentially coloring the vertices in the independent sets: $f(v) = 1 \quad \forall v \in y_{\pi(1)}, \quad f(v) = 2 \quad \forall v \in y_{\pi(2)}, \dots \quad f(v) = \gamma \quad \forall v \in y_{\pi(\gamma)}$.

The idea of the heuristic is as follows: color the 2-level graph using a sequential coloring heuristic that checks feasibility on two levels: G_0 and G_1 . The span of this coloring will serve as a upper bound on the optimal span. Next obtain an estimate of the chromatic number of the co-channel graph. This value will serve as as a lower bound on the optimal span. At step 1, pick a value of γ_1 half way between the upper and lower bound. Construct an interset graph and search for a Hamiltonian path. If no path exists, increase the value of γ . If a path exists, decrease the value of γ and repeat the interset construction and Hamiltonian path search procedures until at step j no Hamiltonian path is found. The best span obtainable is established as γ_{j-1} and the corresponding coloring is a sequential coloring of the Hamiltonian path found at step $j - 1$.

Lanfear uses a simulated annealing process to find a γ -coloring of G_0 that maximizes the number of edges in the interset graph. The disadvantage of the algorithm is that we cannot guarantee that a Hamiltonian path will be found when one exists. However, Lanfear found that in practice it performed much better than a smallest-last order sequential heuristic. The pseudo-code for Lanfear's algorithm is in Figure 14.

A problem with Lanfear's algorithm is that it may achieve $spt(G)$ but not necessarily the minimum order for this value of span. Consider the

```

procedure Interset Heuristic  $(G(V, G_0, G_1), T(0) = \{0\}, T(1) = \{0, 1\}, G_0 \supset G_1)$ 
1   lowerbound:    $l = \chi(G_0);$ 
2   upperbound:    $(u, \text{bestcolor}) = \text{TwoLevelSequential}(G, T);$ 
3   while  $l < u$ 
4        $\gamma = \lfloor \frac{l+u}{2} \rfloor;$ 
5       do ConstructIntersetGraph  $G^I(Y, E)$ 
6            $Y = [y_1, \dots, y_\gamma] = \text{color}(G_0, \gamma);$ 
7            $\max |E(G^I)| \leftarrow \text{SimAnneal}(Y);$ 
8       od
9       if  $G^I$  has Hamiltonian path
10      do
11          bestcolor=  $Y;$ 
12           $u \leftarrow \gamma;$ 
13      od
14      else  $l \leftarrow \gamma + 1;$ 
15      fi
16  endwhile
17  return (bestcolor,  $\gamma - 1);$ 
end Interset Heuristic;

```

Figure 14: Pseudo-Code For Lanfear's 2-Level Heuristic

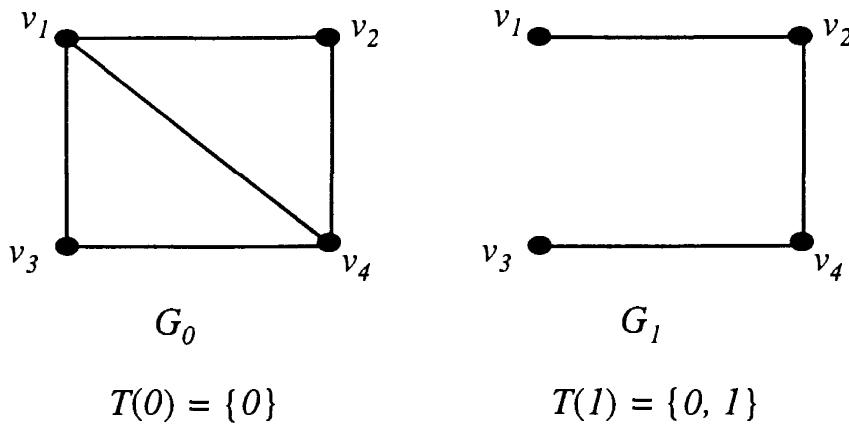


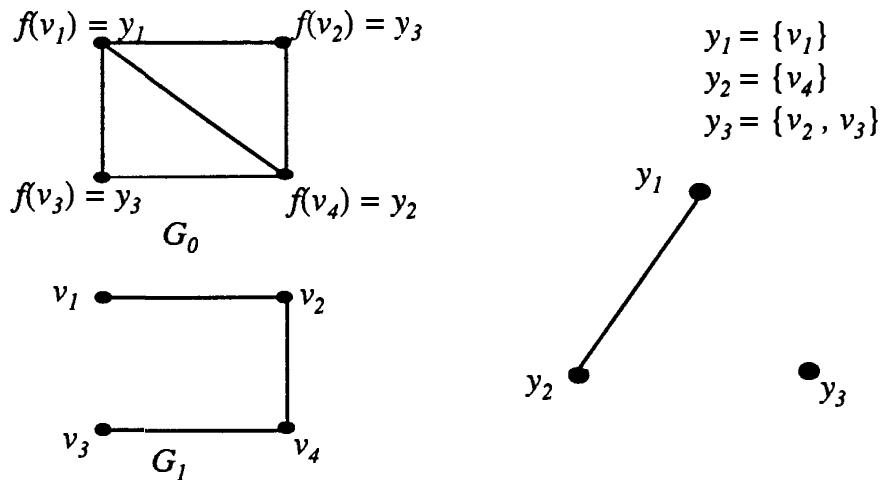
Figure 15: 2-Level Nested Multigraph

following example.

Example 7.1. Consider the 2-level nested graph in Figure 15. Figure 16 shows the interset graph for $\gamma = 3$ and Figure 17 the interset graph for $\gamma = 4$. Notice that for $\gamma = 3$ there is no Hamiltonian path whereas for $\gamma = 4$ there is. Hence we conclude that $spt(G) = 3$ and the colors of $\{v_1, v_2, v_3, v_4\}$ are $\{3, 1, 2, 4\}$. Clearly the order of this coloring is four. However, if we instead use the colors $\{1, 4, 4, 2\}$ then we obtain the T -order $\chi_T(G) = 3$ which is optimal. So we see that Lanfear's algorithm does not necessarily find the coloring that minimizes order for span equal to $spt(G)$. A coloring that minimizes the order for the span equal to $spt(G)$ is termed a *restricted span* coloring. Hence, as is, Lanfear's algorithm does not find restricted span colorings. Restricted colorings are discussed in [46, 76] as a more complete definition of optimality for multilevel colorings. Characterizing restricted colorings remain an open problem.

7.5 Simulated Annealing, Tabu Search, and Genetic Algorithms

This section presents a group of algorithms known as *global heuristics* due to their ability to escape entrapment at local optima. All of the methods presented provide the ability to find global solutions in an efficient manner. However, they all make extensive use of parameters that must be determined empirically and the quality of the solutions is usually sensitive to these parameters. Thus a “well-tuned” heuristic may work very well for a particular

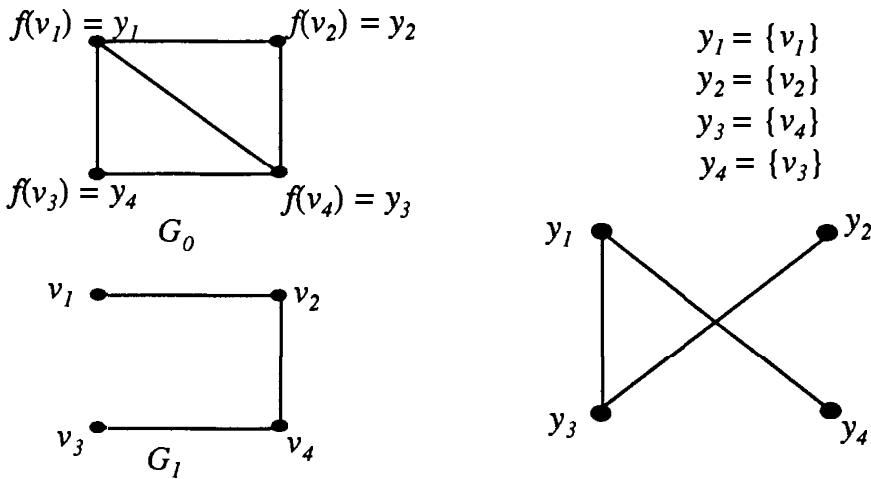
Figure 16: Interset Graph For $\gamma = 3$

type of FAP but the tuning procedure may require substantial effort.

7.5.1 Simulated Annealing

Simulated Annealing (SA) was formally introduced for general optimization by Kirkpatrick, Gelatt and Vecchi [57]. Since that time it has been used quite successfully on a number of combinatorial optimization problems [2]. Not surprisingly, the simulated annealing algorithm is modeled on the process of cooling or annealing metals. The basic idea is to cool a metal from a molten to a solid form by carefully following a prescribed cooling schedule such that the resulting product is at a minimum energy state. Products with higher energy states often contain defects which detract from their strength. The manner of defining the energy function and the temperature or annealing schedule that is used to cool the metal are critical to the success of metallurgic annealing.

For the SA algorithm, the energy function is the objective function (to be minimized). SA begins with an initial solution and iteratively perturbs or *transitions* this solution in hopes of eventually finding a solution of lower energy. The variance of this transition is a function of a temperature parameter, T and is great for high temperatures and much smaller for lower temperatures. The parameter T follows an exponential cooling schedule so that during the early “hot” iterations SA may accept solutions which are

Figure 17: Interset Graph For $\gamma = 4$

drastically different from the incumbent, while during the later “cool” iterations less variability is permitted. The algorithm terminates when no further energy reduction is realized. The global aspect of SA is due to its ability to accept transition solutions which are worse than the incumbent thereby avoiding local entrapment. Figure 18 shows a general pseudo-code for a simulated annealing procedure for the FAP.

Not specified in Figure 18 are the parameters α, β , the initial temperature T , the energy function C , and the transition mechanism (the method which generates new solutions). Parameter α controls the cooling rate and is typically chosen between 0.8 and 0.99 [80]. Often, T and α are chosen together by a trial and error process such that during initial iterations nearly any perturbation of the solution is accepted while during later iterations only transitions which improve the energy function will be accepted.

Parameter β is the number of iterations performed prior to a reduction in temperature and is selected empirically so that run-times are acceptable. Quellmalz, Knälmann and Müller [70] restrict the total run-time to a fixed value. They determine the number of temperature steps by using a trial run to compute the time for one **Do** iteration and then divide the total run-time by this result. The Eindhoven CALMA team [30] do not restrict total run-time and as a result, the test problems with constraint violation costs which vary by orders of magnitude experience many, small temperature steps and subsequently converge on a solution extremely slowly.

```

procedure SimulatedAnnealing( $G, \alpha, \beta, T$ )
1    $f = \text{BuildInitialSolution } (G);$ 
2    $C(f) = \text{SolutionEnergy};$ 
3    $T = \text{InitializeTemperature};$ 
4   DO Until no change in  $C$ 
5     for  $i = 1 \dots \beta$ 
6        $f^i = \text{Perturb}(f, T) ;$ 
7       if  $C(f^i) \leq C(f)$  then  $f \leftarrow f^i;$ 
8       elseif  $e^{\frac{C(f)-C(f^i)}{T}} \geq \text{uniform}(0, 1)$  then  $f \leftarrow f^i;$ 
9     rof;
10     $T \leftarrow \alpha T;$ 
11  OD;
12  return  $f;$ 
end SimulatedAnnealing;

```

Figure 18: Pseudo-Code For Simulated Annealing

The energy function for an FAP could be the objective of any of the IP formulations presented in Section 5. In the papers by the Eindhoven CALMA team (Quellmalz, Knälmann and Müller, and Hurley and Smith [51]) an energy function is given which incorporates the interference constraints with a penalty $a_{p_{i,j}}$ for violating the constraint on edge (v_i, v_j) such as shown in IP-4. The Eindhoven report also considers a minimum order energy function as given in IP-2.

All three reports generate transitions by selecting a point in a well defined neighborhood of the incumbent solution. The neighborhood considered for interference energy functions is developed by selecting a vertex at random and randomly assigning a frequency to it from its domain. This neighborhood is obviously restricted in size and well defined.

The second neighborhood is used by the Eindhoven team for minimum order energy functions. For this objective, we assume that the incumbent solution is feasible. A transition is defined by randomly selecting a frequency f^* from the incumbent solution and then deleting f^* from all vertices it has been assigned to. The neighbor is developed by finding a replacement frequency from the union of the domains minus f^* which incurs the least number of constraint violations. Ties are broken by choosing the frequency which maximizes the size of the smallest domain which belongs to the violating vertices. Obviously the solutions in this neighborhood will all have

order one smaller than the incumbent. Even though this neighborhood is bounded in size, it may not be feasible. Hence, the first neighborhood can be switched to if the transitions become too infeasible.

SA appears to work well for infeasible FAPs that are well conditioned; that is have constraint violation costs which vary by less than an order of magnitude. The solutions are not often optimal but are typically very close to optimal. If run-time is not constrained as in [30], the solution quality may suffer for problems with large domains.

7.5.2 Tabu Search

Tabu Search (TS), defined by Glover in [41, 42] was used by Castelino, Hurley and Stephens [17] for the minimum interference FAP and by Bouju et al. [12] and the Eindhoven CALMA team [30] on both minimum interference and minimum order FAP formulations. Tabu search is similar to SA in that an incumbent solution is iteratively replaced by a new solution picked from a well defined neighborhood of the incumbent. The TS selection rule also incorporates randomness and terminates when no improvement is realized for some number of iterations. However, unlike SA, TS is greedy and only selects a new solution if it is improving. If there are no improving solutions in the neighborhood, TS picks the solution that degrades the cost by the least. For the FAP, the neighborhoods may be defined for TS in the same manner as for SA. The TS algorithm sequences through the neighbors of the incumbent in a random manner. The Eindhoven team uses a *first improvement strategy*, that is, the first solution encountered that offers an improvement to the incumbent is selected to replace the incumbent.

To prevent cycling, TS maintains a *Tabu list* of the changes made during the past r iterations, where r is determined empirically. If a solution in the neighborhood uses a change in the Tabu list, it is not eligible to be selected unless it results in a cost that is better than any encountered thus far. For the minimum interference formulation, the Tabu list consists of all of the vertices that were reassigned during the past r solutions. For the minimum order formulation, the Tabu list consists of the frequencies deleted in the last r solutions.

The Eindhoven CALMA team found that TS worked well for minimum order FAP formulations, especially if the interference graph was sparse. However, on the minimum interference formulations convergence was slow and solutions not satisfactory. Bouju et al. also found Tabu to converge very slowly. This is probably due to the large size of the neighborhoods which often must be completely searched before the next iterate can be selected.

7.5.3 Tabu Thresholding

Looking for faster convergence for minimum interference FAP formulations, Castelino and Stephens [18] investigate the use of *Tabu Thresholding* (TT), a variant of Tabu Search described by Glover [43]. For each iteration of TT, two phases are executed, the *improving* phase and the *mixed* phase.

The improving phase functions very much like TS except that no Tabu list is maintained. Instead, the neighborhood is partitioned into n subsets, one for each vertex, which in-turn are partitioned into b blocks where b is chosen to be much smaller than n . Beginning with the first block, a subset is chosen from it at random. If the best solution in this subset improves the incumbent solution, then it replaces the incumbent and the neighborhood of the new incumbent is searched in a like manner. If the best solution in a subset fails to improves the incumbent solution, then it is rejected and the next subset is randomly selected. The improving phase ends when either all subsets of all blocks have been considered or when k iterations have been completed, where k is chosen to restrict the run-time. The solution at the end of the improving phase is used to start the mixed phase.

The mixed phase begins by randomly reordering all of the subsets, irrespective of the blocks, making sure to place the last improving subset at the end of the list. Then TT proceeds as in the improving phase only now the incumbent is always updated by the best solution in the chosen subset, even if makes no improvement upon the incumbent. This phase may terminate if a solution is found that is better than any encountered thus far. Otherwise, it terminates when t iterations have been performed, where t is chosen at random between fixed bounds L and U . The solution at the end of the mixed phase is then used to start another iteration of the improving phase. The algorithm continues in this manner alternating between the improvement and mixed phases until no improvement is made on the best global solution for some number of iterations.

Castelino and Stephens showed that TT offered a significant improvement over TS for minimum interference FAPs in that it converged more rapidly.

7.5.4 Genetic Algorithms

Genetic algorithms (GA) were introduced by Holland [50] for machine learning applications. They were soon being used in many problems including combinatorial optimization [44].

The GA approach is simple and modeled on the evolution of a species via

```

procedure Genetic( $G, \alpha$ )
1   Initialize  $P$ ;
2    $c_i = \text{cost}(P_i), i = 1, \dots, \alpha$ ;
3    $C = \min\{c_i\}$ ;
4    $f \leftarrow \arg \min\{c_i\}$ ;
5   DO until no improvement in  $C$ 
6        $Q = \text{SelectParents}(P)$ ;
7        $R = \text{CreateChildren}(Q)$ ;
8        $P \leftarrow \text{MergeChildren}(R, P)$ ;
9        $c_i = \text{cost}(P_i), i = 1, \dots, |P|$ ;
10      if  $\min\{c_i\} < C$ 
11           $f \leftarrow \arg \min\{c_i\}, C \leftarrow \min\{c_i\}$ ;
12      fi
13  OD;
14  return  $f$ ;
end Genetic;

```

Figure 19: Pseudo-Code For A Genetic Algorithm

inheritance and mutation. A population of solutions P is considered. These solutions are initialized by some heuristic technique and may or may not be feasible. Each solution x in the population is referred to as a *chromosome*. The algorithm *selects* a subset of P denoted by Q such that good solutions are more likely to be selected for Q than are bad ones. The solutions in Q are called *parent chromosomes*. The parent solutions are then used to *create offspring* or *children chromosomes* which are placed in set R . Finally, the children are *merged* back into the original population using a heuristic for selecting which children to add to P and which members of P to delete. Bounds on the size of P are used to limit run-time. Each time P is updated, the lowest cost solution thus far is kept as the global solution f . These steps are repeated until no change in the global cost C occurs for several iterations. A Pseudo-code for GA is provided in Figure 19.

For the FAP, the chromosome solution x is a vector represented by one of 3 methods:

1. Vertex Sequenced. Here x is integer valued of length n where the x_i element (*gene*) is an integer representing the channel number assigned to vertex v_i .
2. Channel Sequenced. Here x is integer valued of length n where the first

k_1 elements of x represent the vertices that receive the lowest assigned channel, the following k_2 elements represent the vertices assigned to the second lowest assigned channel, and so on until the last k_m elements represent the vertices assigned to the highest assigned channel. In this representation, m is the number of channels in the assignment.

3. Binary. Here x is binary valued of length bn where each channel is represented by a b -bit code so that the first b elements of x represent the channel assigned to v_1 , the second b elements represent the channel assigned to v_2 and so on. The value of b is determined by the size of the largest domain. So if, for example, the largest domain contained 60 channels, then $b = 6$ would be needed ($2^5 = 32$ and $2^6 = 64$).

The children chromosomes are created from the parent chromosomes using two operations; *crossover* and *mutation*. Crossover results when elements or genes of 2 parent vectors are exchanged. The location and quantity of genes to be exchanged depend upon the crossover technique (of which there are many) and the representation of x . When all parents share the same value for a particular gene, all the children will likewise share that value. *Mutation* is an operation that prevents this from happening by randomly selecting a gene and changing its value.

Crompton, Hurley, and Stephens [25] developed a GA for the minimum interference FAP using the channel sequenced chromosomal representation. They found that this particular representation worked well but was difficult to implement in a data structure. Castelino, Hurley, and Stephens [17] report that their TS algorithm outperformed the GA used by Crompton, Hurley, and Stephens on the same data.

Smith, Kapsalis, and Rayward-Smith [82] used all three chromosome representations in a GA for minimum order and minimum interference FAP formulations. They found in general, the vertex sequenced representation to be the most robust and simplest to implement. The binary representation was by far the worst. Furthermore, they find that genetic algorithms appear to be less sensitive to parameter tuning than SA or TS but that the quality of solutions is very dependent on the crossover and representation methods used. Improvement in convergence is expected if a local search phase inserted into the GA.

7.5.5 Neural Networks

Kurokawa and Kozuka [59] implement a Hopfield type *neural network* (NN) to solve a satellite communications FAP. Typically NNs are used in pat-

tern matching, classification, and machine learning applications. Their use in combinatorial optimization problems has been limited by poor performance. Kurokawa and Kozuka consider the problem of assigning channels to two satellites such that interference is minimized. The channels of one satellite are fixed and the channels on the second are varied but the available spectrum is the same for both satellites. The Hopfield network was able to quickly converge on optimal solutions for this problem. It remains to be seen how these results could be extended to more general FAPs.

7.6 Polyhedral Methods

7.6.1 Linear Relaxations

Consider IP-4. If the inequality constraints are represented by $Ax \leq c$, and the equality constraints by $Bx = d$, where $x \in \{0,1\}^{nm}$, n is the number of vertices, and m the number of frequencies available, then a feasibility IP may be defined as follows:

IP- 5.

$$\text{find } x \in \{0,1\}^n \quad (51)$$

subject to

$$Ax \leq c \quad (52)$$

$$Bx \leq d \quad (53)$$

Karmarkar has shown [54] that integer program feasibility problems of the form IP-5 may be solved using an interior point method on a linear relaxation of the IP with a potential function for the objective. The process is as follows. The integrality constraints are transformed from $\{0,1\}$ to $\{-1,1\}$. The linear relaxation of the subsequent problem is formed by allowing $-1 \leq x_i \leq 1, i = 1, 2, \dots, n$. A non-convex *potential function* $\phi(x)$ is developed which has the property that any minima of the function are feasible solutions to IP-5. A local minimum of ϕ is found by successively solving quadratic approximations of it over an ellipsoid inscribed in the relaxation polytope and centered on a point interior to the relaxation. At each step the solution to the quadratic approximation is rounded to $\{-1,1\}$ and if the rounded solution is not feasible to IP-5, then this point becomes the next iterate and a new ellipsoid is centered on it. If the rounded solution is feasible to IP-5, then it is a local minimum of the relaxation.

Multiple local solutions of IP-5 may be obtained by modifying the potential function in some subtle manner and then solving the new relaxation. The best local solution is kept as a minimizer of IP-5. Alternatively, the objective (for minimum order minimize $\sum_{f \in F} y_f$) could be converted to a linear constraint as $\sum_f y_f \leq M$ where M is a bound on the order. Successively solving this new feasibility problem while decreasing M to the best obtained order will approach the minimum order solution. Warners et al. [90] use Karmarkar's technique on minimum order, and minimum span FAPs with limited success. The algorithm converges extremely slowly due to small ellipsoid constructions. In order to improve their results, a quadratic relaxation was developed.

7.6.2 Quadratic Relaxations

Quadratic Formulation. Given an interference graph $G(V, E)$ and forbidden channel separations T , define a new decision variable that is a function of the decision variable x_{if} :

$$q_{ifjg} = \begin{cases} 1 & \text{if } x_{if} = 1, x_{jg} = 1, \text{ and } (v_i, v_j) \in E(G), |f - g| \in T \\ 0 & \text{otherwise} \end{cases}.$$

for $i, j \in I, i \neq j$, and $f, g \in F$.

Define the vector $x = \{x_{11}, x_{12}, \dots, x_{1m}, x_{21}, x_{22}, \dots, x_{nm}\}$ where $n = |I|$ and $m = |F|$. We say that the assignment $F \rightarrow x$ has no interference if

$$\sum_{i=1}^n \sum_{f=1}^m \sum_{j=1}^n \sum_{g=1}^m x_{if} x_{jg} q_{ifjg} = 0. \quad (54)$$

Consider an edge (v_i, v_j) in the interference diagram. If we define the matrix

$$Q_{ij} = \begin{pmatrix} q_{i1j1} & q_{i1j2} & \dots & q_{i1jm} \\ q_{i2j1} & q_{i2j2} & \dots & q_{i2jm} \\ \vdots & & \ddots & \\ q_{imj1} & q_{imj2} & \dots & q_{injm} \end{pmatrix}$$

and let

$$Q = \begin{pmatrix} Q_{11} & Q_{12} & \dots & Q_{1n} \\ Q_{21} & Q_{22} & \dots & Q_{2n} \\ \vdots & & \ddots & \\ Q_{n1} & Q_{n2} & \dots & Q_{nn} \end{pmatrix}$$

then equation (54) is equivalent to

$$x^T Q x = 0. \quad (55)$$

QAP- 1.

$$\text{minimize } x^T Q x \quad (56)$$

subject to

$$\sum_{f \in F} x_{if} = 1 \quad \forall i \in I \quad (57)$$

$$x_{if} \in \{0, 1\} \quad \forall i \in I, \forall f \in F \quad (58)$$

The optimal solution to QAP-1 is, of course, zero and any optimal solution is a feasible assignment but not necessarily optimal to the original IP problem.

Warners et al. [90] applied Karmarkar's approach described in Section 7.6.1 to the quadratic relaxation and were able to obtain good solutions for some instances of the CALMA problems. Convergence was much improved over the linear relaxation but was still not as fast as some of the meta-heuristics described in Section 7.5. Also the quadratic formulation requires substantially more memory than the linear one, certainly a consideration when dealing with modern computer architectures. Warners et al. report that convergence may improve with better methods of adapting the potential function and with the addition of a local search.

8 Evaluation of Algorithms

The ability to generate test problems with known optimal solutions is always of great interest to algorithm designers. In this section we address two fundamental approaches to developing test problems for frequency assignment algorithms.

Zoellner and Beall [95] developed a problem generator method called *vertex saturation* which generates test instances for frequency assignment problems that have co-channel and first adjacent channel constraints only. These graphs are in fact 2-level unit disk graphs. They are created by generated a large number of random points on a 2-dimensional grid. These points are candidate vertices. The generator considers each candidate in sequence and attempts to color it the smallest color from a set of colors \mathcal{F} such that both of the following hold:

1. The Euclidean distance between the current point and all previous points that are colored the same color is greater than some threshold $D(0)$
2. The Euclidean distance between the current point and all previous points that are colored the same color or one color different is greater than $D(1)$ where $D(1) < D(0)$.

If there is no feasible color for a candidate, then the candidate is rejected and the next point considered. If the point can be colored, then the candidate becomes a vertex in the constructed graph with the feasible color assigned to that vertex. The procedure continues until all candidates have been considered. A graph is considered to be saturated when candidates are no longer being added as vertices. Figure 20 illustrates a construction of a graph with 47 vertices reaching saturation after about 500 candidate points have been considered. A saturated graph will have used all frequencies possible in the span of \mathcal{F} . Hence, the constructed graph will have a minimum span assignment that is precisely the colors selected in the construction.

When $\mathcal{F} = \{1, 2, \dots, r\}$, the optimal solutions to some of these constructed graphs are exactly \mathcal{F} , i.e., there are no *holes* in the spectrum. Clearly, with no holes, the order of the solution is equal to $spt(G) + 1$. Note that this does not imply that the minimum span assignment is also a minimum order assignment: $\chi_T(G) \neq spt(G) + 1$. Indeed, if we apply an algorithm that finds $\chi_T(G)$, then the resulting span may, and most likely will, be larger than $spt(G)$.

Another method, called *Generating Radiolink frequency Assignment Problems Heuristically* or GRAPH for short, developed by Benthem [5, 6, 7], constructs a multigraph with an unknown solution but a known lower bound. GRAPH first constructs a large clique on some predefined number of vertices, then several, *independent*, smaller cliques are added until all vertices are members of exactly one clique. Edges are randomly added to connect the cliques until the ratio of co-channel edges to vertices reaches the threshold $|E(G_0)| \approx 6|V|$. The lower bound on the minimum order and span is the size of the largest clique. A graph with an exact solution can be created by first coloring the cliques before inter-connecting them (since each clique is independent, they may be colored from the same colors) and then only adding edges between cliques that maintain feasibility.

GRAPH has the ability to generate graphs with set colorings; that is, the feasible color spectrum is a finite set and not necessarily consecutive. It can also develop preassigned frequencies; transmitters that have a frequency already assigned and that cannot be reassigned without some specified penalty.

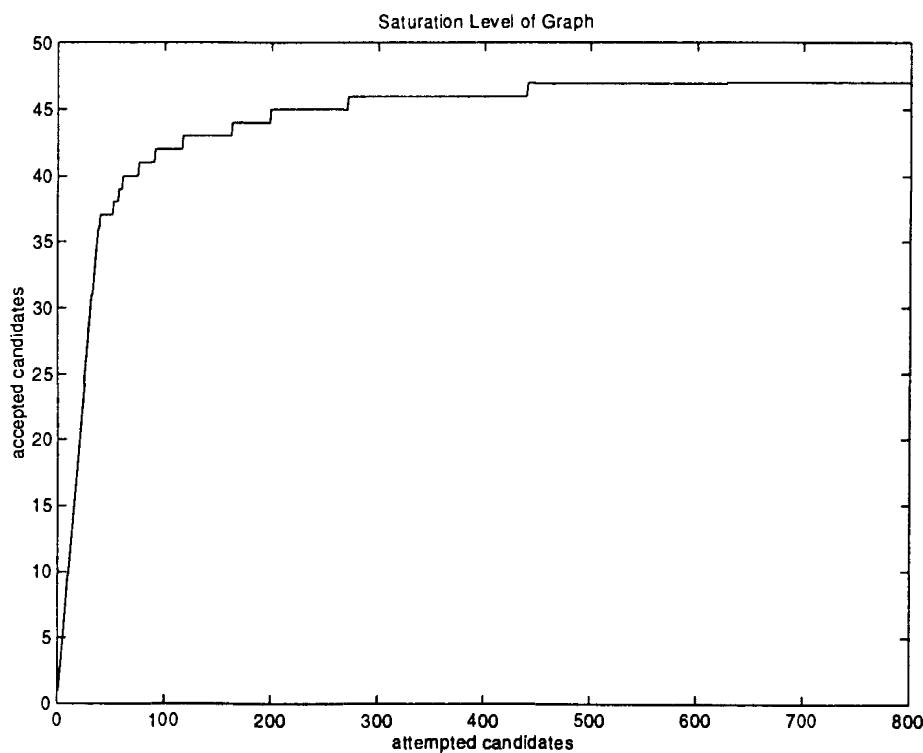


Figure 20: Typical Saturation Curve For Vertex Saturation Graph Generator

When generating test graphs, we would like to construct cases that are fairly realistic. Benthem et al. [6] found that for CALMA type scenarios, realistic graphs have a ratio of co-channel edges to number of vertices approximately equaling 6. However, they do not mention what the ratios of $\frac{|E(G_k)|}{|V(G)|}$ for $k = 1, 2, \dots, K$ are. In other words, how dense should the adjacent channel graphs be to represent realistic problems? Obviously this will in some degree depend upon the situation. However, discounting spurious filtering and antenna effects, the graph density is significantly a function of the waveform modulation type. Further research is needed to characterize how modulation type and graph density are related.

References

- [1] K.I. Aardal, A. Hipolito, C.P.M. Van Hoesel, B. Jansen, C. Roos and T. Terlaky, A branch-and-cut algorithm for the frequency assignment

- problem, *EUCLID CALMA Project*, Delft and Eindhoven Universities of Technology, The Netherlands, (1995).
- [2] E. Aarts and J. Korst, *Simulated Annealing and Boltzman Machines: A Stochastic Approach To Combinatorial Optimization and Neural Computing*, John Wiley and Sons (1989).
 - [3] I. Baybars , Optimal assignment of broadcasting frequencies, *European Journal of Operations Research* , Vol. 9, (1982) pp. 257-263.
 - [4] E. A. Bender and H. S. Wilf, A theoretical analysis of backtracking in the graph coloring problem, *Journal of Algorithms* **6**, (1985), pp. 275-282.
 - [5] H.P. Benthem, GRAPH: Generating radio link frequency assignment problems heuristically, Master's Thesis, Faculty of Technical Mathematics and Informatics, Delft University, Delft, The Netherlands, (1995).
 - [6] H. van Benthem, A. Hipolito, B. Jansen, C. Roos, T. Terlaky and J. Warners, EUCLID CALMA technical report, GRAPH: a test case generator, generating radiolink frequency assignment problems heuristically, *EUCLID CALMA Project*, Delft and Eindhoven Universities of Technology, The Netherlands, (1995).
 - [7] H. van Benthem, A. Hipolito, B. Jansen, C. Roos, T. Terlaky and J. Warners, GRAPH, a test problem generator for the radiolink frequency assignment problem, *EUCLID CALMA Project Supplemental Report 2.3.2a:*, Delft and Eindhoven Universities of Technology, The Netherlands, (1995).
 - [8] L.A. Berry and D. H. Cronin, The spectrum cost of frequency distance rules, *IEEE International Symposium on Electromagnetic Compatibility*, (1983) pp. 75-78.
 - [9] D.P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed., (Prentice-Hall, 1992).
 - [10] B. Bollobás and A. J. Harris, List colorings of graphs, *Graphs and Combinatorics* **1**, (1985) pp. 115-187.
 - [11] I. Bonias, *T-colorings of complete graphs*, Ph.D. Thesis, Department of Mathematics, Northeastern University, Boston, MA (1991).

- [12] A. Bouju, J.F. Boyce, C.H.D. Dimitropoulis, G. vom Scheidt, and J.G. Taylor, Tabu search for the radio link frequency assignment problem, *Applied Decision Technologies, London [ADT95], UNICOM Conference*, (1995).
- [13] F. Box, A heuristic technique for assigning frequencies to mobile radio nets, *IEEE Trans. Vehicular Technology*, Vol. VT-27, (1978), pp. 57-74.
- [14] D. Brelaz , New methods to color the vertices of a graph, *Communications ACM*, Vol. 22, (1979) pp. 251-256.
- [15] S.H. Cameron, The solution of the graph coloring problem as a set covering problem, *IEEE Transactions on Electromagnetic Compatibility*, Vol EMC-19, (1973) pp. 320-322.
- [16] F. Carmassi and L. Tomati , A theory of frequency assignment in broadcasting network planning, *European Broadcast Union Technical Review*, No. 198, (Apr. 1983) pp. 72-81.
- [17] D.J. Castelino, S. Hurley and N.M. Stephens, A Tabu search algorithm for frequency assignment, *Annals of Operations Research*, Vol 41, (1993), pp. 343-358.
- [18] D. Castelino and N. Stephens, Tabu thresholding for the frequency assignment problem, *Meta-Heuristics: Theory and Applications* (Edited by I.H. Osman and J.P. Kelly), Kluwer Academic Publishers 1996.
- [19] V. Chvátal, Perfectly ordered graphs, *Annals of Discrete Mathematics*, 21, (1984) pp. 63-65.
- [20] L.W. Couch, *Modern Communications Systems: Principles and Practices*, Prentice-Hall, Inc., (1995).
- [21] M.B. Cozzens and F.S. Roberts, T -colorings of graphs and the channel assignment problem, *Congressus Numerantium*, Vol 35 (1982), pp. 191-208.
- [22] M.B. Cozzens and F.S. Roberts, Double semiorders and double indifference graphs, *SIAM Journal Algebraic Discrete Methods* 3 (1982) pp. 566-583.
- [23] M.B. Cozzens and F.S. Roberts, Greedy algorithms for T -colorings of complete graphs and the meaningfulness of conclusions about them, *Journal of Combinatorics, Information and System Sciences*, Vol 16 No. 4, (1991), pp. 286-299.

- [24] M.B. Cozzens and D.I. Wang, The general channel assignment problem, *Congressus Numerantium*, Vol 41 (1984), pp. 115-129.
- [25] W. Crompton, S. Hurley and N.M. Stephens, Frequency assignment using a parallel genetic algorithm, *Proceedings of Natural Algorithms in Signal Processing*, (1993).
- [26] C.E. Dadson, J. Durkin and R.E. Martin, Computer prediction of field strength in the planning of radio systems, *IEEE Trans. Vehicular Technology*, Vol. VT-24, (1975), pp. 1-8.
- [27] D. de Werra and Y. Gay, Chromatic scheduling and frequency assignment, *Discrete Applied Mathematics* **49**, (1994) pp. 165-174.
- [28] J. P. Doignon, Threshold representations of multiple semiorders, *SIAM Journal Algebraic Discrete Methods* **8** (1987) pp. 77-84.
- [29] R.B. Eggleton, P. Erdős and D.K. Skilton, Coloring the real line, *Journal of Combinatorial Theory, Series B*, **39**, (1985), 86-100.
- [30] Eindhoven RLFAP Group, Radio link frequency assignment project, *EUCLID CALMA Project Report 2.3.3 Local Search*:, Eindhoven University of Technology, The Netherlands, (1995).
- [31] A. Eisenblätter, A frequency assignment problem in cellular phone networks (extended abstract), *Network Design, Connectivity and Facility Location, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 35, American Mathematical Society (1997).
- [32] P. Erdős, A. L. Rubin and H. Taylor, Choosability in graphs, *Congressus Numerantium* **26**, (1979), pp. 125-157.
- [33] L.R. Foulds, *Graph Theory Application*, Springer-Verlag New York Inc., (1992).
- [34] R.A. Frazier, Compatibility and the frequency selection problem, *IEEE Trans. Electromagnetic Compatibility*, Vol. EMC-17, (1975), pp. 248-275.
- [35] D. R. Fulkerson and O.A. Gross, Incidence matrices and interval graphs, *Pacific Journal of Math* **15**, (1965) pp. 835-855.
- [36] Z. Füredi, J. R. Griggs and D.J. Kleitman, Pair labellings with given distance, *SIAM Journal of Discrete Mathematics* **2**, (1989) pp. 491-499.

- [37] A. Gamst, Some lower bounds for a class of frequency assignment problems, *IEEE transactions on vehicular technology*, Vol. VT-35, No. 1 (Feb. 1986), pp. 8-14.
- [38] A. Gamst and K. Ralf, Computational complexity of some interference graphs, *IEEE Transactions on Vehicular Technology*, Vol 39 No. 2, (1990) pp. 140-149.
- [39] A. Gamst and W. Rave, On frequency assignment in mobile automatic telephone systems, *GLOBCOM 82, IEEE Global Telecommunications Conference*, Vol 1, (1982) pp. 309-315.
- [40] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide To The Theory of NP-Completeness*, W.H. Freeman and Company, New York, NY (1979).
- [41] F. Glover, Tabu search - Part I. *ORSA Journal on Computing* **1**: 190-206, 1989.
- [42] F. Glover, Tabu search - Part II. *ORSA Journal on Computing* **2**: 4-32, 1990.
- [43] F. Glover, Tabu Thresholding: Improved search strategies by non-monotonic search trajectories, *ORSA Journal on Computing* to appear.
- [44] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading MA (1989).
- [45] M. Grötschel, L. Lovász, and A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* **1**, (1981) 169-197.
- [46] W.K. Hale, Frequency assignment: theory and applications, *Proc. of The IEEE* Vol 68, No. 12, (1980) pp. 1497-1514.
- [47] W.K. Hale, New spectrum management tools, *IEEE International Symposium on Electromagnetic Compatibility*, Boulder, CO (1981) pp. 47-53.
- [48] F. Harary, *Graph Theory*, Addison-Wesley Publishing Company, Reading, MA (1969).
- [49] A. Hertz, COSINE: a new graph coloring algorithm, *Operations Research Letters* **10**, (1991) pp. 411-415.

- [50] H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI (1975).
- [51] S. Hurley and D. H. Smith, Fixed spectrum frequency assignment using natural algorithms, *Genetic Algorithms in Engineering Systems: Innovations and Applications*, IEE Conference Publication No. 414, (1995).
- [52] T.R. Jensen and B. Toft, *Graph Coloring Problems*, John Wiley and Sons, New York, (1995).
- [53] P.K. Johri, An insight into dynamic channel assignment in cellular mobile communications systems, *European Journal of Operations Research*, **74**, (1994) pp. 70-77.
- [54] , N. Karmarkar, An interior point approach to \mathcal{NP} -Complete problems - part 1, *Contemporary Mathematics*, Vol 114, (1990) pp. 297-308.
- [55] I. Katzela and M. Naghshineh, Channel assignment schemes for cellular mobile telecommunication systems: a comprehensive survey, *IEEE Personal Communications* (June 1996), pp. 10-31.
- [56] D. Karger, R. Motwani and M. Sudan, Approximate graph coloring by semidefinite programming, *INCOMPLETE*.
- [57] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, Optimization by simulated annealing, *Science*, Vol 220, (1983) pp. 671-680.
- [58] Y. Kishi, T. Mizuike and F. Watanabe, A unified approach for frequency assignment of cellular mobile networks, *Third Annual International Conference on Universal Personal Communications, San Diego, CA*, (1994) pp. 563-567.
- [59] T. Kurokawa and S. Kozuka, Use of neural networks for the optimum frequency assignment problem, *Electronics and Communications in Japan*, Part 1, Vol. 77, No. 11, (1994).
- [60] T.A. Lanfear, *Graph Theory and Radio Frequency Assignment*, Technical Report, Allied Radio Frequency Agency NATO Headquarters, B-1110, Brussels, Belgium, (1989).
- [61] D.S.P. Leung, Application of the partial backtracking technique to the frequency assignment problem, *IEEE International Symposium on Electromagnetic Compatibility*, Boulder, CO (1981) pp. 70-74.

- [62] D. D. Liu, Graph homomorphisms and the channel assignment problem, Ph.D. Thesis, Department of Mathematics, University of South Carolina, Colombia, SC (1991).
- [63] D.W. Matula and L.L. Beck , Smallest-last ordering and clustering and graph coloring algorithms, *Journal of The ACM*, Vol. 30, (1983) pp. 417-427.
- [64] B.H. Metzger, Spectrum management technique, *paper presented at the 38th National ORSA Meeting*, Detroit, MI, (1970).
- [65] L.C. Middlecamp, UHF taboos - history and development, *IEEE Transactions on Consumer Electronics*, Vol CE24, (1978) pp. 514-519.
- [66] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons, Inc., (1988).
- [67] R.J. Opsut and F.S. Roberts, I-colorings, I-phasings, and I-intersection assignments for graphs, and their applications, *Networks*, Vol. 13 (1983) pp. 327-345.
- [68] P.M. Pardalos, F. Rendl and H. Wolkowicz, The Quadratic Assignment Problem: A Survey and Recent Developments, *Quadratic Assignment and Related Problems, Panos M. Pardalos and Henry Wolkowicz (eds.)*, *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 16 , American Mathematical Society, (1994) pp. 317-342.
- [69] J. Peemöller, A correction to Brelaz's modification of Brown's coloring algorithm, *Communications of the ACM*, Vol. 26, No. 8, (1983) pp. 595-597.
- [70] A. Quellmalz, A. Knälmann and B. Müller, Efficient frequency assignment with simulated annealing, *IEE Ninth International Conference on Antennas and Propagation*, Eindhoven, The Netherlands, Vol. 2, (1995).
- [71] A. Raychaudhuri, Intersection assignments, T -coloring, and powers of graphs, Ph.D. Thesis, Department of Mathematics, Rutgers University, New Brunswick, NJ, (1985).
- [72] A. Raychaudhuri, Further results on T -coloring and frequency assignment problems, *SIAM Journal on Discrete Mathematics*, Vol. 7, (1994), pp. 605-613.

- [73] F.S. Roberts, On the Compatibility between a graph and a simple order, *Journal Combinatorial Theory*, **11** (1971), pp 28-38.
- [74] F.S. Roberts, On the mobile radio frequency assignment problem and the traffic light phasing problem, *Annals of New York Academy of Sciences* Vol. 319 (1979) pp 466-483.
- [75] F.S. Roberts, From garbage to rainbows: generalizations of graph coloring and their applications, in Y. Alavi, G. Chartrand, O.R. Oellerman and A.J. Schwenk (eds.) *Graph Theory, Combinatorics, and Applications*, Vol. 2 (Wiley, New York, 1991), pp. 1031-1052.
- [76] F.S. Roberts, T -colorings of graphs: recent results and open problems, *Discrete Math.* **93**, (1991) pp. 229-245.
- [77] F.S. Roberts, No-hole 2-distant colorings, *Mathl. Comput. Modelling* Vol. 17, No. 11 (1993) pp. 139-144.
- [78] D. Sakai, Generalized graph colorings and unit interval graphs, Ph.D. Thesis, RUTCOR, Rutgers University, New Brunswick University, NJ (1992).
- [79] D. Sakai and C. Wang, No-hole $(r+1)$ -distant colorings, *Discrete Math.* **119**, (1993), pp. 175-189.
- [80] S. S. Skiena, *The Algorithm Design Manual*, TELOS, Springer-Verlag New York Inc., (1998).
- [81] D.H. Smith , Graph coloring and frequency assignment, *ARS Combinatoria*, Vol. 25C, (1988) pp. 205-212.
- [82] G.D. Smith, A. Kapsalis, V.J. Rayward-Smith, Radio link frequency assignment project, *EUCLID CALMA Project Report 2.1 Genetic Algorithm Approaches to Solving the Radio Link Frequency Assignment Problem*, University of East Anglia, UK (1995).
- [83] S. Stahl, n-tuple colorings and associated graphs, *J. Combinatorial Theory* **20** (1976) pp. 185-203.
- [84] B.A. Tesman, T -colorings, list T -colorings, and set T -colorings of graphs, Ph.D. Thesis, Department of Mathematics, Rutgers University, New Brunswick, NJ, (1989).
- [85] B.A. Tesman, Applications of forbidden difference graphs to T -colorings, *Congrussus Numerantium* **74** (1990) pp. 15-24.

- [86] B. A. Tesman, Set T -colorings, *Congrussus Numerantium* **77** (1990) pp. 229-242.
- [87] S. Tiourine, C. Hurkens and J. K. Lenstra, An overview of algorithmic approaches to frequency assignment problems, *EUCLID CALMA Project Overview Report*, Delft University of Technology, The Netherlands, (1995).
- [88] D. Sakai Troxell, No-hole k-tuple $(r+1)$ -distant colorings, *Discrete Applied Math.* **64**, (1996) pp. 67-85.
- [89] D. -I. Wang, The channel assignment problem and closed neighborhood containment graphs, Ph.D. Thesis, Northeastern University, Boston, MA (1985).
- [90] J.P. Warners, T. Terlaky, C. Roos and B. Jansen, *A Potential Reduction Approach to The Frequency Assignment Problem*, Technical Report 95-98, Faculty of Technical Mathematics and Informatics, Delft University, Delft, The Netherlands, (1995).
- [91] D. Werra, Y. Gay, Chromatic scheduling and frequency assignment, *Discrete Applied Math.* **49**, (1994) pp. 165-174.
- [92] H. S. Wilf, Backtrack: an $O(1)$ expected time algorithm for the graph coloring problem, *Information Processing Letters* **18**, (1984) pp. 119-121.
- [93] A. Wisse, Mathematical model for the radio link frequency assignment problem, *EUCLID CALMA Project*, Delft University of Technology, The Netherlands, (1994).
- [94] J.A. Zoellner, Frequency assignment games and strategies, *IEEE Transactions on Electromagnetic Compatibility*, Vol EMC-15, (1973) pp. 191-196.
- [95] J.A. Zoellner and C.L. Beall, A breakthrough in spectrum conserving frequency assignment technology, *IEEE Transactions on Electromagnetic Compatibility*, Vol EMC-19, (1977) pp. 313-319.

Algorithms for the Satisfiability (SAT) Problem

Jun Gu

*Dept. of Electrical and Computer Engineering,
University of Calgary,
Calgary, Alberta T2N 1N4, Canada*
E-mail: j.gu@computer.org

Paul W. Purdom

*Dept. of Computer Science,
Indiana University, Bloomington, Indiana 47405*
E-mail: pwp@cs.indiana.edu

John Franco

*Dept. of Computer Science,
University of Cincinnati, Cincinnati, OH 45221-0008*
E-mail: John.Franco@UC.Edu

Benjamin W. Wah

*Dept. of Electrical and Computer Engineering,
Univ. of Illinois at Urbana-Champaign,
Urbana, IL 61801-3082*
E-mail: b-wah@uiuc.edu

Contents

1	Introduction	382
2	Constraint Satisfaction Problems	383

3 Formulations and Algorithm Classes	385
3.1 Preliminaries	385
3.2 Formulations of SAT	386
3.3 Basic SAT Algorithm Classes	390
3.4 Parallel SAT Algorithms	393
3.5 Algorithm Categories	396
3.6 Performance Evaluation	398
4 SAT Input Models	400
4.1 Random Input Models	401
4.2 Hardness	402
4.3 Comparison of Random Input Models	402
4.4 Practical Input Models	405
5 Splitting and Resolution	406
5.1 Resolution	407
5.2 Backtracking	409
5.3 Backtracking and Resolution	412
5.4 Clause Area	413
5.5 Improved Techniques for Backtracking	413
5.6 Some Remarks on Complexity.	415
6 Local Search	416
6.1 Framework	416
6.2 Early Work	417
6.3 A Three-Level Search Space Model	425
6.4 Four Components in Local Search	427
6.5 Randomized Local Search	432
6.6 Randomized Local Search with Trap Handling	437
6.7 Greedy Local Search	440
6.8 Tabu Local Search	441
6.9 Local Search for <i>DNF</i> formulas	442
7 Nonlinear Unconstrained Methods	443
7.1 <i>UniSAT</i> : Universal SAT Input Models	443
7.2 Nonlinear Unconstrained Method	445
7.3 Discrete Nonlinear Unconstrained Optimization	448
7.4 Continuous Nonlinear Unconstrained Optimization	449
7.5 Complete Nonlinear Unconstrained Optimization	449
7.6 Convergence Property and Average Time Complexity	451

8 Nonlinear Constrained Method	452
8.1 Continuous Lagrangian Search Algorithms	452
8.2 Discrete Lagrangian Search Algorithms	455
8.3 An Implementation of a Discrete Lagrangian Algorithm	456
9 Integer Programming Method	461
9.1 An Integer Programming Formulation for SAT	461
9.2 Linear Program Relaxation	462
9.3 Branch and Bound Method	463
9.4 Cutting-Plane Method	464
9.5 Interior Point Method	465
9.6 Improved Interior Point Method	466
10 Special Classes of SAT	467
10.1 2-SAT	468
10.2 Horn Formulas	469
10.3 Extended Horn Formulas	471
10.4 Formulas from Balanced $(0, \pm 1)$ Matrices	471
10.5 Single-Lookahead Unit Resolution	472
10.6 q-Horn Formulas	474
10.7 Renamable Formulas	475
10.8 Formula Hierarchies	476
10.9 Non-linear Formulations	476
11 Advanced Techniques	478
11.1 General Boolean Representations	478
11.2 Binary Decision Diagram (BDD)	479
11.3 Multispace Search	481
11.4 Parallel SAT Algorithms and Architectures	485
11.5 The <i>Multi-SAT</i> Algorithm	486
12 Probabilistic and Average-Case Analysis	488
12.1 The l -SAT Distribution	489
13 Performance Evaluation	505
14 Applications	521
15 Future Work	525
16 Conclusions	527
References	

1 Introduction

An instance of the satisfiability (SAT) problem is a Boolean formula that has three components [102, 191]:

- A set of n variables: x_1, x_2, \dots, x_n .
- A set of literals. A literal is a variable ($Q = x$) or a negation of a variable ($Q = \bar{x}$).
- A set of m distinct clauses: C_1, C_2, \dots, C_m . Each clause consists of only literals combined by just logical *or* (\vee) connectives.

The *goal* of the satisfiability problem is to determine whether there exists an assignment of truth values to variables that makes the following Conjunctive Normal Form (*CNF*) formula satisfiable:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_m, \quad (1)$$

where \wedge is a logical *and* connective.

The SAT problem is a *core* of a large family of computationally intractable NP-complete problems [102, 191]. Such NP-complete problems have been identified as central to a number of areas in computing theory and engineering. Since SAT is NP-complete, it is unlikely that any SAT algorithm has a fast worst-case time behavior. However, clever algorithms can rapidly solve many SAT formulas of practical interest. There has been great interest in designing efficient algorithms to solve most SAT formulas.

In practice, SAT is fundamental in solving many problems in automated reasoning, computer-aided design, computer-aided manufacturing, machine vision, database, robotics, integrated circuit design automation, computer architecture design, and computer network design (see Section 14). Therefore, methods to solve SAT formulas play an important role in the development of efficient computing systems.

Traditional methods treat a SAT formula as a discrete, constrained decision problem. In recent years, many optimization methods, parallel algorithms, and practical techniques have been developed. In this survey we describe sequential and parallel SAT algorithms and compare the performance of major SAT algorithms including variable setting, resolution, local search, nonlinear unconstrained method, nonlinear constrained method, integer programming method, and practical SAT algorithms. At the end of

this survey, we give a collection of practical applications of the satisfiability problem.

In the next section, we describe the constraint satisfaction problem (CSP) and its close relationship to the SAT problem. Section 3 describes various formulations of SAT and gives a brief overview of the basic sequential and parallel SAT algorithms, and discusses various categories of algorithms and performance evaluation methods. In Section 4, some SAT problem-instance models are given. Section 5 describes the variable setting and resolution procedures for solving SAT formulas. Local search algorithms, nonlinear unconstrained methods, nonlinear constrained techniques, and integer programming approaches for solving SAT formulas are discussed, respectively, in Sections 6, 7, 8, and 9. Section 10 discusses special subclasses of the SAT problem. Some advanced techniques for solving SAT formulas are described in Section 11. Section 12 gives probabilistic and average-case analysis of the SAT problem.

2 Constraint Satisfaction Problems

A constraint satisfaction problem (CSP) is to determine whether a set of constraints over discrete variables can be satisfied. Each constraint must have a form that is easy to evaluate, so any difficulty in solving such a problem comes from the interaction between the constraints and the need to find a setting for the variables that simultaneously satisfies all the constraints [443]. In a SAT formula, each constraint is expressed as a clause, making SAT a special case of the constraint satisfaction problem (see Figure 1). Due to this close relationship, any CSP algorithm can be transformed into a SAT algorithm, and this can usually be done in a way that maintains the efficiency of the algorithm.

A discrete CSP model consists of the following three components [211, 234]:

- *n variables*: $\{x_1, x_2, \dots, x_n\}$. An assignment is a tuple of n values assigned to the n variables.
- *n domains*: $\{D_1, D_2, \dots, D_n\}$. Domain D_i contains d possible *values* (also called *labels*) that x_i may be instantiated, i.e., $D_i = \{l_{i,1}, l_{i,2}, \dots, l_{i,d}\}$.
- A subset of $D_1 \times D_2 \times \dots \times D_n$ is a set of *constraints*. A set of *order-l constraints* ($l \leq n$) imposed on a subset of variables $\{x_{i_1}, x_{i_2}, \dots, x_{i_l}\} \subseteq$

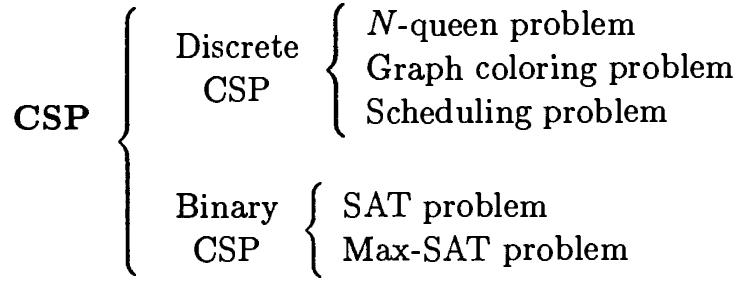


Figure 1: Some examples of the constraint satisfaction problem (CSP). SAT problem is a special case of CSP, i.e., a CSP with binary values.

$\{x_1, x_2, \dots, x_n\}$ is denoted as

$$C_{i_1, i_2, \dots, i_l} \subseteq D_{i_1} \times D_{i_2} \times \dots \times D_{i_l}.$$

An order- l constraint indicates the compatibility (*i.e.*, consistency/inconsistency or conflicting measure) among l variables for a given variable assignment. The variables *conflict* if their values do not satisfy the constraint. In practice, two frequently used constraints are *unary constraints* imposed on a single variable ($C_i \subseteq D_i$) and *binary constraints* imposed on a pair of variables ($C_{i,j} \subseteq D_i \times D_j$).

Solving a CSP entails minimizing local inconsistency and finding a consistent value assignment (*i.e.*, a consistent labeling) to the variables subject to the given constraints.

Constraint satisfaction problems are extremely common. Most NP-complete problems are initially stated as constraint satisfaction problems. Indeed, the proof that a problem is NP-complete implies an efficient way to transform the problem into a constraint satisfaction problem. For a few special forms of the constraint satisfaction problem there exist algorithms that solve such formulas in polynomial worst-case time. When no polynomial-time algorithm is known for a particular form of constraint satisfaction problem, it is common practice to solve such formulas with a search algorithm.

Problems that are commonly formulated as constraint satisfaction or satisfiability problems for the purposes of benchmarking include graph coloring and the n -queens problems. In the case of the n -queens problem, although analytical solutions for this problem exist [2, 10, 31], they provide a restricted subset of solutions. In practical applications, one must use a

search algorithm to find a *general* solution to the CSP or SAT problems.

3 Formulations and Algorithm Classes

In this section, we first describe various formulations of SAT, then give a brief overview of the basic sequential and parallel SAT algorithms, and discuss various categories of algorithms and performance evaluation methods.

3.1 Preliminaries

To simplify our discussion, throughout this chapter, let:

- \mathcal{F} be a *CNF* Boolean formula,
- N be the number of unsatisfiable clauses in \mathcal{F} ,
- m be the number of clauses in \mathcal{F} ,
- n be the number of variables in \mathcal{F} ,
- C_i be the i th clause,
- $|C_i|$ be the number of literals in clause C_i ,
- $Q_{i,j}$ be the j th literal in the i th clause, and
- l be the average number of literals: $\frac{\sum_{i=1}^m |C_i|}{m}$,

where $i = 1, \dots, m$ and $j = 1, \dots, n$.

On Boolean space $\{0, 1\}^n$, let:

- $F(\mathbf{x})$ be a function from $\{0, 1\}^n$ to integer N ,
- x_j be the j th variable,
- \mathbf{x} be a vector of n variables,
- $C_i(\mathbf{x})$ be the i th clause function, and
- $Q_{i,j}(\mathbf{x})$ be the j th literal function of the i th clause function,

where $i = 1, \dots, m$ and $j = 1, \dots, n$.

On real space E^n , let:

- $N(\mathbf{x})$ be a real function from $\{0, 1\}^n$ to E ,
- $f(\mathbf{y})$ be a real function from E^n to E ,
- y_j be the j th variable,
- \mathbf{y} be a vector of n variables,
- $c_i(\mathbf{y})$ be the i th clause function, and
- $q_{i,j}(\mathbf{y})$ be the j th literal function of the i th clause function,

where $i = 1, \dots, m$ and $j = 1, \dots, n$.

On real space E^n , also let:

- w_j be the j th integer variable, and
- \mathbf{w} be a vector of n integer variables,

where $i = 1, \dots, m$ and $j = 1, \dots, n$.

Following [367], a real-valued function f defined on a subset of E^n is said to be *continuous* at \mathbf{y} if $f(\mathbf{y}_k) \rightarrow f(\mathbf{y})$. A set of real-valued functions f_1, f_2, \dots, f_m on E^n form a vector function $\mathbf{f} = (f_1, f_2, \dots, f_m)$ whose i th component is f_i . It is *continuous* if each of its component functions is continuous.

If f has second partial derivatives which are continuous on this set, we define the *Hessian* of f at \mathbf{y} to be the $n \times n$ matrix denoted by

$$\mathbf{H}(\mathbf{y}) = \nabla^2 f(\mathbf{y}) = \left[\frac{\partial^2 f(\mathbf{y})}{\partial y_i \partial y_j} \right]. \quad (2)$$

We call $\mathbf{y} \in E^n$ with $f(\mathbf{y}) = 0$ a solution of f , denoted as y^* .

Two aspects of iterative optimization algorithms are their global convergence and local convergence rates [367]. *Global convergence* concerns, starting from an initial point, whether the sequence of points will converge to the final solution point. *Local convergence rate* is the rate at which the generated sequence of points converge to the solution.

3.2 Formulations of SAT

SAT problem can be expressed by Conjunctive Normal Form (*CNF*) formulas (e.g., $(x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2)$) or Disjunctive Normal Form (*DNF*) formulas (e.g., $(\bar{x}_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$). Instances of SAT can be formulated based on discrete or continuous variables [548, 550].

Discrete Formulations. These can be classified as unconstrained versus constrained.

(a) *Discrete Constrained Feasibility Formulations*. The goal is to satisfy all constraints. One possible formulation is the *CNF* formulas given by (1). A second formulation is the *DNF* formulas [212] discussed in Section 6.9.

(b) *Discrete Unconstrained Formulations*. A common formulation for *CNF* formulas exists [214, 217, 482]. The goal is to minimize $N(\mathbf{x})$, the number of unsatisfied clauses, under the interpretation that numeric variable

$x_i = 1$ ($x_i = 0$) if Boolean variable $x_i = \text{true}$ ($x_i = \text{false}$), respectively. That is,

$$\min_{\mathbf{x} \in \{0,1\}^n} N(\mathbf{x}) = \sum_{i=1}^m C_i(\mathbf{x}) \quad (3)$$

where

$$C_i(\mathbf{x}) = \prod_{j=1}^n Q_{i,j}(x_j) \quad (4)$$

$$Q_{i,j}(x_j) = \begin{cases} 1 - x_j & \text{if } x_j \text{ in } C_i \\ x_j & \text{if } \bar{x}_j \text{ in } C_i \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

In this case, $N(\mathbf{x}) = 0$ when all the clauses are satisfied.

A similar formulation for DNF formulas exists (See Section 6.9, [242], [212], [2]. Under the interpretation that numeric variable $x_i = 1$ ($x_i = 0$) if Boolean variable $x_i = \text{true}$ ($x_i = \text{false}$), respectively, the goal is to solve

$$\min_{\mathbf{x} \in \{0,1\}^n} F(\mathbf{x}) = \sum_{i=1}^m C_i(\mathbf{x}), \quad (6)$$

where

$$C_i(\mathbf{x}) = 1 - \prod_{j=1}^n Q_{i,j}(x_j) \quad (7)$$

$$Q_{i,j}(x_j) = \begin{cases} x_j & \text{if } x_j \text{ in } C_i \\ 1 - x_j & \text{if } \bar{x}_j \text{ in } C_i \\ 1 & \text{otherwise} \end{cases} \quad (8)$$

All the clauses are satisfied when $F(\mathbf{x}) = 0$.

Alternatively, DNF formulas can be solved as follows:

$$\max_{\mathbf{x} \in \{0,1\}^n} F(\mathbf{x}) = \sum_{i=1}^m C_i(\mathbf{x}), \quad (9)$$

where

$$C_i(\mathbf{x}) = \prod_{j=1}^n Q_{i,j}(x_j), \quad (10)$$

and $Q_{i,j}(x_j)$ is given by (8).

Usually, the question of falsifiability for a *DNF* formula is more interesting than the question of satisfiability. This can be solved as follows:

$$\min_{\mathbf{x} \in \{0,1\}^n} F(\mathbf{x}) = \sum_{i=1}^m C_i(\mathbf{x}), \quad (11)$$

where $C_i(\mathbf{x})$ is given by (10). A formula is falsifiable if $F(\mathbf{x}) = 0$ for some \mathbf{x} .

(c) *Discrete Constrained Formulations.* There are various forms of this formulation. One approach is to formulate SAT formulas as instances of the 0-1 integer linear programming (ILP) problem.

Another approach is to minimize the objective function $N(\mathbf{x})$, the number of unsatisfiable clauses, subject to a set of constraints, as follows [548, 550]:

$$\begin{aligned} \min_{\mathbf{x} \in \{0,1\}^n} \quad & N(\mathbf{x}) = \sum_{i=1}^m C_i(\mathbf{x}) \\ \text{subject to} \quad & C_i(\mathbf{x}) = 0 \quad \forall i \in \{1, 2, \dots, m\}. \end{aligned} \quad (12)$$

A formulation based on *DNF* can be defined similarly.

This formulation uses additional constraints to guide the search. The violated constraints provide another mechanism to bring the search out of a local minimum. This formulation is used in a Lagrange multiplier-based method to solve the SAT problem (see Section 8.2 and [548, 550]).

Continuous Formulations. In formulating a discrete instance of SAT in continuous space, we transform discrete variables in the original formula into continuous variables in such a way that solutions to the continuous problem are binary solutions to the original formula. Such a transformation is potentially beneficial because an objective in continuous space may “smooth out” some infeasible solutions, leading to a smaller number of local minima explored. In the following, we show two such formulations.

(a) *Continuous Unconstrained Formulations.* There are many possible formulations in this category. A simple formulation, *UniSAT* (Universal SAT Problem Models) [212, 214, 216, 215], suggests:

$$\min_{\mathbf{y} \in E^n} f(\mathbf{y}) = \sum_{i=1}^m c_i(\mathbf{y}), \quad (13)$$

where

$$c_i(\mathbf{y}) = \prod_{j=1}^n q_{i,j}(y_j) \quad (14)$$

$$q_{i,j}(y_j) = \begin{cases} |y_j - T| & \text{if } x_j \text{ in } C_i \\ |y_j + F| & \text{if } \bar{x}_j \text{ in } C_i \\ 1 & \text{otherwise} \end{cases} \quad (15)$$

where T and F are positive constants.

Two special formulations to (15) exist. In the *UniSAT5* model [216, 218]

$$q_{i,j}(y_j) = \begin{cases} |y_j - 1| & \text{if } x_j \text{ is in } C_i \\ |y_j + 1| & \text{if } \bar{x}_j \text{ is in } C_i \\ 1 & \text{otherwise} \end{cases} \quad (16)$$

and in the *UniSAT7* model [216, 218]:

$$q_{i,j}(y_j) = \begin{cases} (y_j - 1)^2 & \text{if } x_j \text{ is in } C_i \\ (y_j + 1)^2 & \text{if } \bar{x}_j \text{ is in } C_i \\ 1 & \text{otherwise} \end{cases} \quad (17)$$

Values of \mathbf{y} that make $f(\mathbf{y}) = 0$ are solutions to the original formula in (1). *UniSAT5* can be solved with efficient, discrete, greedy local search algorithms (Section 7 and [218]). *UniSAT7* requires computationally expensive continuous optimization algorithms, rendering them applicable to only small formulas (Section 7 and [218, 223]).

(b) *Continuous Constrained Formulations*. This generally involves a heuristic objective function that measures the quality of the solution obtained (such as the number of clauses satisfied). One formulation similar to (13) is as follows.

$$\begin{aligned} \min_{\mathbf{y} \in E^n} \quad & f(\mathbf{y}) = \sum_{i=1}^m c_i(\mathbf{y}) \\ \text{subject to} \quad & c_i(\mathbf{y}) = 0 \quad \forall i \in \{1, 2, \dots, m\} \end{aligned} \quad (18)$$

where $c_i(\mathbf{y})$ is defined in (14).

The key in this approach lies in the transformation. When it does **not** smooth out local minima in the discrete space or when solution density is low, continuous methods are much more computationally expensive to apply than discrete methods.

Since (18) is a continuous constrained optimization problem with a non-linear objective function and nonlinear constraints, we can apply existing Lagrange-multiplier methods to solve it. Our experience is that a Lagrangian transformation does not reduce the number of local minima, and continuous Lagrangian methods are at least an order-of-magnitude more expensive to apply than the corresponding discrete algorithms [81].

3.3 Basic SAT Algorithm Classes

The existing SAT algorithms can be grouped into the following several classes. Most existing SAT algorithms can be grouped into these categories.

- *Discrete, constrained algorithms.* Algorithms in this category treat a SAT formula as an instance of a constrained decision problem, applying discrete search and inference procedures to determine a solution. One straightforward way to solve an instance of SAT is to enumerate all possible truth assignments and check to see if one satisfies the formula. Many improved techniques, such as consistency algorithms [234, 369], backtracking algorithms [35, 54, 65, 336, 435], term-rewriting [131, 275], production system [492], multi-valued logic [488], Binary Decision Diagrams [60, 17], chip and conquer [195], resolution and regular resolution [198, 365, 407, 459, 524, 535, 559], independent set algorithm [288], matrix inequality system [522], and discrete Lagrangian algorithm [548], have been proposed.

Many of the discrete constrained algorithms eliminate one variable at a time. This can be done either by making repeated use of resolution, as was done in the original version of the Davis-Putnam (DP) procedure [119], or by assigning some variable each possible value and generating a sub-formula for each value, as was done in Loveland's modification to the DP procedure [118, 365]. Resolution generates only one new formula, but in the worst case the number of clauses in that new formula will be proportional to the square of the number of clauses in the original formula. Assigning values to a variable (often called searching) generates two new formulas. For random formulas, resolution methods are fast when the number of clauses is small compared to the number of values [167, 87], while search methods are fast except when the number of clauses is such that the expected number of solutions is near one [443]. The two approaches can be combined, using resolution on some variables and search on others.

D i s c r e t e	Constrained	1960: Davis-Putnam (DP) algorithm [119] 1965: Resolution [459] 1971: Consistency algorithms [96, 211, 278, 551] 1978: Loveland's Davis-Putnam (DPL) [118, 365] 1986: Parallel consistency chips [211, 232, 233] 1986: Binary decision diagrams (BDD) [17, 60] 1988: Chip and conquer [195] 1990: DPL plus heuristic (DPLH) [291] 1989: Local search & backtracking [217] 1993: Backtracking and probing [443] 1994: Parallel DP algorithm [39] 1994: Matrix inequality system [522] 1996: CSAT [152] 1996: Lagrangian algorithm [548]
	Unconstrained	1987: Randomized Local search (SAT1) [212, 217] 1987: Parallel local search (SAT1.6) [212, 217] 1988: Local search for n -queen [211, 494, 495] 1990: <i>Unison</i> algorithm and hardware [502, 503] 1991: Local search complexity [226, 413] 1991: Local search for 2-SAT [412] 1992: Local search with traps (SAT1.5) [214, 217] 1992: Greedy local search – GSAT [482]
C o n t i n u o u s	Constrained	1986: Branch-and-bound (APEX) [36] 1988: Programming models [36, 292] 1988: Cutting plane [266, 264] 1989: Branch-and-cut [267] 1989: Interior point method [310, 308]
	Unconstrained	1987: <i>UniSAT</i> models [212, 216, 218] 1987: Nonlinear optimization (SAT6) [212, 218] 1989: Neural net models [298, 76] 1990: Nonlinear optimization & backtracking [21] 1991: SAT14 algorithms [218]

Figure 2: Some typical algorithms for the SAT problem.

Other specific algorithms using these principles include simplified DP algorithms [184, 206, 440], and a simplified DP algorithm with strict ordering of variables [276]. The DP algorithm improved in certain aspects over Gilmore's proof method [200]. Analyses of SAT algorithms often concentrates on algorithms that are simple because it is difficult to do a correct analysis of the best algorithms. Under those conditions where simple algorithms are fast, related practical algorithms are also fast. (It is difficult to tell whether a practical algorithm is slow under conditions that make the corresponding simplified algorithm slow.)

A number of special SAT problems, such as 2-satisfiability and Horn clauses, are *solvable* in polynomial time [5, 102, 407]. There are several linear time algorithms [18, 156] and polynomial time algorithms [412, 472] existing.

- *Discrete, unconstrained algorithms.* In this approach, the number of unsatisfiable *CNF* (or satisfiable *DNF*) clauses is formulated as the value of the objective function, transforming the SAT formula into a discrete, unconstrained minimization problem to the objective function. Local search is a major class of discrete, unconstrained search methods [214, 217, 234, 226, 413, 482]. It can be used to solve the transformed formula (see Section 6).

Early work in constraint satisfaction and complexity study contributed to the development of local search algorithms for the SAT problem (see Sections 6.2, 6.4, and Figure 3). There were two major approaches in this area: randomized local search (*SAT1*) and greedy local search (*GSAT*). The *SAT1* algorithm was the first local search algorithm developed from the VLSI engineering and scheduling applications. The *GSAT* algorithm was derived from the early local search algorithms for the n -queen problem.

- *Constrained programming algorithms.* Methods in this class were developed based on the fact that *CNF* or *DNF* formulas can be transformed to instances of Integer Programming, and possibly solved using Linear Programming relaxations [36, 265, 266, 292, 310, 308, 411, 558]. Many approaches, including branch-and-bound [36], cutting-plane [266, 264], branch-and-cut [267], interior-point [310, 308], and improved interior-point [489], have been proposed to solve the integer program representing the inference problem. Researchers found integer programming methods faster than resolution for certain classes

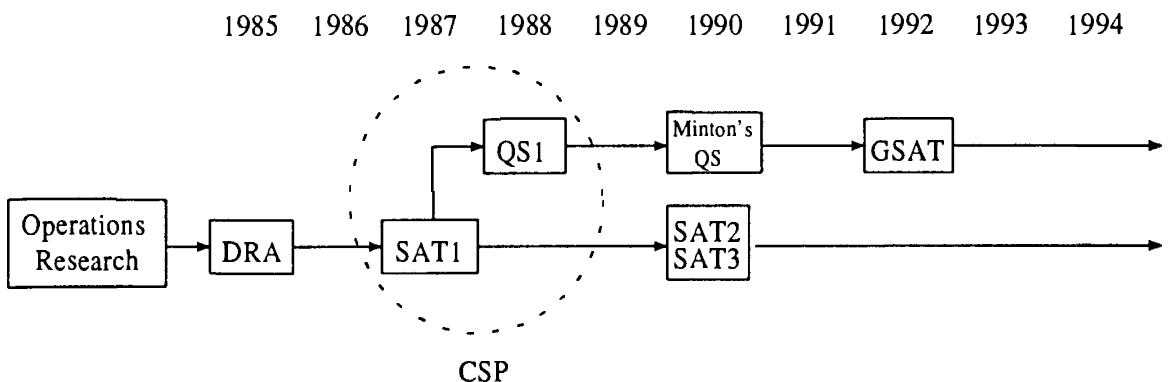


Figure 3: Early development of local search algorithms for SAT problem. There were two major approaches: randomized local search (*SAT1*) and greedy local search (*GSAT*). The *SAT1* was the first local search algorithm developed for the VLSI engineering and scheduling applications. The *GSAT* algorithm was derived from the early local search algorithms for the n -queen problem.

of problems, although these methods do not possess a robust convergence property and often fail to solve hard instances of satisfiability [36, 265, 266, 292, 310, 308].

- *Unconstrained, nonlinear optimization algorithms.* Special models have been formulated to transform a discrete formula on Boolean space $\{0, 1\}^n$ (a decision problem) into an unconstrained *UniSAT* problem on real space E^n (an unconstrained nonlinear optimization problem). The transformed formulas can be solved by many existing nonlinear optimization methods [212, 214, 216, 218, 234] (see Section 7).

3.4 Parallel SAT Algorithms

In practice, most sequential SAT algorithms can be mapped onto parallel computer systems, resulting in parallel SAT algorithms [220]. Accordingly, as given in Figure 4, there are *four* classes of parallel algorithms for solving SAT.

- *Parallel, discrete, constrained algorithms.* Many discrete, constrained SAT and CSP algorithms have been implemented in parallel algorithms or put on special-purpose, hardware VLSI architectures. These include parallel consistent labeling algorithms [527, 381], parallel discrete relaxation (DRA) chips [232, 211, 233], parallel arc consistency

D i s c r e t e	Constrained	<ul style="list-style-type: none"> 1983: Parallel CLP algorithms [527, 381] 1986: Parallel DRA chips [211, 232, 233] 1987: Parallel DP algorithm [88] 1988: Parallel AC algorithms [471] 1988: Parallel CSP architectures [211, 233] 1990: <i>Unison</i> algorithm and hardware [502, 503] 1992: Vectorized DP algorithm [158] 1994: MIMD DP algorithm [39]
	Unconstrained	<ul style="list-style-type: none"> 1987: <i>CNF</i> local search [212, 217] 1987: <i>DNF</i> local search [212, 218] 1987: Parallel local search [212, 217] 1991: Discrete $\alpha\beta$ relaxation [213] 1993: Multiprocessor local search [500, 499]
C o n t i n u o u s	Constrained	<ul style="list-style-type: none"> 1989: Interior point method [310, 308]
	Unconstrained	<ul style="list-style-type: none"> 1987: <i>UniSAT</i> models [212, 218] 1987: Nonlinear optimization (SAT6) [212, 21] 1991: Continuous $\alpha\beta$ relaxation [213] 1991: SAT14 algorithms [218] 1991: Parallel nonlinear optimization [216, 21] 1992: Neurocomputing [220]

Figure 4: Some parallel SAT/CSP algorithms.

(PAC) algorithms [471], parallel constrained search architectures [211, 233], parallel *Unison* algorithms [502], parallel *Unison* architectures [503], parallel DP algorithms [39, 88, 158], and parallel logical programming languages [100, 361, 542, 543, 544]. A speedup greater than the number of processors sometimes occurs because of correlations among variable settings that lead to solutions [396, 348].

- *Parallel, discrete, unconstrained algorithms.* A number of discrete local optimization algorithms were implemented on parallel computing machines. These include *CNF* local search [212, 217], *DNF* local search [212, 218], parallel local search [212, 217], and multiprocessor local search [500, 499]. A new $\alpha\beta$ relaxation technique was developed in a parallel and distributed environment [213].
- *Parallel, constrained programming algorithms.* Kamath *et al.* implemented an interior point zero-one integer programming algorithm on a *KORBX(R)* parallel/vector computer [310, 308].
- *Parallel, unconstrained, nonlinear optimization algorithms.* Several of these algorithms have been implemented: *UniSAT* models [212, 218], parallel, continuous $\alpha\beta$ relaxation [213], and parallel nonlinear optimization algorithms [216, 218].

Computer architectures affect the data structures, implementation details, and thus the performance of SAT algorithms. Most early studies of CSP/SAT algorithms were performed on sequential computers. Some recent work has been concentrated on parallel programming on multiprocessors. McCall *et al.* [381, 527] simulated an 8-processor architecture with various system, topology and performance criteria for the forward checking CSP algorithm. Samal implemented several parallel AC algorithms on a CRAY computer [470] and an 18-node BBN Butterfly Plus MIMD, shared-memory, homogeneous parallel processor [469]. Cooper and Swain implemented parallel AC algorithms on a Connection Machine [104]. Kamath and Karmarkar *et al.* implemented an interior point zero-one integer programming algorithm for SAT on a *KORBX(R)* parallel/vector computer [310, 308]. Recently, Fang and Chen implemented a vectorized DP algorithm on an *ETA10Q* vector computer [158]. Speckenmeyer and Böhm have experimented with the parallelization of variants of the Davis-Putnam-Loveland (DPL) procedure on a message based MIMD Transputer system built with 320 (INMOS T800/4MB) processors [39]. In their implementation, for some small k , each

of 2^k processors solves a formula arising at depth k of a DPL search tree, and computation ceases as soon as one processor reports that its formula is satisfiable. Speckenmeyer noticed that the time to completion was usually less than $N/2^k$ where N is the time taken by the serial version [39].

Research works continue by building special-purpose VLSI architectures to speed up SAT/CSP computations. For an n -variable and d -value instance of CSP, Wang and Gu [553, 554] gave an $O(n^2d^2)$ time parallel DRA2 algorithm and an SIMD DRA2 architecture. Furthermore, Gu and Wang [232] gave an $O(n^2d)$ time parallel DRA3 algorithm and a dynamic DRA3 architecture for solving general DRA problems. Later, Gu and Wang [211, 233] developed an $O(nd)$ time, massively parallel DRA5 algorithm and a parallel DRA5 VLSI architecture. For problems of practical interest, parallel DRA algorithms running on special-purpose VLSI architectures offer many orders of magnitude in performance improvement over sequential algorithms.

Recently, Sosić, Gu, and Johnson have developed a number of parallel algorithms and architectures for differential, non-clausal inference of SAT formulas [502, 503].

3.5 Algorithm Categories

Some SAT algorithms are *complete* (they definitely determine whether an input has a solution or does not have one) [60, 119, 118, 365, 459], while others are *incomplete* (they sometimes determine whether or not the input has a solution, but in other cases they cannot find one) [217, 218, 308, 412].

Most incomplete algorithms find one solution (or perhaps several solutions) in *favorable cases*, but give up or do not terminate in other cases. In such cases one does not know whether the input has no solution or the algorithm did not search hard enough. Some incomplete algorithms can verify that a formula has no solution but can not find one if at least one solution exists. Such is the case for incomplete algorithms that check for patterns that imply unsatisfiability. Incomplete algorithms, however, are of particular interest for inputs that are so difficult that a complete algorithm cannot solve them in reasonable time.

Complete algorithms can perform one of the following actions: (1) determine whether or not a solution exists, (2) give the variable settings for one solution, (3) find all solutions or an optimal solution, (4) prove that there is no solution. Algorithms of the first type would be of theoretical interest only were it not for the fact that any such algorithm can be modified, with little loss of efficiency, to give an algorithm of the second type. Algorithms

of the third type are needed when there is some measure of the solution quality, and the optimal solution is sought or when the overall problem has constraints in addition to those of the SAT instance. The algorithms are essential to many important practical applications that are NP-hard in nature.

Requiring a program to produce each solution in explicit form ensures that the worst-case time will be exponential whether or not $P = NP$ (because some inputs have an exponential number of solutions). An alternative is to give the solutions in some compressed form. For example, some algorithms implicitly list all solutions by giving *cylinders of solutions*, *i.e.*, the settings of some variables with the understanding that the remaining variables are *don't cares* which can have any value. For some formulas, using this approach to represent all solutions is much more compact than an explicit representation [65, 385]. Binary Decision Diagrams (BDD) are a more sophisticated and compact way to represent the set of all solutions [60, 17]. Some instances of SAT, however, have a structure such that it is faster to generate the solution to various subsets of the constraints (depending on a subset of the variables) and then test whether those various solution sets have anything in common rather than try to solve the entire formula at once. This type of SAT algorithm shows greater efficiency improvements for certain practical engineering design problems [449].

The techniques used in complete SAT algorithms can usually be adapted to provide *exact solutions* to optimization problems. The techniques used in incomplete SAT algorithms can usually be adapted to provide *approximate solutions* to optimizations problems. They normally lead to algorithms that produce low (but not necessarily the lowest) value of the function.

For random sets of formulas, the probability that a particular formula has at least one solution is perhaps the most important parameter for determining how difficult the set will be for a particular algorithm. The best known algorithms have difficulty when the probability is near 0.5, but are fast when the probability is close to 0 or 1. We use formulas generated from the 3-SAT model as an example. Figure 5 shows, for 50 variables, the real execution results of a Davis-Putnam (DP) algorithm for 100 to 500 clauses. The computing time used by a program for the DP algorithm ([119, 118, 365]) is shown for the 3-SAT model (solid line) and the average 3-SAT model (dotted line) [212, 217, 392, 110]. Random formulas generated in the left region are usually satisfiable, and the procedure is fast. Random formulas in the right region are usually unsatisfiable, and the procedure is fast. For random formulas in the middle, many are satisfiable and many are

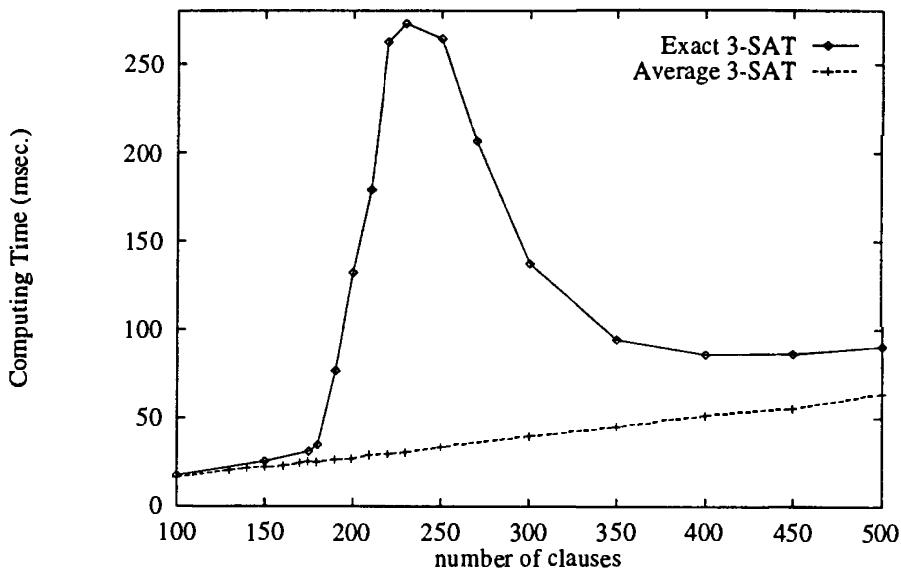


Figure 5: Computing time of a Davis-Putnam (DP) algorithm for the exact and the average 3-SAT models (with 50 variables) on a SUN SPARC 1 workstation. The horizontal axis is measured by m or m/n .

unsatisfiable; the procedure is slow. Because the DP algorithm is a complete algorithm, it is able to verify satisfiability and unsatisfiability. So it gives results for random formulas in all three regions.

The results of the DP algorithm may not hold for a different SAT algorithm. A local search may often find a solution for a satisfiable *CNF* much more quickly than the DP algorithm but does not always verify satisfiability and cannot prove unsatisfiability. In particular, it gives no answer if a *CNF* formula is not satisfiable. Thus, for most formulas in the peak region and nearly all formulas in the right region, a local search algorithm will not terminate in a reasonable amount of time.

3.6 Performance Evaluation

The performance of an algorithm can be determined *experimentally* or *analytically*. It is feasible to do experimental studies with typical or random formulas, but not with worst-case formulas (there are too many formulas of a given size to experimentally determine which one leads to the worst-case time). It is feasible to do analytical studies with random or worst-case formulas but not with typical formulas (typical sets of formulas seldom have a

mathematical structure suitable for analysis).

Experimental studies are sometimes inconclusive because they consider a relatively small number of input possibilities. Such restrictions are often forced because the space of likely input formulas, and even the size of such formulas, is so large. Analytical studies are intended to determine performance over broad families of inputs where each family typically represents a class of formulas of a particular size. However, such studies have the drawback that only the simplest of algorithms can be analyzed. To compensate for this, several features of a complex algorithm can be removed, leaving a rather simple, more analyzable one. The simplified algorithm usually contains one or two simple techniques, such as the unit-clause-rule, or the pure-literal-rule. An analytical result on the simplified algorithm provides a bound on the performance of the complex algorithm, and this bound is sometimes sufficient to understand the behavior of the complex algorithm. Such an approach has the following side benefit: analytical studies can suggest which simple techniques should be included in practical algorithms. In fact, most of the 6 prize winners of the 1991 SAT contest were associated with analytical studies of SAT algorithms [70, 71]. The two top winners were associated with both experimental and analytical studies of SAT algorithms. The analytical studies of SAT algorithms involve the following.

1. Worst-Case Studies.

Unless $P = NP$, all SAT algorithms have a worst-case time that is superpolynomial in the input size [102]. A number of studies have concentrated on the worst-case analysis of variable setting algorithms for solving SAT [195, 395, 327].

2. Probabilistic Studies.

Since the typical performance of many satisfiability algorithms is much better than any proven worst-case results, there is considerable interest in evaluating the probabilistic performance of these algorithms. Such studies use some model for generating random formulas and then calculate the performance of algorithms on these formulas. The two most widely used measures of performance are average time and probabilistic time.

Average time is a weighted average of the time (or some related measure, such as the number of nodes) to solve a given sample of formulas. An algorithm must solve each formula for the average to be defined. In *probabilistic time*, the time to solve a formula is determined by the probability of the formula being satisfiable.

bilistic time studies, an algorithm is given a deadline (usually specified as a polynomial in the length of input formulas), and one studies the fraction of formulas that are solved within the deadline. Probabilistic time studies can be performed on algorithms which give up on some fraction of the formulas so long as that fraction is less than the goal fraction.

For incomplete algorithms, the average time is not defined so only the fraction of inputs solved can be studied. One can also use various hybrid measures, such as the average time used to solve the easiest 90 percent of the inputs.

The literature contains a number of studies of the average time and probabilistic time performance of certain SAT algorithms [54, 169, 427, 168, 204, 205, 206, 276, 434, 442, 437]. Despite the worst-case complexity of SAT, algorithms and heuristics with polynomial average time complexities have been reported [83, 84, 166, 435, 439, 440, 441, 562]. This subject is treated in more detail in Section 12.

3. Number of Solutions.

Some researchers investigated the number of solutions of random SAT formulas. Extending Iwama's work [288] Dubois gave a combinatorial formula computing the number of solutions of any set of clauses [149]. Dubois and Carlier also studied the mathematical expectation of the number of solutions for a probabilistic model [150].

During the past two decades, many performance studies were performed through sampling techniques [320, 434, 511],¹ experimental simulations [55, 194, 248], analytical studies [54, 178, 179, 442, 435, 439, 510], as well as the combined effort of the above approaches [194, 273, 320, 434, 510].

4 SAT Input Models

In this section, we describe several basic SAT input models and their characteristics.

¹Knuth [320] first showed how to measure the size of a backtrack tree by repeatedly following random paths from the root. Purdom [434] gave a modified version of Knuth's algorithm which greatly increases the efficiency of Knuth's method by occasionally following more than one path from a node. Stone and Stone [511] presented a variant of the algorithms of Knuth and Purdom for estimating the size of the unvisited portion from the statistics of the visited portion.

4.1 Random Input Models

The running time of a SAT algorithm depends on the type of input being solved. The following SAT input models are often used to generate a variety of input types.

- **Hardest formulas.** Generate that formula that is the most difficult for the algorithm being measured. This approach is often used for analytical studies. There are too many possible formulas to use this approach in experimental studies. Bugrara and Brown [66] reported the effects these minor variations have on the average time needed by the simple backtracking algorithm.

Experimental studies sometimes include results for the hardest formulas from the set of formulas tested, but such results are quite different from what the results would be if the entire set of possible formulas had been tested.

Most analytical studies use the following two basic models to generate random *CNF* formulas. Each model has several variations depending on whether identical clauses are permitted, whether a variable and its negation can occur in a clause, etc.

- **The l -SAT model.** In the l -SAT model, a randomly generated *CNF* formula consists of m independently generated random clauses. Each clause is chosen uniformly from the set of all possible clauses of exactly l literals that can be composed from a variable set $X = \{x_1, \dots, x_n\}$ such that no two literals are equal or complementary. The number of possible clauses is $2^l \binom{n}{l}$. This model is sometimes called the *fixed-clause-length* model. Similar models were used in [54, 83, 84, 169, 427, 217, 218, 392, 385, 439].
- **The average l -SAT model.** In the average l -SAT model, a randomly generated *CNF* formula consists of m independently generated random clauses. In each clause, each of n variables occurs positively with probability $p(1 - p)$, negatively with probability $p(1 - p)$, both positively and negatively with probability p^2 , not at all with probability $(1 - p)^2$, where p can be a function of m and n . The average number of literals in a clause is $l = 2pn$. This model is also called the *random-clause-length* model. This model and variations were used in [166, 167, 206, 214, 217, 218, 266, 267, 310, 308, 489].

Most papers use just one model, but the performance of simple backtracking has been considered under a number of related models [66].

4.2 Hardness

Various SAT algorithms differ greatly in the amount of time they need to solve particular inputs. For example, Iwama's algorithm [288] is fast for random formulas with lots of solutions and slow for random formulas with few solutions, while simple backtracking [441] is fast on formulas with few solutions and slow on formulas with many solutions. Therefore, the *hard-and-easy distributions* of SAT formulas depend not only on the inherent property of the SAT input models but also on the algorithms used to solve the formulas. Any particular SAT formula is easy for some algorithm (for example a table lookup algorithm with that formula in its lookup table). Thus, hardness is a *property* of large sets of formulas rather than individual formulas.

For sets of formulas generated by random models with parameters, the probability of finding a solution varies with the parameter settings. Those sets generated with parameters set in regions where solutions are going from unlikely to common are particularly difficult for all algorithms that have been studied (see Figure 6).

For random l -SAT formulas fewer literals and larger number of clauses reduce the possibility of making all clauses jointly satisfiable. Therefore the computing time for random l -SAT formulas increases, up to a point, when m/n increases or the number of literals l ($l > 3$) in each clause decreases (Figure 5). Inspection of Figure 5 reveals a “hump” of difficulty for l -SAT formulas where 50% of the sample space is satisfiable, but a “flat” increase in difficulty for random l -SAT formulas in a correspondingly similar region of the parameter space.

4.3 Comparison of Random Input Models

The structural properties of random formulas generated by the two input models given above can be quite different and this can have a significant impact on the performance of a complete SAT algorithm. This significance is felt especially in the region of the parameter space for which random formulas are nearly equally likely to be satisfiable or unsatisfiable. Figure 5 shows, for 50 variables, the actual computing time of a complete SAT (*SAT14.11*)

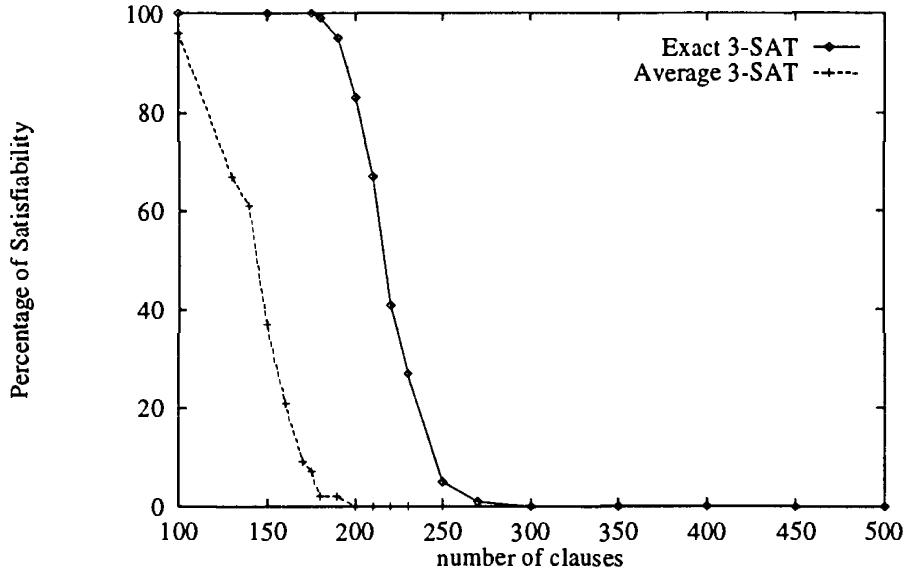


Figure 6: Percentage of satisfiability for formulas with 50 variables generated by the 3-SAT and the average 3-SAT input models, respectively. The horizontal axis is measured by m or m/n .

algorithm² for random formulas generated from the 3-SAT model and the average 3-SAT model [212, 217, 219, 218]. Figure 6 shows the percent of random formulas that are satisfiable as a function of formula size for both models. For a complete algorithm, the problem instances generated from the average 3-SAT model is much easier than those generated from the 3-SAT model. It takes a complete algorithm much less computing time to solve formulas generated from the average 3-SAT model.

For an incomplete algorithm such as local search, however, the situation is different. In Figure 6, for the same number of clauses (i.e., the same m/n values), problem instances generated from the average 3-SAT model have much lower percentage of satisfiability (compared to those generated from the 3-SAT model). The sat-and-unsat boundary of the average 3-SAT model is shifted to the left and is drawn by smaller m/n values than those for the 3-SAT problem model. For the same m/n values, more problem instances generated from the average 3-SAT model are unsatisfiable, making it harder for a local search algorithm to handle the average 3-SAT problem model.

²SAT14.11 is a backtracking algorithm combined with coordinate descent in the real space [218].

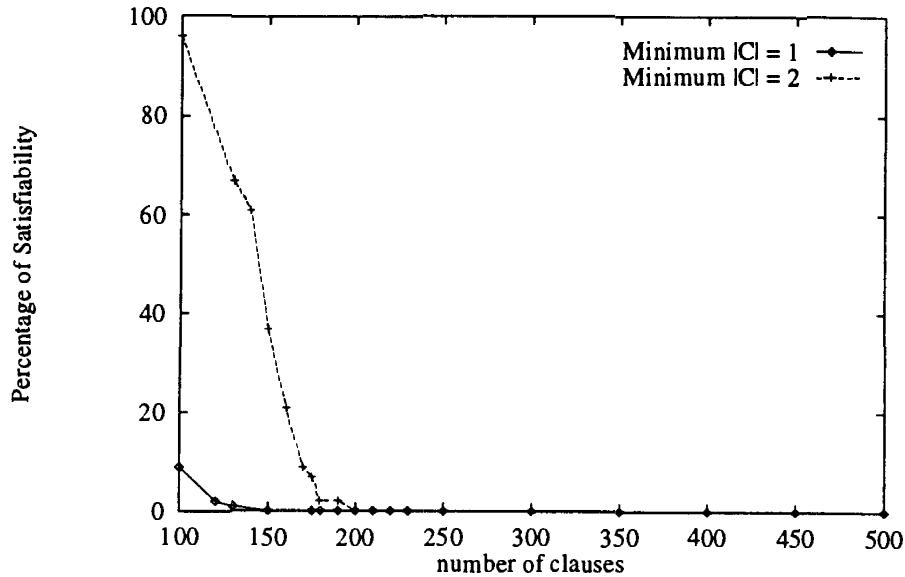


Figure 7: Percentage of satisfiability for two average 3-SAT problem models with $|C|_{min} = 1$ and $|C|_{min} = 2$ (50 variables), respectively. Problem instances generated with smaller length of the shortest clauses have much lower percentage of satisfiability.

Experimental results confirmed that it took a local search algorithm much longer time to solve problem instances generated from the average l -SAT models [214].

Many factors can affect the property of the random models significantly. For the same average 3-SAT problem model even a slight variation to the length of the shortest clause in a *CNF* formula would significantly shift the sat-and-unsat boundary. In Figure 7, the solid curve was generated from an average 3-SAT model. The length of the shortest clause in the model was 1. The dotted curve was generated from the same average 3-SAT model but the length of the shortest clause was set to 2. Clearly, shorter clauses enforce tighter constraints and generate much harder random instances for the same model.

Incomplete algorithms that fail on unsatisfiable inputs can be effective only in the half-planes $m/n < 2^l/l$ for the l -SAT model and $pn > \ln(m)$ for the average l -SAT model, where the probability that a random formula is satisfiable is high (see Section 12). Incomplete algorithms that fail on satisfiable inputs can be effective only in regions complementary to those

above.

Experience with the best complete algorithms has caused some to conclude the following:

1. Average l -SAT formulas are easy for the best algorithms;
2. l -SAT formulas are difficult even for the best algorithms; and
3. Formulas generated by both models are of similar difficulty when the average clause length is large.

Obviously, there are some conflicts in these beliefs.

4.4 Practical Input Models

Random input models such as those discussed above are suitable for analytical studies of SAT algorithms because they generate formulas which possess a symmetry that can be exploited for analysis. Actual formulas often have a different structure. Therefore, structured problem instances and practical SAT applications are essential to evaluate the performance of SAT algorithms. Examples of these are the following:

- **Regular SAT models.** Models derived from problems such as graph coloring and n -queens, are used to assess the performance of SAT algorithms [216, 218].
- **Practical applications problems.** Models derived from practical application domains, such as integrated circuit design, mobile communications, computer architecture and network design, computer-aided manufacturing, and real-time scheduling, have a variety of special characteristics (see Section 14).

Some experiments strongly suggest that there is little correlation between the performance of a SAT algorithm tested through random input models and the performance of the same algorithm tested through practical input models. Local search is faster for some random inputs but can be slower than a complete SAT algorithm for problems raised from practical applications. The boundary phenomenon discussed in random models is an artifact of some probabilistic models. It has not yet been observed in practical input models.

Practical applications are ultimately the *most important*, although it is difficult for people outside the area of application to understand how

important or difficult a particular application problem is. It is also difficult to develop a general theory on the speed of SAT algorithms on applications. Much research is, therefore, done on the more regular source of problems in the hope of better understanding the speed that SAT algorithms will have when applied to a wide range of practical applications.

5 Splitting and Resolution

Recursive replacement of a formula by one or more other formulas, the solution of which implies the solution of the original formula, is an effective paradigm for solving *CNF* formulas. Recursion continues until one or more primitive formulas have been generated and solved to determine the satisfiability of the original. One way to achieve this is through splitting.

In splitting, a variable v is selected from a formula, and the formula is replaced by one sub-formula for each of two possible truth assignments to v . Each sub-formula has all the clauses of the original except those satisfied by the assignment to v and otherwise all the literals of the original formula except those falsified by the assignment. Neither sub-formula contains v , and the original formula has a satisfying truth assignment if and only if either sub-formula has one. Splitting insures that a search for a solution terminates with a result.

Another effective paradigm is based on resolution [459]. In resolution, a variable v is selected and a resolvent (see below) obtained using v is added to the original formula. The process is repeated to exhaustion or until an empty clause is generated. The original formula is not satisfiable if and only if an empty clause is a resolvent. Although there is only one new formula on each step, the total number of steps (or resolvents) can be extremely large compared to the number of clauses in the original formula. Many algorithms that use resolution form all possible resolvents using a particular variable at one time. When this is done, the original clauses that contain the variable and its negation may be dropped. An algorithm may use both splitting and resolution.

Early examples of these approaches are the two forms of the Davis Putnam procedure. The original DP procedure used resolution [119] while the revised version, *i.e.*, the *Davis-Putnam-Loveland* (DPL) *procedure*, used splitting [118, 365]. Combining splitting with depth-first search in the DPL procedure avoids memory explosion that occurs on many inputs when they are solved by the original DP procedure.

Most recursive SAT algorithms use the following primitive conditions to stop the recursion:

1. formulas with an empty clause have no solution.
2. formulas with no clauses have a solution.
3. formulas with no variables (*i.e.*, all variables have been assigned values) are trivial.

The following subsections present various SAT algorithms, organized by the basic approach that each algorithm takes. Some of these algorithms are much simpler than you would want to use in practice but are of interest because it has been possible to analyze their running time for random formulas.

5.1 Resolution

Given two clauses $C_1 = (v \vee x_1 \vee \dots \vee x_{l_1})$ and $C_2 = (\bar{v} \vee y_1 \vee \dots \vee y_{l_2})$, where all x_i and y_j are distinct, the resolvent of C_1 and C_2 is the clause $(x_1 \vee \dots \vee x_{l_1} \vee y_1 \vee \dots \vee y_{l_2})$, that is, the disjunction of C_1 and C_2 without v or \bar{v} . The resolvent is a logical consequence of the logical *and* of the pair of clauses. Resolution is the process of repeatedly generating resolvents from original clauses and previously generated resolvents until either the null clause is derived or until no more resolvents can be created [459]. In the former case (a refutation) the formula is unsatisfiable and in the latter case it is satisfiable.

For some formulas the order in which clauses are resolved can have a big effect on how much effort is needed to solve it. The worst-case associated with the best possible order (the order is selected after the formula is given) has received considerable study [184, 524, 239, 532]. These studies all used formulas that have no solution, but where this is not obvious to the resolution algorithm. Eventually a much stronger result was shown: nearly all random l -SAT formulas need exponential time when the ratio of clauses to variables is above a constant (whose value depends on l) [87]. The constant is such that nearly all of the formulas in this set have no solution.

A number of restrictions and at least one extension to resolution have been proposed and applied to *CNF* formulas. Restrictions aim to shorten the amount of time needed to compute a resolution derivation by limiting the number of possible resolvents to choose from at each resolution step. The extension aims to provide shorter derivations than possible for resolution

alone by adding equivalences which offer more clauses on which to resolve. A nice treatment of these refinements can be found in [64], Chapter 4. We mention here a few of these.

Set of Support [561]. Split a given formula into two sets of clauses \mathcal{F}_∞ and \mathcal{F}_f such that \mathcal{F}_∞ is satisfiable. Permit only resolutions involving one clause either in \mathcal{F}_f or an appropriate previous resolvent. Set \mathcal{F}_f is called the support set. This restriction can be useful if a large portion of the given formula is easily determined to be satisfiable.

P- and N-Resolution. If one of the two clauses being resolved has all positive literals (resp. negative literals), then the resolution step can be called a P-resolution (resp. N-resolution) step. In P-resolution (resp. N-resolution) only P-resolution (resp. N-resolution) steps are used. Clearly there is great potential gain in this restriction due to the usually low number of possible resolvents to consider at each step. However, it has been shown that some formulas solved in polynomial time with general resolution require exponential time with N-resolution.

Linear Resolution. We have linear resolution if every resolution step except the first involves the most recently generated resolvent (the other clause can be a previous resolvent or a clause in the given formula). Depending on the choice of initial clause and previous resolvents it is possible not to complete a refutation.

Regular Resolution[524]. In every path of a resolution tree no variable is eliminated more than once.

Davis-Putnam Resolution. Once all the resolvents with respect to a particular variable have been formed, the clauses of the original formula containing that variable can be dropped. Doing this does not change the satisfiability of the given formula, but it does change the set of solutions to the extent that the value of that variable is no longer relevant. When dropping clauses, it is natural to first form all the resolvents for one variable, then all the resolvents for a second variable, and so on. When doing resolution in this way, it is easy to find one satisfying assignment if the formula is satisfiable. At the next to last step the formula has just one variable, so each value can be tested to see which one satisfies the formula (perhaps both

will). Pick a satisfying value and plug it into the formula for the next step, converting it into a one variable formula. Solve that formula and proceed in this manner until an assignment for all variables is found.

Extended Resolution [524]. For any pair of variables a, b in a given formula \mathcal{F} , create a variable z not in \mathcal{F} and append the following expression to \mathcal{F} : $(z \vee a) \wedge (z \vee b) \wedge (\bar{z} \vee \bar{a} \vee \bar{b})$. Judicious use of such extensions can result in polynomial size refutations for problems that have no polynomial size refutations without extension.

The following strategies help reduce the time to compute a resolution derivation.

Subsumption. If the literals in one clause are a subset of those in another clause, then the smaller clause is said to *subsume* the larger one. Any assignment of values to variables that satisfies the smaller clause also satisfies the larger one, so the larger one can be dropped without changing the set of solutions. Subsumption is of particular importance in resolution algorithms because resolution tends to produce large clauses.

Pure Literals. A literal is *pure* if all its occurrences are either all positive or all negative. No resolvents can be generated by resolving on a pure literal, but all clauses containing a pure literal can be removed without loss. An important improvement to the basic resolution algorithm is to first remove clauses containing pure literals (before resolving on non-pure literals) [119].

Although resolution can be applied to SAT, the main reason for interest in resolution is that it can be applied to the more difficult problem of solving sentences of first order predicate logic. There is a vast literature on that subject. Bibel has a good book on the topic [33].

5.2 Backtracking

Backtracking algorithms are based on splitting. During each iteration, the procedure selects a variable and generates two sub-formulas by assigning the two values, *true* and *false*, to the selected variable. In each sub-formula, those clauses containing the literal which is *true* for the variable assignment are erased from the formula, and those clauses which contain the literal which is *false* have that literal removed. Backtrack algorithms differ in the

way they select which variable to set at each iteration. The unit clause rule, the pure literal rule, and the smallest clause rule, are three most common ones. We state each algorithm informally.

The flow of control in splitting-based algorithms is often represented by a search tree. The root of the tree corresponds to the initial formula. The internal nodes of the tree correspond to sub-formulas that cannot be solved directly, whereas the leaf nodes correspond to sub-formulas that can be solved directly. The nodes are connected with arcs that can be labeled with variable assignments.

Simple Backtracking [54]. If the formula has an empty clause (a clause which always has value *false*) then exit and report that the formula has no solution. If the formula has no variables, then exit and report that the formula has a solution. (The current assignment of values to variables is a solution to the original formula.) Otherwise, select the first variable that does not yet have a value. Generate two sub-formulas by assigning each possible value to the selected variable. Solve the sub-formulas recursively. Report a solution if any sub-formula has a solution, otherwise report no solution.

Unit Clause Backtracking [435]. This algorithm is the same as simple backtracking except for how variables are selected. If some clause contains only one of the unset variables then select that variable and assign it a value that satisfies the clause containing it; otherwise, select the first unset variable.

In practice, this improved variable selection often results in much faster backtracking [35].

Clause Order Backtracking [65]. This algorithm is the same as simple backtracking except for how variables are selected. If this setting does not solve the formula, then select the first clause that can evaluate to both *true* and *false* depending on the setting of the unset variables. Select variables from this clause until its value is determined.

By setting only those variables that affect the value of clauses, this algorithm sometimes avoids the need to assign values to all the variables. The algorithm as stated finds all the solutions, but in a compressed form. The solutions come in cylinders, where some variables have the value “don’t care.” Thus, a single solution with unset variables represents the set of solutions

obtained by making each possible assignment to the unset variables.

Probe Order Backtracking [443]. This algorithm is the same as simple backtracking except for how clauses are selected. Temporarily set all the unset variables to some predetermined value. Select the first clause that evaluates to *false* with this setting. Return previously unset variables back to unset and continue as in clause order backtracking.

For practical formulas one should consider adding the following five refinements to probe order backtracking: stop the search as soon as one solution is found, carefully choose the probing sequence instead of just setting all variables to a fixed value [356, 497, 501], probe with several sequences at one time [70, 71], carefully select which variable to set [70, 71], use resolution when it does not increase the input size [167]. The sixth best prize winning entry in the 1992 SAT competition used an improvement on probe order backtracking [71].

Franco [166] noticed that a random assignment solves a nonzero fraction of the formulas in the average l -SAT model when pn is large compared to $\ln m$. Simple uses of that idea does not lead to good average time [443], but combining the idea with clause order backtracking leads to probe order backtracking, which is fast when pn is above $\ln m$. Probe order backtracking appears to have some similarities to one method that humans use in problem solving in that it focuses the algorithm's attention onto aspects of the problem that are causing difficulty, *i.e.*, setting variables that are causing certain clauses to evaluate to *false*. For the same reason it is somewhat similar to some of the incomplete searching algorithms discussed in Section 6.

Shortest Clause Backtracking. This algorithm is the same as clause order backtracking except for the clause selected. In this case, select the shortest clause.

The corresponding idea for constraint satisfaction is to first set a variable in the most constraining relation. This idea is quite important in practice [35].

Jeroslow-Wang [291]. A backtrack search can sometimes be terminated early by checking whether the remaining clauses can be solved by a Linear Programming relaxation (see Sections 9.2 and 9.3). An implementation of this idea can be expensive. Jeroslow and Wang have proposed a

simpler and effective technique that is similar in spirit. The idea is, before splitting, to apply a procedure that iteratively chooses the variable and value which, in some sense, maximizes the chance of satisfying the remaining clauses. The procedure does not backtrack and is, therefore, reasonably fast. Assignments determined by the procedure are temporarily added to the current partial truth assignment. If the procedure succeeds in eliminating all clauses then the search is terminated and the given formula is satisfiable. Otherwise, the procedure fails, control is passed to the split, temporary assignments are undone, and backtracking resumes.

The choice of variable and value at each iteration maximizes the weight $w(S_{i,j})$ where, for a subset of clauses S , $w(S) = \sum_{C \in S} 2^{-|C|}$, and for $i \in \{0, 1\}$, $1 \leq j \leq n$, $S_{i,j}$ is the subset of remaining clauses containing variable v_j as a positive literal if $i = 0$ and as a negative literal if $i = 1$. The length of clause C , denoted $|C|$ above, is the number of literals that are not falsified by the current partial assignment and the sum is over clauses that are not satisfied by the current partial assignment. The weight given above may be compared to that given by Johnson in [293] (see also, *Other Non-Backtracking Heuristics* below).

\star

5.3 Backtracking and Resolution

Some algorithms have adapted ideas inspired by resolution to splitting algorithms. For example, from the resolution view point, pure literals are interesting in that they lead to a single sub-formula that is no more complex than the original formula, while from the perspective of splitting, pure literals lead to two sub-formulas, but the solutions to the sub-formula where the literal has the value *false* are a subset of the one where the literal has the value *true*. Therefore, the original formula has a solution if and only if the formula associated with the *true* literal does.

The Pure Literal Rule Algorithm [204]. Select the first variable that does not have a value. (If all variables have values, then the current setting is a solution if it satisfies all the clauses.) If some value of the selected variable results in all clauses that depend on that variable having the value *true*, then generate one sub-formula by assigning the selected variable the value that makes its literals *true*. Otherwise, generate a sub-formula for both values of the selected variable. Solve the one or two sub-formulas.

5.4 Clause Area

A clause with l distinct literals leads to the fraction $1/2^l$ of the possible variable settings not being solutions. One can think of the clause as blocking out area $1/2^l$ on the Venn diagram for the formula. Iwama showed that combining this idea with inclusion-exclusion and careful programming leads to an algorithm which runs in polynomial average time when $p > \sqrt{(\ln m)/n}$ [288]. If the sum of the area of all clauses is less than 1, then some variable setting leads to a solution. This idea works particularly well with shortest-clause backtracking since that algorithm tends to eliminate short clauses. See [171] for a probabilistic analysis of this idea. No average-time analysis has been done.

5.5 Improved Techniques for Backtracking

This section considers some refinements that can be added to the basic backtracking and resolution techniques. Several of these are similar to techniques that have already been discussed.

Branch Merging. This is complementary to preclusion. Backtracking is frequently used on problems such as the n -queens problem where there is a known symmetry group for the set of solutions. In such cases many search trees possess equivalent branches which can be merged to reduce search effort [35, 557]. The use of the symmetry group can greatly speed up finding the solutions. See [73, 74] for examples from the field of group theory. Brown, Finklestein, and Purdom [52, 53] gave additional problems that arise in making the backtracking techniques work with a backtracking algorithm which needs to set variables in different orders on different branches of the search tree.

Search Rearrangement. This is also known as *most-constrained search* or *nonlexicographic ordering search*. When faced with several choices of extending a partial solution, it is more efficient to choose the one that offers the fewest alternatives [35]. That is, nodes with fewer successors should be generated early in the search tree, and nodes with more successors should be considered later. The vertical (variable) ordering and horizontal (value) ordering are special cases of search rearrangement [55, 178, 435, 439, 442, 511]. The rule used to determine which variable to select next is often called the branching rule. Many researchers are actively investigating the selection of

branching variables in the DP procedures. Hooker studied the branching rule and its effect with respect to particular problem instances [263]. Böhm and Speckenmeyer experimented with branching effect with a parallel DP procedure implemented on an MIMD machine [39]. Boros, Hammer, and Kogan developed branching rules that aim at the fastest achievement of q-Horn structures [42]. Several particular forms of search rearrangement were discussed in Section 5.2.

From 2-SAT to General SAT. In many practical applications, the constraints in the problems are coded as 2-SAT formulas. In SAT problem formulation, very frequently in practical applications, many of the constraints will be coded as 2-SAT clauses.

An important heuristic to SAT problem solving is to first solve 2-SAT clauses with fast polynomial time algorithms. This fast operation can significantly reduce the search space. The truth assignment to the rest of the variables can be handled with a DP procedure. This idea has been used in SAT solver *Stamm* [70, 71], Gallo and Pretolani's 2-SAT relaxation [70, 71, 432], and Larrabee's algorithm [336, 487]. Similar ideas to solving 2-SAT clauses were developed. Eisele's SAT solver uses a weighted number of occurrences whereas occurrences in 2-SAT clauses count more than other occurrences [70, 71]. Dörre further added a limited amount of forward checking to quickly determine 2-SAT formulas in the Eisele-Dörre SAT solver [70, 71]. In the SAT contest [70, 71] the winning programs with 2-SAT solvers were slightly slower than those without.

Similar techniques were developed that use Horn-SAT relaxation in satisfiability testing [109, 186]. In Crawford's Tableau [109], Horn clauses are separated from non Horn clauses. Based on the DPL procedure, Tableau applies in priority the unit clause rule and if necessary branches on a variable selected in the non Horn clauses using three successive heuristics.

Backtracking with Lookahead. A lookahead processor is a pre-processing filter that prunes the search space by inconsistency checking [232, 233, 369, 400]. Backtracking with lookahead processing is performed by interplaying a depth-first tree traversal and a lookahead tree pruning processor that deletes nodes on the search tree whose value assignments are *inconsistent* with those of the partial search path. Techniques in this class include partial lookahead, full lookahead [232, 233, 248], forward checking [248, 248], network-based heuristics [369, 129], and discrete relaxation

[232, 233, 319, 461].

Backtracking for Proving Non-Existence. Dubois, Andre, Boufkhad, and Carlier have recently proposed a complete SAT algorithm, *CSAT* [152]. The *CSAT* was developed for the proof of the non-existence of a solution. The algorithm uses a simple branching rule and a local processing at the nodes of search trees (to detect further search path consistency and make search decision). It performed efficiently on some DIMACS benchmarks.

Intelligent Backtracking. This is performed directly to the variable that causes the failure, reducing the effect of thrashing behavior. Methods in this category include dependency-directed backtracking [506, 146], revised dependency-directed backtracking [424], simple intelligent backtracking [180], and a number of simplifications [57, 120, 121, 122, 124, 126, 193, 248, 462].

Freeman [177] recently present an intelligent backtracking algorithm, *POSIT*, for PrOpositional SatIstiability Testbed. In this algorithm he used Mom's heuristic, detecting failed literals, and minimizing constant factors to speed up backtracking search.

5.6 Some Remarks on Complexity.

The worst-case time for all known SAT algorithms is exponential in the first power of the input size. The naive algorithm that tries every variable setting requires time 2^n for n variable formulas. For l -SAT, the best known bound on worst-case complexity has been worked down from 1.618^n [395] to slightly below 1.5^n obtained by Schiermeyer [474, 475]. Other work on the topic is given in [195].

As with other NP-complete problems there are no exponential lower bound results for SAT. However, it has been proven that all resolution algorithms need time that is exponential in the first power of the input size [239, 87, 532]. No such lower bound analyses have been done on splitting-based algorithms.

For a comprehensive treatment of the complexity of propositional proofs, see a recent survey by Urquhart [535].

6 Local Search

Local search is a major class of discrete, unconstrained optimization procedures that can be applied to a discrete search space. Such procedures can be used to solve SAT by introducing an objective function that counts the number of unsatisfiable (*CNF*) or satisfiable (*DNF*) clauses and solving to minimize the value of this function [212, 214, 217, 226, 413, 482].

In this section, we briefly summarize local search, its early development for the SAT problem, a search space model, and some basic heuristics for local search. We then describe randomized local search, randomized local search with trap handling, and greedy local search in detail.

6.1 Framework

Local search, or *local optimization*, is one of the primitive forms of continuous optimization applied to a discrete search space. It was one of the early techniques proposed to cope with the overwhelming computational intractability of NP-hard combinatorial optimization problems.

There have been two major periods for the development of local search. Early local search method was able to solve small size, unconstrained problems such as small TSP instances [359]. During the middle and late eighties, more powerful techniques (see Section 6.4) for randomized local search were developed which were capable of solving large size, highly constrained CSP, SAT and scheduling problems efficiently [211, 214, 498, 501, 230].

Given a minimization (maximization) problem with objective function f and feasible region R , a typical local search procedure requires that, with each solution point $\mathbf{x}_k \in R$, there is a predefined *neighborhood* $N(\mathbf{x}_k) \subset R$. Given a current solution point $\mathbf{x}_k \in R$, the set $N(\mathbf{x}_k)$ is searched for a point \mathbf{x}_{k+1} with $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$ ($f(\mathbf{x}_{k+1}) > f(\mathbf{x}_k)$). If such a point exists, it becomes the new current solution point, and the process is iterated. Otherwise, \mathbf{x}_k is retained as a *local optimum* with respect to $N(\mathbf{x}_k)$. Then, a set of feasible solution points is generated, and each of them is “locally” improved within its neighborhood. To apply local search to a particular problem, one needs only to specify the neighborhood and the procedure for obtaining a feasible starting solution.

Local search can be efficient for two reasons. First, at the beginning of search, a full assignment is assigned to all the variables in the search space. Search efforts are focused on a single path in the search space. Second, local search refines for improvement within its local neighborhood using a

testing for improvement and, if there is any improvement, takes an action for improvement. Since the objective function has a polynomial number of input numbers, both *testing* and *action* can be done efficiently. Little effort is needed to generate the next solution point. A major weakness of local search is that the algorithm has a tendency to get stuck at a locally optimum configuration, i.e., a local minimum.

Greedy local search pursues only paths where every step leads to an improvement, but this leads to a procedure that becomes stuck much more often than the randomized local search. Greedy local search procedure gets stuck in flat places as well as at local minima.

Many search techniques, such as statistical optimization [75, 477], simulated annealing [315], stochastic evolution [467], and conflict minimization [211, 389, 495, 501], are either local search or variations of local search. For most search problems encountered, in terms of computing time and memory space, local search often achieves many orders of magnitude of performance improvement over conventional techniques such as Branch-and-Bound [214, 217, 446, 497, 501].

6.2 Early Work

Early work on constraint satisfaction, simulated annealing, and complexity theory contributed significantly to the original development of local search algorithms for SAT problem. Five notable early developments are: (1) Boolean local relaxation, (2) *min-conflict* formulation for SAT and CSP, (3) the *n*-queen models and algorithms for scheduling applications, (4) a simulated annealing algorithm, and (5) a 2-SAT algorithm.

1. Boolean Local Relaxation.

Boolean local relaxation may be viewed as a kind of deterministic local search. The basic idea was to formulate conflicts among truth values in a set of Boolean objective functions and then use a Boolean local search procedure to minimize the conflicts. This Boolean *min-conflict* heuristic was developed for solving the CSP and SAT problems.

For a variable having m values, m Boolean labels are used to indicate the variables' instantiation to a particular value. The conflicts produced by a value assignment are accumulated in a set of Boolean objective functions (one for each label). The objective function for the i th variable and k th

label, $f_{i,k}$, is defined as [211]:

$$f_{i,q} = \prod_{j=1}^n \sum_{p=1}^m l_{i,q} \wedge C_{i,j}(q, p) \wedge l_{j,p}, \quad (19)$$

where $C_{i,j}(q, p)$ is a constraint between labels $l_{i,q}$ and $l_{j,p}$. Note that the right hand side of Eq. (19) is a *CNF* formula with *extended literals*.

In an early implementation, the local conflict minimization procedure was performed using a Boolean relaxation procedure (Figure 8). During each

```

procedure DRA()
boolean inconsistency;
begin
    inconsistency := TRUE;
    k := 0;
    while inconsistency = TRUE do
        begin
            inconsistency := FALSE;
            for variable  $i := 1$  to  $n$ 
                for label  $q := 1$  to  $m$ 
                begin
                     $f_{i,q}^{k+1} := \text{evaluate\_objective\_function}(l, C);$ 
                    /* local conflict minimization */
                    if  $f_{i,q}^{k+1} = f_{i,q}^k$  then continue;
                    if  $f_{i,q}^{k+1} < f_{i,q}^k$  then
                         $f_{i,q}^k := f_{i,q}^{k+1};$  inconsistency := TRUE;
                    end;
                    k := k + 1;
                end;
            end;
        end;

```

Figure 8: **DRA**: A local relaxation algorithm.

iteration, the algorithm checks each variable for every label and iteratively minimizes the objective functions by flipping truth values assigned to the labels: If the objective function does not change, keep it; If the objective

function can be reduced, reduce it, and report the *inconsistency* status. The iteration will terminate once the *inconsistency* signal turns off.

Boolean local relaxation was suitable to VLSI circuit implementation. During 1985 to 1988, several parallel DRA algorithms, i.e., DRA2, DRA3 and DRA5, were implemented in VLSI chips to speed up CSP/SAT computations [211, 232, 554, 233]. Furthermore they were used in backtracking search for CSP and SAT applications [211].

Because of its iterative local conflict minimization and its direct applications to SAT and CSP, Boolean local search made itself a *predecessor* of several early local search algorithms for CSP, SAT and scheduling problems.

2. *Min-Conflict* Formulation for SAT and CSP.

Objective functions in DRA algorithms were defined for Boolean labels. During the late eighties, Gu [211] observed that if the *conflicts* of all the Boolean objective functions were formulated in *one* objective function, then the global minimum of the objective function would correspond to a conflict-free solution of the given CSP problem. Two first such formulations were developed to solve SAT and n -queen problems.

In the SAT problem, the problem “board” is an n by m rectangle matrix with n variables placed in the horizontal direction and m clauses placed in the vertical direction (Figure 9 (a)). If we evaluate variables in the horizontal direction and count the number of unsatisfiable clauses (i.e., “conflicts”) in the vertical direction, we can formulate an objective function as the number of unsatisfied clauses over *all* the variables [211, 212, 214, 217]. Thus, the global minimum of the objective function corresponds to the solution of the SAT problem. Reducing the objective function to zero would produce a solution to the SAT problem. For the n -queen problem [211], we place n queens in the columns of their row’s permutation. This guarantees that no two queens would attack each other on the same row or the same column. The “conflicts” among queens possibly occurring on the diagonal lines are collected to formulate the objective function for solving the n -queen problem (see Figure 9 (b)). Reducing the conflicts to zero would produce a solution to the n -queen problem.

Following conflict formulation of objective function, apparently, the iterative local minimization procedure used in the Boolean relaxation becomes a local search procedure to minimize the objective function. This gave the *min-conflict* heuristic in the discrete space. This idea led directly to the early design of *SAT1* algorithms, the first local search algorithm for SAT. Follow-

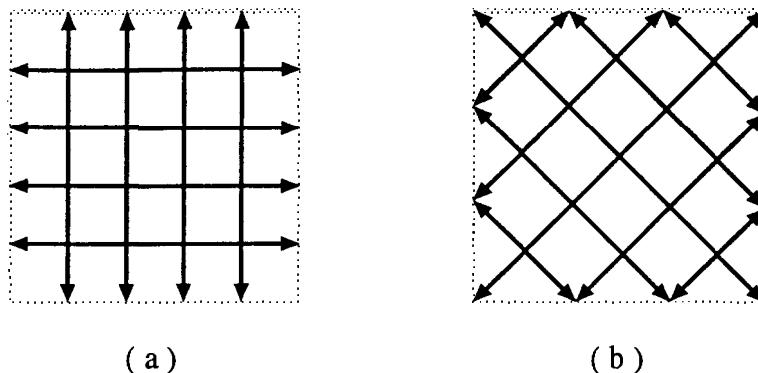


Figure 9: The conflicting direction in (a) the satisfiability problem and (b) the n -queen problem.

ing this, Gu gave a number of randomized local search algorithms for the SAT problem [212, 214, 217]. Furthermore efficient heuristics (Section 6.4) were developed to improve the performance of the local search algorithms. Due to the important industrial applications at that time, the effectiveness of the *SAT1* algorithms was tested through two CSP benchmarks, i.e., the SAT problem and the n -queen problems.

An important industrial application for SAT was VLSI engineering. Many VLSI circuit design problems, such as VLSI circuit testing, synthesis, and routing, can be formulated as instances of the SAT or MAX-SAT problems. During the late eighties, the *SAT1* algorithm family was first developed for VLSI applications and was found to be efficient for many VLSI circuit design problems. At that time, however, there was little progress in the theoretical analysis of the *SAT1* algorithms.

Another important application area for the SAT problem was industrial scheduling. During the late eighties, IBM and NASA were working on a number of important scheduling projects. These applications involved solving large size scheduling and task assignment problems under critical timing, spatial, and resource constraints. The scheduling and task assignment problems are well-known as the satisfiability problem. SAT formulas characterize these problems precisely. Significant local search solutions to the scheduling and task assignment problems were derived from the *SAT1* algorithm family. Due to its abstract *CNF* formulation, however, the SAT problem was not able to provide a descriptive geometric model that was able

to demonstrate the scheduling and task assignment operations expressively. And it happened to be that the n -queen problem provided an ideal model for these large size scheduling problems.

3. N -Queen Scheduling Models and the QS Algorithms.

The n -queen and scheduling are two problems in constraint satisfaction. During the late eighties, Gu worked on various n -queen problem models for combinatorial optimization [211, 234] and found that, by a remarkable coincidence, the n -queen model represents a significant model for scheduling and task assignment problems.

The underlying structure of the n -queen problem, represented by a complete constraint graph, gives a relational model with fully specified constraints among the multiple objects [211]. Variations on the dimension, the objects' relative positions, and the weights on the constraints led to a **hyper-queen** problem model which had several simple and basic models:

- **n -queen** problem: the *base* model. N queens are indistinguishable and the constraints among queens are specified by the binary values (i.e., 1 or 0).
- **w -queen** problem: the *weighted* n -queen model. N queens are distinguishable. Each is associated with a cost. The constraints among queens are specified by some weights.
- **3d-queen** problem: queens are to be placed in a 3-dimensional ($l \times m \times n$) rectangular cuboid. A special case, **nm-queen**, is to place queens on an n by m rectangle.
- **$q+-$ queen** problem: more than one queens are allowed to be placed on the same row or the same column.

The *hyper*-queen problem can model the objects/tasks, the performance criteria, the timing, spatial, capacity and resource constraints for a wide range of scheduling and task assignment problems. This made the n -queen problem a general model for many industrial scheduling and task assignment problems. By a remarkable coincidence, the models of several difficult scheduling projects at that time were either the n -queen or the *hyper*-queen problems [299, 511]. All of them required efficient solutions to the n -queen or *hyper*-queen problems.

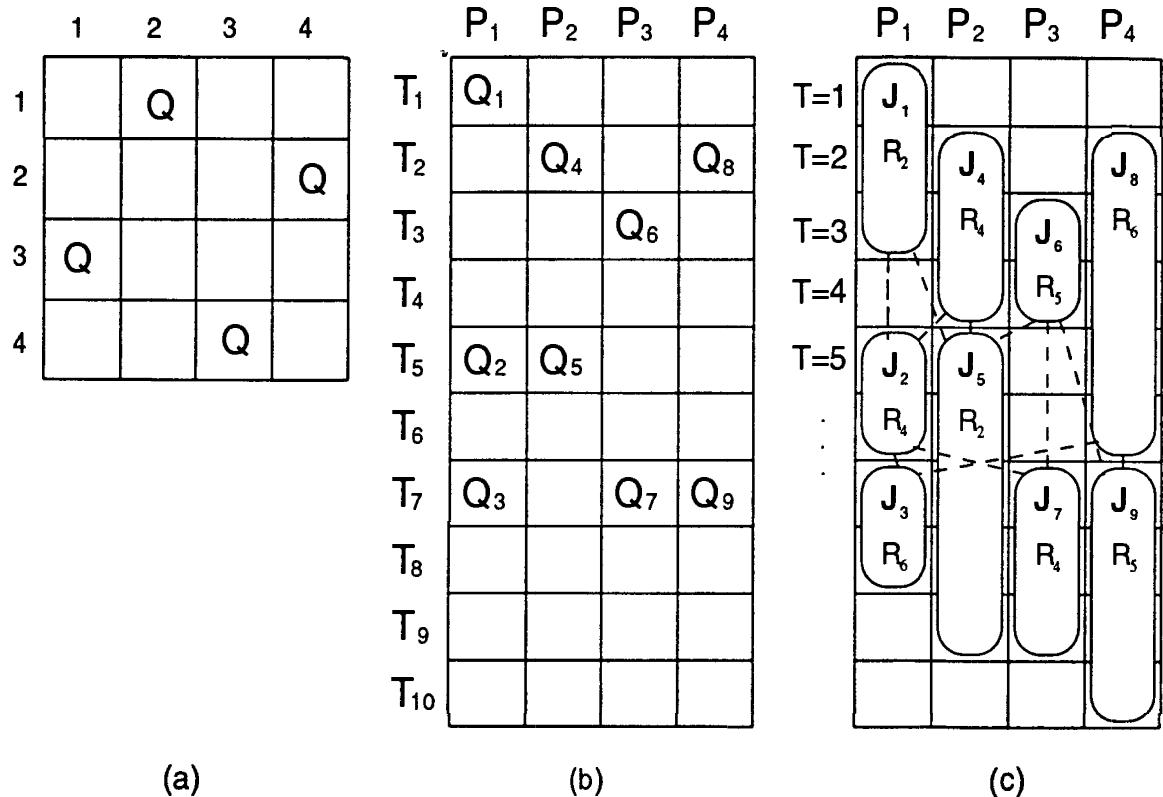


Figure 10: The n -queen scheduling model. Instead of minimizing conflicts among queens on diagonal lines, where we reduce conflicts among jobs and minimize the longest execution path following the given topological constraints.

Let's consider a static scheduling problem where we are to schedule an edge-weighted *directed acyclic graph* (DAG), or called *task graph*, to a set of homogeneous processors to minimize the completion time. This problem is known to be NP-hard in its general forms [191]. Using the *hyper-queen* model, a set of weighted queens is to be placed on a T by P rectangle (see Figure 10). Here T denote the execution time, P the number of processors, J_i the i th queen (i.e., the i th job and its execution time), and c_{ij} the communication time from the i th job to the j th job, the goal is to place the *job queens* onto the T by P board and minimize the longest execution path, following the given topological constraints. Instead of minimizing conflicts among queens on diagonal lines as in the n -queen problem, here we reduce conflicts among jobs and minimize the longest execution path following the given topological constraints.

We search different geometric patterns in the n -queen and the scheduling problems. These patterns produce different constraints in CSP and do not affect the formulation and the use of *min-conflict* heuristic. In fact, once the constraints are extracted, an algorithm developed for the n -queen problem can be used to solve the scheduling problem with or without minor modifications. The *hyper-queen* models has freed the n -queen problem from its puzzle game background and made many practical scheduling and task assignment applications feasible. These include [219, 501] task scheduling, real-time system, task assignment, computer resource management, VLSI circuit design, air traffic control, communication system design, and so on.

Polynomial time, analytical solutions for the n -queen problem exist but they cannot solve the general search problems and have no use in practice [2, 10, 31, 157, 259, 455]. Following local conflict minimization [211, 232], a *QS1* algorithm was developed during late 1987 and was implemented during early 1988. It was the first local search algorithm developed for solving the large size n -queen problem [211, 494, 495, 496]. Three improved local search algorithms for the n -queen problem were developed during 1988 to 1990 [328, 329, 302, 464]. *QS2* is a near linear-time local search algorithm with an efficient random variable selection strategy [497]. *QS3* is a near linear-time local search algorithm with efficient pre- and random variable selection and assignment [497]. *QS4* is a linear time local search algorithm with efficient partial and random variable selection and assignment techniques [498, 501]. Compared to the first local search algorithm [211], partial and random variable selection/assignment heuristics have significantly improved search efficiency by orders of magnitude. *QS4*, for example, was able to solve 3,000,000 queens in a few seconds.

Three years after releasing the *QS1* algorithm, Minton *et al.* independently reported a similar local search algorithm for the n -queen problem [388, 389]. A major difference between Minton's algorithms and Sosic and Gu's algorithms was that Minton's algorithm was a one dimensional local search without using random heuristics.

Early local search solutions for scheduling applications were developed during the late eighties. Since then more than one hundred industrial companies worldwide have developed similar scheduling software systems for various industrial scheduling applications.

4. A Simulated Annealing Algorithm for *Max-SAT*.

Motivated by the method of simulated annealing, Hansen and Jaumard [246] proposed a steepest ascent, mildest descent algorithm for the maximum satisfiability (*Max-SAT*) problem. In this approach, Hansen and Jaumard focused on a local change and defined an objective function based on a switching variable and its related clauses. The objective function maximizes local compensation for each variable which can be used for solving the *Max-SAT* problem. The objective function can not be used for the SAT problem unless another objective function whose global minimum corresponds to a solution of the SAT problem is given. Furthermore, Hansen and Jaumard used local optima checking to handle the local optimum and found it by providing additional guidance to the search direction.

5. Theoretical Study for *2-SAT* and *SAT1*.

During the study of the complexity of a certain natural generalization of SAT, Papadimitriou gave a randomized algorithm for the 2-SAT problem [412]. Furthermore Papadimitriou showed that such a randomized algorithm finds assignments for 2-SAT instances in $O(n^2)$ steps with probability approaching one, where n is the number of variables. With further extensions [414], the algorithm can be applied to solve the random 3-SAT problems.

During the early nineties, researchers started to work on the theoretical analysis of local search algorithms for CSP and SAT problems. In 1991 two theoretical studies that focused on the SAT problem were reported. Gu and Gu took three algorithms (i.e., *SAT1.1*, *SAT1.2*, and *SAT1.3*) from the *SAT1* algorithm family and made average time complexity study for the SAT problem [226].

Early on, local search methods for the large size SAT, n -queen and scheduling problems attracted great attention in the AI area. This was partly due to the close relationship among SAT, n -queen and scheduling that all three problems are special cases in CSP. If one can find an efficient (non-analytical) search algorithm for one of them, then the algorithm can be directly translated to an efficient algorithm for the other problems.

6.3 A Three-Level Search Space Model

A large number of real experimental data suggest that there are several typical local minimum structures when solving NP-hard optimization problems (see Figure 11). A *valley* and a *basin* are ideal cases that one can find a

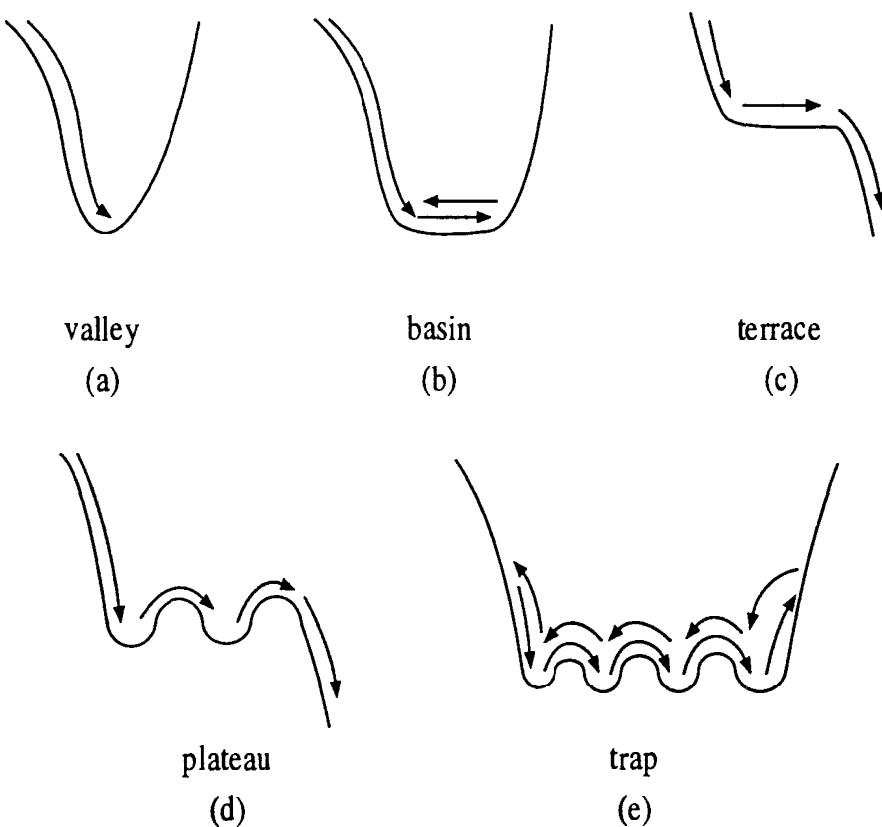


Figure 11: There are a number of local minimum structures. A trap is a “well” of local minima and is difficult to deal with.

global minimum quickly. Local search and the related heuristics can handle a *terrace* and a *plateau* without much difficulty. The most difficult situation is a *trap* where a group of local minima is confined in a “well.” The search process walks around the set of local minima periodically and cannot get away without special mechanism. In general there may be many traps in a search problem. The characteristics of a trap are closely related to the objective function used, the search space structure, and the search algorithm structure.

Further observations suggest that a search space may be roughly divided into several different levels, depending on the problem structures (Figure 12).

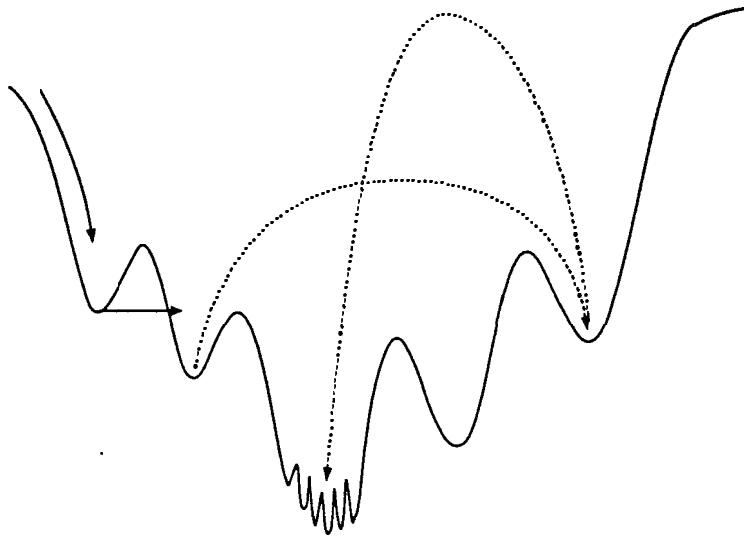


Figure 12: An informal example of the three-level search space model. A search process would go through an *open search* in the upper open portion of the search space, a *peak search* in the middle portion of the search space, and a *trap search* in the valley portion of the search space.

In the model, a search space is roughly viewed in three levels: top level, middle level, and bottom level. The *top level* is the upper open portion of the search space with smoothing edges. For a minimization problem, most optimization algorithms can descend quickly in the top level and thus perform quite well. The *middle level* is the middle portion of the search space where there are relatively big “mountain peaks.” During the descent, the search process may encounter problems and it may have to use some tunneling and random heuristics (see Section 6.4) to proceed. The *bottom level* is the bottom portion of the valleys (particular the lowest valley) where there are many traps. When a local search process falls into a trap it may become locked into a loop of local minima. Most algorithms do not succeed in this stage and have difficulty continuing.

The above observations suggest to use multiphase search to handle the NP-hard problems [214, 228, 224]. That is, we may use an *open search* in the top level, a *peak search* for searching “coarse” peak structures in the middle level, and a *trap search* for tracking “fine” rugged trap surface structures in the valleys. Following this, a variety of heuristics and techniques have been proposed to improve the performance of local search.

6.4 Four Components in Local Search

A number of efficient local search algorithms for SAT, CSP and scheduling problems have been developed since 1987. Past lessons showed that the following **four components** are crucial to the development of an efficient local search algorithm for the satisfiability and NP-hard problems. They are: (1) the min-conflict heuristics, (2) the best-neighbor heuristics, (3) the random value/variable selection heuristics, and (4) the trap handling heuristics.

Some of these heuristics can help local search handle local minima, performing effective global search.

1. The Min-Conflicts Heuristics.

Different forms of min-conflict heuristics were proposed during 1985 and 1987 for solving SAT, CSP and scheduling problems [211].³ The min-conflict heuristics aim at performing local conflict minimization in Boolean, discrete, and real spaces which are important to handle constraints in a constrained optimization problem [464]:⁴

Min-Conflict Heuristic (Boolean Space) [211]. Multiple values to be assigned to a variable are represented by a vector of Boolean labels. Each Boolean label, either “1” or “0,” indicates the variable’s instantiation to a specific value. Two labels are *conflicting* if their values do not satisfy the given constraint. The *conflicts* (due to an assignment) are formulated as a set of objective functions. The objective functions are minimized by changing values assigned to the labels.

Min-Conflict Heuristic (Discrete Space) [211]. Interrelated objects are chosen as variables. Two variables are *conflicting* if their values do not satisfy the given constraint. The *number of conflicts* (due to an assignment) is formulated in an objective function. The objective function is iteratively minimized by changing values assigned to the variables.

Min-Conflict for SAT [212, 226, 214, 215, 217]. Using *conflicts* as

³In the early days *min-conflict* was variously called inconsistency removing, inconsistency resolution, conflict resolution, enforce local consistency, and local conflict minimization [211]. Later, Minton shortened these words into a concise term: min-conflicts.

⁴The min-conflict heuristics also work in real space (see examples in [215, 216, 218]).

objective [211], the objective function for the SAT problem gives the number of unsatisfied clauses. A *CNF* is true *if and only if* the objective function takes the global minimum value 0 on the corresponding solution point.

This objective function is the basis for the design of the *SAT1*, *SAT2*, *SAT3*, and *GSAT* algorithms [214, 215, 226, 217, 482].

Performance. The min-conflict heuristics have been applied to solve the CSP, SAT and scheduling problems since 1985 [211, 212, 495, 494, 226, 214]. They showed significant performance improvements when compared to traditional backtracking search algorithms. The effectiveness of min-conflicts heuristic was further observed by Russel and Norvig [464], Kumar [328, 329], Johnson [302], Minton *et al.* [389], and Selman *et al.* [482].

Early local search algorithms can handle small size path-finding problem such as small TSP instances. By grouping conflicts into an objective function, local search with min-conflicts heuristics can now solve large size, highly constrained SAT, CSP and scheduling problems.

2. The Best-Neighbor Heuristics.

Local search proceeds by taking *any* feasible solution point that reduces the objective function. Among many neighboring feasible solution points, local search does not take into account its neighbors' relative performance with respect to the objective function.

Best-Neighbor Heuristic [214, 217, 218]. A *greedy* local search algorithm selects the *best neighbor* that yields the minimum value to the objective function and takes this *best neighbor direction* as the descent direction of the objective function.

In a real search space, continuous optimization algorithms can find the best neighbor feasible solution efficiently. A number of local search and nonlinear optimization algorithms have been developed to solve the SAT problem [214, 217, 218]. The first version of the *GSAT* algorithm was proposed as a *greedy* local search algorithm [482].

Performance. Greedy local search pursues only paths where every step leads to an improvement, but this leads to a procedure that becomes stuck much more often than the randomized local search. Greedy local search procedure gets stuck in flat places as well as at local minima depending on the application cases. In practice, the best neighbor heuristic should be

used in conjunction with random value/variable selection and trap handling heuristics described next.

3. The Random Value/Variable Heuristics.

Random value assignment and random variable selection techniques are fundamental to the design of an effective local search algorithm for NP-hard problems [234, 209, 229].

Random Flip Heuristic [211, 214, 215, 226, 217]: Randomly flip the truth values of $1 \leq k \leq n$ variables in the SAT formula.

This simple heuristic was first implemented in several *SAT1* algorithms as *local handler(s)* in 1987. It has been proven to be effective in improving the performance of local search algorithms [214, 226, 215, 217].

During 1988 to 1990 a similar heuristic, **random swap**, was used to develop local search algorithms for the CSP (e.g., *n*-queen) problems. It showed significant performance improvement for solving large size *n*-queen problems [211, 496, 497, 498, 501].

Random Value (Assignment) Heuristics [211, 214, 226, 217, 359, 411, 497, 498, 501]. These include: randomly select a value that generates the minimum number of conflicts; randomly select a value if there is a symmetry (i.e., more than one value producing the same performance); and randomly select a value for conflict minimization when local minima are encountered.

A simple random value assignment heuristic, **random disturbance**, was used early in solving the TSP problem.

Random Variable (Selection) Heuristics [211, 214, 226, 217]. There are two important heuristics:

1. **Any Variable Heuristic:** select any variable randomly.
2. **Bad Variable Heuristic:** select a variable from the conflicting variables randomly.

The random variable selection heuristic is one of the most important heuristics in the design of local search algorithms for NP-hard problems. It was first used in the local search solution for the SAT problem [212] and then used for the local search solution for the CSP (e.g., *n*-queen) problems [497].

Conflicting variables in the SAT problem contribute to the unsatisfied clauses. Accordingly we have:

Bad Variable Heuristic for the SAT problem [214, 217, 497, 498, 501, 496, 494]: randomly select a variable in the unsatisfied clauses for conflict minimization.

The bad variable heuristic was first implemented to solve the large size n -queen problems during 1988 to 1990 [497, 498, 501, 496, 494] and was implemented in the *SAT2* algorithm in 1990 [214, 217]. The bad variable heuristic was independently developed by Papadimitriou for solving the 2-SAT problem in 1991 [412] and was recently used in the development of *GSAT* and *WSAT* algorithms by Selman *et al.* [485].

Partial/Pre- Random Variable Selection Heuristics [211, 214, 226, 217]. Partial variable random selection makes use of partial or alternating variable selection techniques. Variants of partial random selection include partial and alternating selection of conflicting and consistent variables, a combination of partial deterministic and partial random variable selection, partial interleaved selection of the different search phases, and partial random selection with meta-heuristic control. The simplest selection strategies include: select a variable deterministically (randomly) and select another variable randomly for conflict minimization; select a variable deterministically (randomly) from the conflicting variables and select another variable randomly for conflict minimization; select a variable deterministically and select another variable randomly from the conflicting variables for conflict minimization; during certain periods of search, select a variable deterministically (randomly) and select another variable randomly for conflict minimization; during certain periods of search, select a variable deterministically (randomly) from the conflicting variables and select another variable randomly for conflict minimization; during certain periods of search, select a variable deterministically and select another variable randomly from the conflicting variables for conflict minimization.

Partial Random Variable Selection Heuristics for the SAT problem: a variable may be selected from the unsatisfied clauses in a random, partially alternating, partially periodic, or partially interleaving order.

The partial and pre- random variable selection heuristics were implemented in the *SAT3* algorithm in 1990 [214, 217] and were used to solve

the large size n -queen problems around 1990 [497, 498, 501, 496, 494]. A similar heuristic to the partial random variable selection, *random walk*, was developed by Selman, Kautz, and Cohen independently in 1994 [485].

Performance. Random and partial variable selection heuristics were introduced in the design of *SAT1*, *QS2*, *QS3*, and *QS4* algorithms [212, 226, 214, 217, 495, 497, 498, 501]. They can overcome the weakness of the greedy local search algorithms. Compared to greedy local search, they can offer many orders of magnitude of performance improvements in terms of computing time, solving hard, large size SAT, CSP and scheduling problems with multi-million variables in seconds [214, 217, 497, 501]. They were used in the design of *SAT1.5*, *SAT2*, and *SAT3* algorithms [214, 217].

Selman *et al.* have recently applied a number of random variable selection heuristics to improve the performance of the *GSAT* algorithm and found them effective [485].

4. The Trap Handling Heuristics.

Search is a process of combating local minima. When the search process is approaching the final search stage, trap handling heuristics are needed to cope with local minima and traps (see Sections 6.3 and 6.6).

Tunneling Heuristic [226, 217, 555]: Change the value of a variable if it does not change the value of the objective function.

Tunneling Heuristic for the SAT Problem [226, 217, 482]: Flip the truth value of a variable if it does not change the value of the objective function (see Section 6.5).

Local Tracking Heuristics [214, 224]. Local tracking heuristics are used to track and break local loops (a periodic occurrence of a set of local minima). Several frequently used heuristics include: track local loop(s) when falling into a trap; give low priority to flip to the variables in a local minimum loop; give high priority to flip to variables that lead to a *new* descending direction; lock and release trapping variables periodically, adaptively, or statistically; move gently in a trap to handle fine local structures; move strongly in a trap to handle coarse local structures; jump out of a trap if walking inside it sufficiently long.

Multiphase Search Heuristics [212, 214, 498, 501, 217, 446, 555, 224]. Multiphase heuristics are a part of multispace search heuristics [227, 234, 229]. They have been developed to adapt to the different phases of a search process: perform a poor initial search and then a fine local search for conflict minimization; perform a good initial search and then a fine local search for conflict minimization; perform a good initial search, then a rough local search, and a fine local search for conflict minimization; perform an initial search, and then a rough local search and a fine local search alternatively for conflict minimization; perform a rough initial search, then a coarse local search, and finally, a fine local search for conflict minimization.

Multispace Search Heuristics [227, 234, 229]. Structural multispace operations have been developed that empower a search process with an information flux which is derived from a sequence of stepwise structural transformations. These include multispace scrambling, extradimension transition, search space smoothing, multiphase search, local to global passage, tabu search, and perturbations. They can disturb the environment of forming local minima and facilitate efficient local search when there are many local minima.

Performance. Trap handling heuristics have significantly improved the search efficiency of the *SAT1.5* algorithm [214, 217, 224] (see Section 6.6). Multiphase and multispace search heuristics have been applied to a variety of practical applications and found to be effective [227, 214, 217, 224, 446, 555, 498, 501].

6.5 Randomized Local Search

In this section, we describe the basic structure and major components of the randomized local search algorithms for the SAT problem.

Model. Most discrete local search procedures were developed based on a discrete, unconstrained optimization model, the *SAT1 model* [212, 214, 217, 226]. In this model, the truth values assigned to the variables are defined as:

$$x_i = \begin{cases} 1 & \text{if the variable has value } \textit{true} \\ -1 & \text{if the variable has value } \textit{false} \end{cases} \quad (20)$$

The objective function, $F(\mathbf{x})$, in the *SAT1* model counts the number of unsatisfied clauses as its objective value. A *CNF* is true if and only if $F(\mathbf{x})$ takes the global minimum value 0 on the corresponding \mathbf{x} .

Basic Local Search. The *SAT1* algorithm for the SAT problem is shown in Figure 13. It consists of an initialization stage and a search stage. At the beginning of search, a SAT formula is generated. An initial random solution is chosen. The number of unsatisfiable clauses is computed and is assigned as the value of the objective function. During each iteration, function *test_flip()* performs a test to see if the objective function would not increase after a flip. If *test_flip()* returns *true*, a flip operation is performed by procedure *perform_flip()*. Then function *evaluate_objective_function()* updates the objective function.

The procedure terminates when the objective function is reduced to zero, i.e., a solution to the given SAT instance is found. In practice, before the objective function reduces to zero, the procedure may become stuck at local minima. In the *SAT1* algorithm [214, 226], a simple randomized local handler performing random flips was used (Figure 14). It combines local descent (reducing objective function) with the random uphill moves (increasing objective function), improving *SAT1*'s convergence performance effectively. In the *SAT1* algorithm family, one or more local handlers were implemented [226, 214, 217]. If the algorithms have difficulty to proceed, the algorithms will call the local handlers and use other heuristics (see Section 6.4) to improve algorithms' convergence performance.

The random flips used in the *SAT1* algorithms make the *order* of selecting which variable for local examination (i.e., the *for* loop) trivial [226, 214, 217]. One can essentially select any variable randomly for examination.

Tunneling Heuristic. A local handler and its activating condition(s) have significantly effect on the performance (e.g., running time and average running time) of a local search algorithm for the SAT problem. The conditions for activating local handlers differ from algorithm to algorithm (see Figure 14). In *SAT1.1* algorithm, the local handler is called if the objective function is not zero (an aggressive strategy) [226, 217]. In *SAT1.2* algorithm, the local handler is called if the objective function does not increase [226, 217]. In *SAT1.3* algorithm, the local handler is called if the objective function does not increase or the objective function is greater than zero for some iterations [226, 217]. In the last two algorithms, the condition “ob-

```

procedure SAT1 ()
begin
    /* initialization */
    get_a_SAT_instance();
     $\mathbf{x}_0 := \text{select\_a\_random\_initial\_point}();$ 
     $F(\mathbf{x}_0) := \text{evaluate\_objective\_function}(\mathbf{x}_0);$ 

    /* search */
     $k := 0;$ 
    while  $F(\mathbf{x}_k) \neq 0$  do
        begin
            for each variable  $i := 1$  to  $n$  do
                /* if flip( $x_i, \bar{x}_i$ ) does not increase  $F$  */
                if test_flip( $x_i, \bar{x}_i$ ) then
                    begin
                         $\mathbf{x}_{k+1} := \text{perform\_flip}(x_i, \bar{x}_i);$ 
                         $F(\mathbf{x}_{k+1}) := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
                    end;
                /* random flips */
                if local then local_handler();
                 $k := k + 1;$ 
            end;
        end;
    end;

```

Figure 13: **SAT1**: a randomized local search procedure for the SAT problem [214, 217, 226]. Random flips are introduced (1) to disorder the sequence with which the variables are selected for local optimization, and (2) to perturb local search with randomized downhill, tunneling, or uphill moves [226, 217].

```

procedure Local_handler ()
begin
    random select some variable  $x_t$ 's;
     $\mathbf{x}_{k+1} := \text{perform\_flip}(x_t\text{'s}, \bar{x}_t\text{'s});$ 
     $F(\mathbf{x}_{k+1}) := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
end;

```

Figure 14: A simple randomized local handler used in the *SAT1* algorithms [214, 217, 226]. Random flips or a new random solution were applied to the algorithm if (1) $F \neq 0$ (*SAT1* algorithm), (2) $F > 0$ (*SAT1.1* algorithm), (3) $F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k)$ (*SAT1.2* algorithm), and (4) $F > 0$ and $F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k)$ (*SAT1.3* algorithm).

jective function does not increase" means that the objective value is either reduced (*local descent*) or remained unchanged (*tunneling heuristic*).

Instead of making a random swing in the vertical direction in the search space, whenever a local minimum is encountered, one can *tunnel* through the rugged terrain structure in a horizontal direction, moving from one local basin to another local basin in an attempt to locate a better locally optimal solution (Figure 15). A tunnel can be thought of as a short-cut passing through a mountain separating points of equal elevation. Whenever a local minimum is encountered, a tunnel is made through a mountain to a neighboring basin as long as this does not change/increase the objective function. Tunneling can be used to search a region with local minima effectively. The behavior of local search with tunneling illustrates the fact that seemingly innocuous changes in an optimization routine can have a surprisingly large effect on its performance. When tunneling was first implemented in the *SAT1* algorithm in the late eighties, it was proven to be effective for solving SAT problem instances.

Random Flips (Noise). If the local search procedure becomes stuck at a local minimum, it may proceed by using a noise perturbation to change its current location in the search space. The effectiveness with which local minima are handled significantly affects the performance of a local search algorithm. Researchers have proposed a number of techniques such as *jumping*, *climbing*, *annealing*, and *indexing* to handle local minima [227]. In simu-

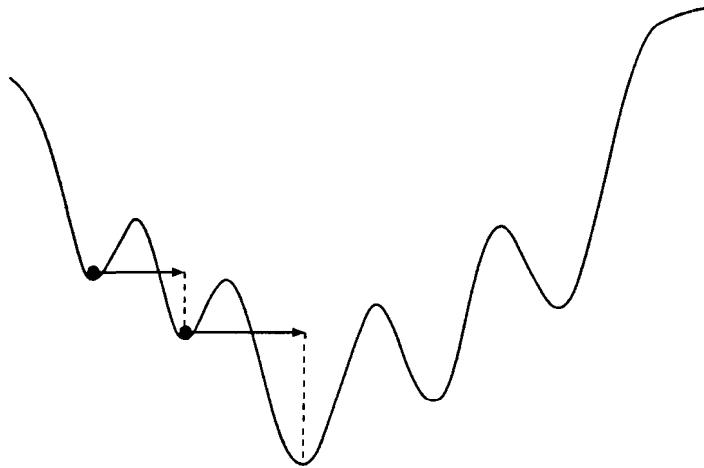


Figure 15: Tunneling

lated annealing, a search process occasionally moves up rather than down in the search space, with large uphill moves being less likely than small ones. The probability of large uphill moves is gradually reduced as the search progresses.

A variety of local handlers have been designed for use in the local search algorithms. *SAT1* uses a local handler that may randomly negate the truth values of one or up to n variables [212, 226, 214, 217]. The basic idea is to generate *random exchanges* in some current solution points when the search is stuck at a local minimum. The search accepts a modified point as a new current solution *not only when the value of the objective function is better but also when it is worse* [226, 212, 214, 217]. This differs from the traditional local search such as *GSAT* that used greedy local descent [482].

Parallel Local Search. Several parallel algorithms and VLSI architectures have been developed to accelerate CSP and SAT computations [232, 233, 217, 502]. Depending on implementations, there are several ways of grouping variables or clauses together in parallel so they can be evaluated simultaneously. In the *SAT1* algorithms, the most frequently used part of computation is the function *evaluate_objective_function()*. It takes $O(ml)$ time to update the objective function. The execution of *evaluate_objective_function* can be done in a simple bit-parallel manner in $O(m)$ time on a sequential computer.

A computer word has 32 or 64 bits. The number of literals in a clause of most practical *CNF* formulas is much less than 32. In a local search al-

gorithm, therefore, one can pack all the literals in a clause into the bits of a computer word and then evaluate all the literals in one clause in parallel. For m clauses, instead of $O(ml)$, it will take procedure *evaluate-objective-function* $O(m)$ time to evaluate and update the objective function. Occasionally, a clause may have more than 32 literals which can be packed in several computer words and be evaluated simultaneously. This general bit-parallel evaluation method was implemented in the *SAT1.7* algorithm [212, 217].

Complete Local Search. Local search algorithms are incomplete, *i.e.*, they can find some solutions for certain *CNF* formulas and give no answer if the *CNF* formula is not satisfiable. To overcome this problem, researchers developed complete local search algorithms to test satisfiability as well as unsatisfiability. The basic idea in the *SAT1.11* and *SAT1.13* algorithms [212, 217] was to combine local search with a systematic search procedure, keeping local search's efficiency while maintaining search completeness by a systematic search method. If at a node of the search tree a solution point is found unsatisfiable, then the algorithm backtracks and continues searching until a solution is found or unsatisfiability is proven.

The *SAT1.11* and *SAT1.13* algorithms were two early experiments of complete local search algorithms. Probe order backtracking is a simplified version of complete local search [443, 444]. Recently Crawford studied a complete local search algorithm [111]. He used weights assigned to clauses to help choose branch variables. Variables occurring in heavily weighted clauses were given precedence.

6.6 Randomized Local Search with Trap Handling

For the SAT problem, with high probability, a greedy local search will fall into a trap much more easily. In this case some variables are updated very quickly. The related clauses oscillate between the *sat* and *unsat* states. The search is limited to these trapping states. Without any help, there is little chance of getting out to explore other solution states.

Based on early observation of trap phenomenon and the development of a three-level search space model (Section 6.3), Gu *et al.* developed a *SAT1.5* algorithm with trap handling ability [214, 224]. The *SAT1.5* algorithm can monitor and break local minimum loops and can handle multiple traps during the search. The current version of the *SAT1.5* algorithm contains advanced data structures and complicated trap detection/handling methods

[214, 217, 224]. For the sake of simplicity, Figure 16 gives a brief outline of the algorithm.

The *SAT1.5* starts with an initial random solution and a set of limiting parameters. Parameter *Max-Time* specifies the maximum number of times allowed to restart a new search. The number of unsatisfiable clauses is computed and is assigned as the value of the objective function. The first *while* loop is limited by the *Max-Time*. Procedure *complete_flip()* flips all the variables that can reduce the value of the objective function. *Evaluate-objective-function()* updates the objective function.

The second *while* loop is a randomized local search with trap tracking and handling. Trap detection facilities are installed several places in the *while* loop to record trap statistics. They are essential to figure out trap “height,” “width,” and other parameters for subsequent decision making. A trap may contain a global minimum solution and it must be searched with reasonable effort. Leaving a trap too early or too late could result in either losing solutions or wasting computing time. The time to jump out of a trap is determined by parameter *Max-Trapping-Times*.

When the search algorithm jumps out of a trap, there are several alternatives to pursue. One is to start a new search. In the *while* loop, several randomized local search procedures deploying random value and random variable heuristics are grouped together with partial random selection heuristics (see Section 6.4). They together select a variable for randomized local search (the objective function *F* may increase during the search).

If a trap is detected, a number of strategies can be used to perform a trap search [217, 224]. In one approach proposed by Gu *et al.*, a sequence of random flip operations is performed (see Figure 16). The intensity of the flip operations evolves from weak to strong, according to the trap structure and search times. That is, a variable in each *unsat clause* is flipped to a *true* value (procedure *weak_flip()*), followed by a random flip of a few percent of the variables (procedure *gentle_flip()*). And finally, if the *Trapping-Times* is sufficiently large, procedure *strong_flip()* would flip a small set of variables randomly. Additional facilities for hill climbing, tabu search, and variable locking/unlocking were developed. The *SAT1.5* algorithm can walk on the rugged surface of a trap adaptively.

The real execution performance of the *SAT1.5* algorithm (Section 13) suggests that it is presently one of the fastest local search algorithms for the SAT problem.

```

procedure SAT1.5 ()
begin
    /* initialization */
    get_a_SAT_instance();
     $\mathbf{x}_0 := \text{select\_a\_random\_initial\_point}();$ 
     $F(\mathbf{x}_0) := \text{evaluate\_objective\_function}(\mathbf{x}_0);$ 
    /* search */
     $k := 0; \text{Restart\_Times} := 0;$ 
    while  $F > 0$  and  $\text{Restart\_Times} < \text{Max\_Time}$  do
        begin
             $\mathbf{x}_{k+1} := \text{complete\_flip}(\mathbf{x}_k);$ 
             $F := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
            clean_trap_records();  $\text{Trapping\_Times} := 0;$ 
            while  $F > 0$  and  $\text{Trapping\_Times} < \text{Max\_Trapping\_Times}$  do
                begin
                     $x_i := \text{select\_one\_var\_to\_flip}(\mathbf{x}_{k+1});$ 
                     $\mathbf{x}_{k+1} := \text{randomized\_local\_search}(x_i, \bar{x}_i);$ 
                     $F := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
                    if a trap is detected then begin
                         $\text{Trapping\_Times} +=;$ 
                        /* random flip vars in conflicting clauses */
                         $\mathbf{x}_{k+1} := \text{weak\_flip}(\mathbf{x}_{k+1});$ 
                         $F := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
                        /* random flip a few percent (pct) of variables */
                         $\mathbf{x}_{k+1} := \text{gentle\_flip}(\mathbf{x}_{k+1}, \text{pct});$ 
                         $F := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
                        /* random flip a small set of variables */
                        if  $\text{Trapping\_Times}$  is large then
                             $\mathbf{x}_{k+1} := \text{strong\_flip}(\mathbf{x}_{k+1}, \text{set});$ 
                             $F := \text{evaluate\_objective\_function}(\mathbf{x}_{k+1});$ 
                        /* initialization for a new trap */
                        clean_trap_records(); end; end;
                    if  $F > 0$  then
                         $\mathbf{x}_{k+1} := \text{restart\_a\_new\_random\_point}();$ 
                         $\text{Restart\_Times} +=;$ 
                         $k := k + 1;$ 
                    end;
                end;
            end;

```

Figure 16: **SAT1.5:** a randomized local search procedure with trap handling ability.

6.7 Greedy Local Search

Traditional local search proceeds by taking a feasible solution point that reduces the value of the objective function. Among many neighboring solution points, local search does not evaluate neighbors' relative performance with respect to the objective function. A *greedy* algorithm selects the *best neighbor* that yields the minimum value of the objective function and takes this *best neighbor direction* as the descent direction of the objective function.

Selman *et al.* proposed a *greedy* local search procedure, i.e., *GSAT*, for the SAT problem [482]. During each search step, the algorithm evaluates

```

procedure GSAT ()
begin
    for  $i := 1$  to MAX-TRIES
         $T :=$  a randomly generated truth assignment
        for  $j := 1$  to MAX-FLIPS
            if  $T$  satisfies  $\alpha$  then return  $T$ 
             $p :=$  a propositional variable such that a change
                in its truth assignment gives the largest
                increase in the total number of clauses
                of  $\alpha$  that are satisfied by  $T$ 
             $T := T$  with the truth assignment of  $p$  reversed
        end for
    end for
    return "no satisfying assignment found"
end

```

Figure 17: **GSAT**: a Greedy local search procedure for the **SAT** problem [482]. **MAX-FLIPS**, **MAX-TRIES** are constants, and α is a set of clauses. During each search step, *GSAT* takes the best neighbor that gives the maximum descent to the objective function.

all the moves and selects the best move that gives the *greatest* decrease in the total number of unsatisfied clauses. If the algorithm becomes stuck at a local minimum, *GSAT* uses *side-walk* (a form of tunneling heuristic) to move aside. In *GSAT* procedure, two parameters, *MAX-TRIES* and *MAX-FLIPS*, were used to control the algorithm's maximum running state. Based on greedy local search, *GSAT* pursues only paths where every step leads to

an improvement, this leads to a procedure that becomes stuck much more often than the randomized local search.

Satisfiability problem is an important problem in VLSI circuit design. Late eighties VLSI researchers experimented a large number of practical SAT formulas and found that greedy local search became stuck much more easily at local minima than randomized local search procedures. Accordingly, Gu proposed a method of combining local descent with randomized multiphase search and trap handling heuristics (see Section 6.4 and Section 6.6). These ideas were used in the early *SAT1* algorithm family design [226, 212, 214, 217].

In order to overcome the weakness of greedy local search, recently Selman *et al.* combined bad variable and partial random variable selection heuristics (Section 6.4) in *random walk* for the *GSAT* algorithm [485, 484]. They found that these random heuristics (such as random flips, selecting a variable in unsat clause, and partial random variable selection) improved the performance of the *GSAT* algorithm significantly [485].

6.8 Tabu Local Search

Mazure, Sais, and Gregoire proposed a tabu search algorithm, *TSAT*, for satisfiability problem [379]. The basic idea behind the *TSAT* is to avoid using randomness in local search algorithm design. *TSAT* makes a systematic use of a tabu list of variables in order to avoid recurrent flips and thus escape from local minima. The tabu list is updated each time a flip is made. *TSAT* keeps a fixed length-chronologically-ordered FIFO list of flipped variables and prevents any of the variables in the list from being flipped again during a given amount of time.

In this study, Mazure *et al.* found that the optimal length of the tabu list is crucial to the algorithm's performance. They showed that, for random 3SAT instances, the optimal length of the tabu list $L(n)$ for *TSAT* is [379]:

$$\cdot \quad L(n) = 0.01875n + 2.8125. \quad (21)$$

Furthermore, they noted that a slight departure from the optimal length leads to a corresponding graceful degradation of the performance of *TSAT*. A more important distance from this optimal length leads to a dramatic performance degradation.

6.9 Local Search for *DNF* formulas

Using the well-known DeMorgan laws, we can obtain an unconstrained optimization model, the *SAT4 model*, for *DNF* formulas [212, 218]. With *SAT4*, a *CNF* formula

$$(x_1 + \bar{x}_2) (\bar{x}_1 + x_2 + \bar{x}_4) (x_2 + \bar{x}_3)$$

can be transformed into a *DNF* formula:

$$\bar{x}_1 x_2 + x_1 \bar{x}_2 x_4 + \bar{x}_2 x_3.$$

For the transformed formula, the objective is to determine whether there exists an assignment where all clauses are falsified. That is, to solve (11).

A number of local search algorithms were developed for *DNF* formulas. Except for different definition and evaluation schemes in the objective function, they have similar structures as in *CNF* local search algorithms. In *SAT1.4* [212], one of the early *DNF* local search algorithms, the objective function is defined as the number of satisfiable *DNF* terms. Our goal here is to reduce the objective function to zero. Experimental results indicate that *DNF* local search algorithms are faster than *CNF* local search algorithms.

7 Nonlinear Unconstrained Methods

In this section, we present a *UniSAT* model that transforms a SAT formula represented as an instance of a discrete constrained decision problem in Boolean $\{0, 1\}$ space into a nonlinear unconstrained optimization problem [212, 216, 218].

The concept of optimization is well rooted as a principle underlying the analysis of many complex decision problems. When one deals with a complex decision problem, involving the selection of values to a number of interrelated variables, one should focus on a single objective (or a few objectives) designed to qualify performance and measure the quality of the decision. The *core* of the optimization process is to minimize (or maximize) an objective function subject to constraints imposed upon values of decision variables in an instance.

Most optimization algorithms are designed as an iterative refinement process. Typically, in seeking a vector that solves an optimization problem, a search algorithm selects an initial vector \mathbf{y}_0 and generates an improved vector \mathbf{y}_1 . The process is repeated to find a better solution \mathbf{y}_2 . Continuing in this fashion, a sequence of ever-improving points $\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_k, \dots$, is found that approaches a solution point \mathbf{y}^* . When it is not possible to find neighboring points to improve, strategies are applied to help escape from local minima.

7.1 UniSAT: Universal SAT Input Models

In *UniSAT* models, we extend discrete search space $\mathbf{x} \in \{0, 1\}^n$ into real space $\mathbf{y} \in E^n$, so that each solution point and the objective function can be characterized quantitatively. Furthermore, by encoding the solution of a SAT formula into the objective function, a direct correspondence between the solutions of the SAT formula and the global minimum points of the objective function can be established. Subsequently, the SAT formula is transformed into an instance of an unconstrained nonlinear optimization problem on E^n .

In *UniSAT* models, using the universal DeMorgan laws, all Boolean \vee and \wedge connectives in *CNF* formulas are transformed into \times and $+$ of ordinary addition and multiplication operations, respectively. The *true* value of the *CNF* formula is converted to the 0 value of the objective function. Given a *CNF* formula \mathcal{F} from $\{0, 1\}^n$ to $\{0, 1\}$ with m clauses C_1, \dots, C_m , we define a real function $f(\mathbf{y})$ from E^n to E that transforms the SAT into an

unconstrained nonlinear optimization problem:

$$\min_{\mathbf{y} \in \mathbb{E}^n} f(\mathbf{y}) \quad (22)$$

where

$$f(\mathbf{y}) = \sum_{i=1}^m c_i(\mathbf{y}). \quad (23)$$

A clause function $c_i(\mathbf{y})$ is a product of n literal functions $q_{i,j}(y_j)$ ($1 \leq j \leq n$):

$$c_i = \prod_{j=1}^n q_{i,j}(y_j). \quad (24)$$

In the *UniSAT5* model [212, 214, 218]

$$q_{i,j}(y_j) = \begin{cases} |y_j - 1| & \text{if literal } x_j \text{ is in clause } C_i \\ |y_j + 1| & \text{if literal } \bar{x}_j \text{ is in clause } C_i \\ 1 & \text{if neither } x_j \text{ nor } \bar{x}_j \text{ is in } C_i \end{cases} \quad (25)$$

and in the *UniSAT7* model [212, 216, 214, 218]:

$$q_{i,j}(y_j) = \begin{cases} (y_j - 1)^2 & \text{if } x_j \text{ is in clause } C_i \\ (y_j + 1)^2 & \text{if } \bar{x}_j \text{ is in clause } C_i \\ 1 & \text{if neither } x_j \text{ nor } \bar{x}_j \text{ is in } C_i \end{cases} \quad (26)$$

The correspondence between \mathbf{x} and \mathbf{y} is defined as follows (for $1 \leq i \leq n$):

$$x_i = \begin{cases} 1 & \text{if } y_i = 1 \\ 0 & \text{if } y_i = -1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Clearly, \mathcal{F} has value *true* iff $f(\mathbf{y}) = 0$ on the corresponding $\mathbf{y} \in \{-1, 1\}^n$.

The *UniSAT5* model on real space is a direct extension of the discrete *SAT4* model on Boolean space. A model similar to *UniSAT5* was proposed independently in the neural network area [298]. A significant difference between the neural network model and *UniSAT5* is their efficiency and practical applicability. The neural network model can only be handled by traditional nonlinear programming methods that are extremely slow [298], whereas *UniSAT5* can be easily solved in conjunction with the local search approach by simple discrete accounting techniques [212, 218].

The *UniSAT* models transform SAT from a discrete, constrained decision problem into a nonlinear unconstrained optimization problem [212, 216, 214, 218]. A good property of the transformation is that *UniSAT* models establish a correspondence between the global minimum points of the objective function and the solutions of the original SAT formula. A CNF \mathcal{F} has value *true if and only if* $f(\mathbf{y})$ takes the global minimum value 0 on the corresponding solution \mathbf{y}^* .

Following the above formulation, with the *UniSAT5* and *UniSAT7* models, a CNF \mathcal{F}

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$

is translated into

$$f(\mathbf{y}) = |y_1 - 1||y_2 + 1| + |y_1 + 1||y_1 - 1||y_3 - 1|$$

and

$$f(\mathbf{y}) = (y_1 - 1)^2(y_2 + 1)^2 + (y_1 + 1)^2(y_1 - 1)^2(y_3 - 1)^2,$$

respectively.

The solution of the SAT formula corresponds to a set of global minimum points of the objective functions. Finding a *true* value of \mathcal{F} is equivalent to finding a *false* value, *i.e.*, 0, of $f(\mathbf{y})$.

The translation of SAT formulas into nonlinear programs is quite different from the integer programming approach described in the next section. In the integer programming approach, one views a SAT formula as an instance of the 0/1 Integer Programming problem and tries solving its linear programming relaxation [36, 265, 266, 292, 310, 308, 558]. If the solution is non-integer, one rounds off the values to the nearest integers and checks whether the solution corresponds to a solution of the original formula. If the rounded off values do not correspond to a solution, one computes another solution of the linear programming problem.

7.2 Nonlinear Unconstrained Method

Several families of nonlinear unconstrained optimization algorithms for the *UniSAT* problem have been developed [212, 216, 218]. *SAT6.0*, a basic nonlinear unconstrained optimization algorithm, is shown in Figure 18. To start, procedure *obtain_a_SAT_instance()* initializes a (given or generated) SAT instance. An objective function, f , is formulated according to a given *UniSAT* model. The SAT formula thus becomes a minimization problem to the objective function. To begin, procedure *select_an_initial_solution()* selects an initial starting point $\mathbf{y}_0 \in E^n$. The corresponding value of the

```

procedure SAT6.0 ()
begin
    /* initialization */
    obtain_a_SAT_instance();
     $y_0 := \text{select\_an\_initial\_solution}();$ 
     $f(y_0) := \text{evaluate\_object\_function}(y_0);$ 

    /* search */
     $k := 0;$ 
    while not(solution_testing()) do
        begin
            for some  $y_{i(k)} s \in y_k$ 
            begin
                /* minimizer */
                if test_min( $f(y_{i(k)} s)$ ) then
                    begin
                         $y_{k+1} := \text{perform\_min}(f(y_{i(k)} s));$ 
                         $f(y_{k+1}) := \text{evaluate\_object\_function}();$ 
                    end
                    if close_to_solution() then  $x := \text{approximate}(y_{k+1});$ 
                end;
                /* local handler */
                if local then local_handler();
                 $k := k + 1;$ 
            end;
        end;
    end;

```

Figure 18: **SAT6.0:** A nonlinear unconstrained optimization algorithm for SAT.

objective function, $f(\mathbf{y}_0)$, is evaluated by function *evaluate_object_function()*.

The optimization process is an iterative minimization to the objective function. Function *test_min()* tests if the value of the objective function can be minimized. If this is true, the minimization operation is performed by procedure *perform_min()*, followed by *evaluate_object_function()* that updates the value of the objective function. Procedures *test_min()*, *perform_min()*, and *evaluate_object_function()* are usually performed together without distinction. Depending on the optimization strategy, the objective function can be minimized in one or up to n dimensions. Methods capable of optimizing f in one dimension include line search, coordinate descent, and coordinate Newton's methods. Methods that optimize f in more than one dimensions include the steepest descent methods, multi-dimensional Newton's methods, and many others.

As the iterative improvement progresses, a global minimum point may be approached gradually. The *closeness* between the present solution point and the global minimum solution point can be tested by solution-point testing or objective-value testing. Procedure *close_to_solution()* performs closeness testing. If the present solution point is sufficiently close to a global minimum point, procedure *approximate()* performs the *round-off* operation that converts a solution point \mathbf{y} in real space E^n to a solution point \mathbf{x} in Boolean space $\{0, 1\}^n$ which may be a solution of the original SAT formula. Procedure *solution_testing()* takes the solution generated from procedure *approximate()* and substitutes it into the given *CNF* formula to verify its correctness.

In practice, the search process could be stuck at a locally optimum point. To improve the convergence performance of the algorithm, one or more local handlers may be added. One effective local handler in *SAT6* is to negate the truth values of up to n variables.

Any existing nonlinear unconstrained optimization method can be used to solve the *UniSAT* problems (see textbooks and literature). So far many unconstrained optimization algorithms have been developed [212, 216, 218]. These include the basic algorithms, steepest descent methods, modified steepest descent methods, Newton's methods, quasi-Newton methods, descent methods, cutting-plane methods, conjugate direction methods, ellipsoid methods, homotopy methods, and linear programming methods. In each algorithm family, different approaches and heuristics can be used to design objective functions, select initial points, scramble the search space, formulate higher-order local handlers, deflect descent directions, utilize parallelism, and implement hardware architectures to speed up computations.

```

Procedure SAT14.5 ()
begin
    /* initialization */
    y := initial_vector();
    local := search := 0; limit :=  $b n \log n$ ;

    /* search */
    while ( $f(\mathbf{y}) \geq 1$  and  $local \leq limit$ ) do
        begin
            old_f :=  $f(\mathbf{y})$ ; search := search + 1;
            /* minimizer */
            for  $i := 1$  to  $n$  do
                minimize  $f(\mathbf{y})$  with respect to  $y_i$ ;
            /* local handler */
            if ( $f(\mathbf{y}) = old\_f$  or ( $search > b' \log n$  and  $f(\mathbf{y}) \geq 1$ )) then
                begin
                    y := initial_vector();
                    search := 0; local := local + 1;
                end;
            end;
            if  $f(\mathbf{y}) < 1$  then y* := round_off(y) else y* := enumerate();
        end;
    
```

Figure 19: **SAT14.5:** An unconstrained optimization algorithm for the *UniSAT5* problem.

7.3 Discrete Nonlinear Unconstrained Optimization

Although nonlinear problems are intrinsically more difficult to solve, an unconstrained optimization problem is conceptually simple and easy to handle. Many powerful solution techniques have been developed to solve unconstrained optimization problems, which are based primarily upon calculus and simple accounting, rather than upon algebra and pivoting, as in the Simplex method. Based on a coordinate descent method [367], Gu has recently given a simple algorithm, the *SAT14.5* algorithm [218, 219], for the *UniSAT5* problem (see Figure 19). The kernel of *SAT14.5* is a discrete *minimizer* that minimizes objective function f by the discrete coordinate descent method.

Given a function f on E^n , the *SAT14.5* algorithm initially chooses a vector \mathbf{y} from E^n and then minimize function f with respect to variables y_j

($1 \leq j \leq n$) in *minimizer* until $f < 1$. Since each variable y_j appears in one clause function c_i at most once, function $f(\mathbf{y})$ can be expressed as

$$f(\mathbf{y}) = a_j|y_j - 1| + b_j|y_j + 1| + d_j$$

for ($1 \leq j \leq n$), where a_j , b_j , and d_j are *local gain factors* that are independent of y_j . They can be computed in $O(ln)$ time. Therefore, $f(\mathbf{y})$ takes its minimum value with respect to y_j at point either $y_j = 1$ or $y_j = -1$. Thus, the *minimizer* optimizes function f as follows: if $a_j \geq b_j$ then set y_j equal to 1; otherwise set y_j equal to -1 .

In practice, before $f < 1$, the algorithm could be stuck at a local minimum point. To overcome this problem, a simple local handler is added. The local handler simply generates a new initial vector \mathbf{y} to start an independent search. In the *SAT14.5* algorithm, if the objective function f can no longer be reduced *or* after $b' \log n$ (b' is a constant, see [218, 219]) iterations of the *while* loop f is still at least one, then the *local-handler* is called.

7.4 Continuous Nonlinear Unconstrained Optimization

Based on a continuous coordinate descent method [367], Gu, Huang and Du have recently developed the *SAT14.7* algorithm for solving *UniSAT7* problems on E^n [218]. For the objective function described in the *UniSAT7* input model, if only one variable, e.g., x_i , is selected for optimization, then

$$f(x_i) = a_i(x_i - 1)^2 + b_i(x_i + 1)^2 + c_i \quad (27)$$

where a_i , b_i , and c_i are constants that can be computed in $O(ml)$ time. Here, $F(x_i)$ can be minimized at:

$$x_i = \frac{a_i - b_i}{a_i + b_i}. \quad (28)$$

7.5 Complete Nonlinear Unconstrained Optimization

The *SAT14.5*, *SAT14.6*, and *SAT14.7* algorithms are *incomplete* algorithms. In order to achieve high computing efficiency and to make them complete algorithms, in *SAT14.11* to *SAT14.20* algorithms we combine nonlinear optimization with backtracking/resolution procedures [212, 218]. Therefore, these algorithms are able to verify satisfiability as well as unsatisfiability. Figure 20 gives a typical backtracking nonlinear optimization algorithm.

```

Procedure SAT14.11 ()
begin
    /* initialization */
    get_a_SAT_instance();
     $x_0 := \text{select\_an\_initial\_point}();$ 
     $f := \text{evaluate\_object\_function}(x_0);$ 

    /* backtracking with nonlinear optimization */
     $x^* := \text{backtracking}(x_0);$ 
end;

Procedure backtracking( $x_i$ )
begin
    /* nonlinear optimization assigns  $v$  to  $x_i$  */
     $v := \text{nonlinear\_optimization}();$ 
     $x_i := v;$ 
     $V_i := V_i - \{v\};$ 
    /* append variable  $x_i$  to the partial path */
    path[ $x_i$ ] :=  $i$ ;

    if path broken then backtracking;
    if solution found then return  $x^*$ ;
    else backtracking(next  $x_i$ );
end;

```

Figure 20: **SAT14.11:** a complete nonlinear optimization algorithm with backtracking.

For small and medium size problems, backtracking is able to verify unsatisfiability quickly for certain classes of formulas but is slow when it comes to verifying satisfiability, as all possible resolutions need to be tried out before concluding that the inference relation holds or that the input formula is satisfiable. From our experience, a *combined* nonlinear optimization algorithm with backtracking/resolution procedures would perform well for certain classes of satisfiable and unsatisfiable formulas.

Recently some researchers investigated the number of solutions of SAT formulas. Extending Iwama's work [288], Dubois gave a combinatorial formula computing the number of solutions of a set of any clauses [149]. He and Carlier also studied the mathematical expectation of the number of solutions for a probabilistic model [150]. For an incomplete SAT algorithm, the number of solutions can have a strong effect on its computing efficiency. For a complete SAT algorithm, however, the number of search levels plays a crucial role. In *SAT14.11* to *SAT14.20* algorithms, the number of solutions is an important strategy to interplay nonlinear optimization and backtracking/resolution procedures [217, 218].

7.6 Convergence Property and Average Time Complexity

Gu, Gu and Du [223] have analyzed the convergence ratios of three basic methods: the steepest descent method, Newton's method, and the coordinate descent method for objective function f defined in the *UniSAT7* input model. They prove that, subject to certain conditions [367], the steepest descent method has a linear convergence ratio $[(A-a)/(A+a)]^2 < 1$, Newton's method has a convergence ratio of order two, and the coordinate descent method has a convergence ratio $(1 - \frac{a}{A(n-1)}) < 1$, where $A \geq a > 0$ are the largest and smallest eigenvalues of the Hessian matrix $\mathbf{H}(\mathbf{y})$, respectively.

From these convergence properties, Gu, Gu, and Du roughly estimate that, subject to certain conditions [367], the *UniSAT7* problem can be solved in $O(\log(n+m))$ iterations by the steepest descent method and can be solved in $O(m \log(n + m))$ iterations by the coordinate descent method, on the assumption that the algorithm is not stuck at a local minimum point.

Gu and Gu have made some preliminary analysis of the typical time complexity of some unconstrained optimization SAT algorithms [219]. It shows that, the *SAT14.5* algorithm, with probability at least $1 - e^{-n}$, finds a solution within $k = O(n(\log n)^2)$ iterations of the *while* loop for a randomly generated satisfiable *CNF* formula with $l \geq 3$ and $m/n \leq \alpha 2^l/l$, where $\alpha < l$ is a constant. From this and the fact that the run time of procedure

enumerate() is $O(2^n)$, the typical time complexity of the *SAT14.5* algorithm is

$$(1 - e^{-n})O(n(\log n)^2(lmn)) + e^{-n}O(2^n) = O(\ln(n \log n)^2).$$

Clearly, the *SAT14.5* algorithm can give an answer to an unsatisfiable *CNF* formula in $O(2^n)$ time.

8 Nonlinear Constrained Method

In this section, we formulate the SAT problem as a constrained nonlinear optimization problem as shown in (12) and (18). We will show two Lagrangian formulations of SAT problems, one in the continuous space and the other in the discrete space.

8.1 Continuous Lagrangian Search Algorithms

As indicated in (18), a SAT problem can first be transformed into a continuous constrained optimization problem.

$$\begin{aligned} \min_{\mathbf{y} \in E^n} \quad & F(\mathbf{y}) = \sum_{i=1}^m c_i(\mathbf{y}) \\ \text{subject to} \quad & c_i(\mathbf{y}) = 0 \quad \forall i \in \{1, 2, \dots, m\} \end{aligned} \tag{29}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)$, and $c_i(\mathbf{y})$ is defined in (14) and (15) and repeated as follows.

$$\begin{aligned} c_i(\mathbf{y}) &= \prod_{j=1}^n q_{i,j}(y_j) \\ q_{i,j}(y_j) &= \begin{cases} (1 - y_j)^2 & \text{if } x_j \text{ in } C_i \\ y_j^2 & \text{if } \bar{x}_j \text{ in } C_i \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

Here, $F(\mathbf{y})$ is a scalar differentiable function that takes the norm of its argument so that $F(\mathbf{y}) = 0$ iff $c_i(\mathbf{y}) = 0$ for all i .

There are three advantages in reformulating the original discrete unconstrained problem into a continuous constrained problem. First, a continuous objective function may smooth out local minima in the discrete space, allowing search methods to bypass these local minima in the continuous space. Second, a continuous objective value can indicate how close the constraints

are being satisfied, hence providing additional guidance in leading to a satisfiable assignment. Third, when the search is stuck in a local minimum and some of the constraints are violated, the violated constraints can provide a force to lead the search out of the local minimum. This is more effective than restarting from a new starting point, as local information observed during the search can be preserved.

In *Lagrangian methods*, Lagrange multipliers are introduced to gradually resolve constraints through iterative updates. They are exact methods that optimize the objective using Lagrange multipliers to meet the Kuhn-Tucker conditions [367]. Eq. (29) can be reformulated using Lagrange multipliers into the following unconstrained problem:

$$L(\mathbf{y}, \lambda) = F(\mathbf{y}) + \lambda^T \mathbf{c}(\mathbf{y}) \quad (\text{Lagrangian function}) \quad (30)$$

$$\mathcal{L}(\mathbf{y}, \lambda) = F(\mathbf{y}) + \|\mathbf{c}(\mathbf{y})\|_2^2 + \lambda^T \mathbf{c}(\mathbf{y}) \quad (\text{Augmented Lagrangian}) \quad (31)$$

where $\mathbf{c} = (c_1(y), c_2(y), \dots, c_m(y))$, and λ^T is the transpose of the set of Lagrange multipliers. The augmented Lagrangian formulation is often preferred because it provides better numerical stability.

According to classical optimization theory [367], all the extrema of (31), whether local or global, are roots of the following sets of equations.

$$\nabla_{\mathbf{y}} \mathcal{L}(\mathbf{y}, \lambda) = 0 \quad \text{and} \quad \nabla_{\lambda} \mathcal{L}(\mathbf{y}, \lambda) = 0 \quad (32)$$

These conditions are necessary to guarantee the (local) optimality to the solution of (29).

Search methods for solving (31) can be classified into local and global algorithms. Local minimization algorithms, such as gradient-descent and Newton's methods, find local minima efficiently and work best in uni-modal problems. Global methods, in contrast, employ heuristic strategies to look for global minima and do not stop after finding a local minimum [416, 518, 367]. Note that gradients and Hessians can be used in both local and global methods [518].

Local search methods can be used to solve (32) by forming a Lagrangian dynamic system that includes a set of dynamic equations to seek equilibrium points along a gradient path. These equilibrium points correspond to the constrained minima of (29). The Lagrangian dynamic system of equations are as follows.

$$\frac{d\mathbf{y}(t)}{dt} = -\nabla_{\mathbf{y}} \mathcal{L}(\mathbf{y}(t), \lambda(t)) \quad \text{and} \quad \frac{d\lambda(t)}{dt} = \nabla_{\lambda} \mathcal{L}(\mathbf{y}(t), \lambda(t)) \quad (33)$$

Locally optimal solutions to the continuous formulation are governed by the Saddle Point Theorem which states that \mathbf{y}^* is a local minimum to the original problem defined in (29) if there exists λ^* such that $(\mathbf{y}^*, \lambda^*)$ constitutes a saddle point of the associated Lagrangian function $F(\mathbf{y}, \lambda)$. Note that the converse of the above condition is not always true. Here, a *saddle-point* $(\mathbf{y}^*, \lambda^*)$ of Lagrangian function $F(\mathbf{y}, \lambda)$ is defined as one that satisfies the following condition.

$$F(\mathbf{y}^*, \lambda) \leq F(\mathbf{y}^*, \lambda^*) \leq F(\mathbf{y}, \lambda^*) \quad (34)$$

for all (\mathbf{y}^*, λ) and all (\mathbf{y}, λ^*) sufficiently close to $(\mathbf{y}^*, \lambda^*)$.

There are four advantages in using a Lagrangian formulation to solve constrained optimization problems.

- Equilibrium points of (32) can be found by local gradient descent and ascent methods defined in (33). The first equation in (33) has a minus sign that optimizes the original variables along a descending path, whereas the second equation optimizes along an ascending path. Alternatively, (33) can be considered as a search algorithm based on a descent algorithm in the original variable space. When the search reaches a local minimum that does not satisfy all the constraints, the search is brought out of the local minimum using the weights imposed by its Lagrange multipliers. This mechanism allows the search to continue in its present trajectory without any breaks.
- Lagrangian search is similar to penalty-based methods in the sense that the Lagrange multipliers are increased like penalties when constraints are violated. However, it is more general than penalty-based methods because the increase of a Lagrange multiplier is self-adjusting and is governed by the amount that the corresponding constraint is violated.
- The search modeled by (33) can be started from any starting point and will continue until an equilibrium point is found.
- Since assignments of \mathbf{y} where the constraints in (29) are satisfied are also assignments that minimize the objective, equilibrium points of (32) found by solving (33) correspond to satisfiable assignments to the original SAT problem.

It is important to note out that a Lagrangian search modeled by (33) is incomplete: if it does not find a solution in a finite amount of time, it does

not prove whether the original SAT problem is satisfiable or not. Hence, infinite time will be required to prove unsatisfiability.

Unfortunately, continuous gradient-based local search methods for solving (33) are very time consuming. Our experience [82] indicates that continuous descent methods are several orders of magnitude more complex than discrete descent methods. For instance, it takes over one hour of CPU time on a Sun SS10/51 workstation to solve a problem with 200 variables and 60 constraints. Consequently, continuous formulations are not promising in solving large SAT problems. In the next subsection, we extend continuous Lagrangian methods to discrete Lagrangian methods. Surprisingly, discrete methods work much better and can solve some benchmark problems that cannot be solved by other local/global search algorithms.

8.2 Discrete Lagrangian Search Algorithms

To overcome the computational complexity of continuous Lagrangian methods while preserving their benefits, we show a discrete constrained formulation of a SAT problem and its solution using a discrete Lagrangian method. The discrete Lagrangian method is extended from the theory of continuous Lagrangian methods.

Recall (12) in Section 3.2 the following discrete constrained formulation of a SAT problem.

$$\begin{aligned} \min_{\mathbf{y} \in \{0,1\}^n} \quad & N(\mathbf{y}) = \sum_{i=1}^m U_i(\mathbf{y}) \\ \text{subject to} \quad & U_i(\mathbf{y}) = 0 \quad \forall i \in \{1, 2, \dots, m\}. \end{aligned} \tag{35}$$

Without showing all the details [548], the continuous Lagrangian method can be extended to work on discrete problems. The *discrete Lagrangian function* for (35) is defined as follows.

$$L(\mathbf{y}, \lambda) = N(\mathbf{y}) + \lambda^T U(\mathbf{y}) \tag{36}$$

where $\mathbf{y} \in \{0,1\}^n$, $U(\mathbf{y}) = (U_1(\mathbf{y}), \dots, U_m(\mathbf{y})) \in \{0,1\}^m$, and λ^T is the transpose of $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_m)$ that denotes the Lagrange multipliers. (Note that λ_i can be continuous variables.)

In a definition similar to that in (34), a saddle point $(\mathbf{y}^*, \lambda^*)$ of $L(\mathbf{y}, \lambda)$ in (36) is defined as one that satisfies the following condition.

$$L(\mathbf{y}^*, \lambda) \leq L(\mathbf{y}^*, \lambda^*) \leq L(\mathbf{y}, \lambda^*) \tag{37}$$

1. Set initial x randomly by a fixed random seed
2. Set initial λ to be zero
3. **while** x is not a solution, i.e., $N(x) > 0$
4. update x : $x \leftarrow x - \Delta_x L(x, \lambda)$
5. **if** condition for updating λ is satisfied **then**
6. update λ : $\lambda \leftarrow \lambda + c \times U(x)$
7. **end if**
8. **end while**

Figure 21: Generic discrete Lagrangian algorithm \mathcal{A} for solving SAT problems.

for all λ sufficiently close to λ^* and for all \mathbf{y} whose Hamming distance between \mathbf{y}^* and \mathbf{y} is 1.

It can be proved that, in discrete space, the set of all saddle points is equal to the set of all solutions that satisfy the following two *first-order necessary conditions for discrete problems*:

$$\Delta_{\mathbf{y}} L(\mathbf{y}, \lambda) = 0 \quad (38)$$

$$\nabla_{\lambda} L(\mathbf{y}, \lambda) = 0, \quad (39)$$

where $\Delta_{\mathbf{y}} L(\mathbf{y}, \lambda)$ is the *discrete gradient* of $L(\mathbf{y}, \lambda)$ at point \mathbf{y} for fixed λ . Informally, $\Delta_{\mathbf{y}} L(\mathbf{y}, \lambda)$ is a vector that points from \mathbf{y} to a neighborhood point of $\mathbf{y} \in \mathcal{N}(\mathbf{y}) \cup \{\mathbf{y}\}$ with the minimum L value, for a user-defined set $\mathcal{N}(\mathbf{y})$ of neighborhood points.

Similar to (33), the *Discrete Lagrangian Method* (DLM) for solving SAT problems can be defined as a set of difference and vector equations,

$$\mathbf{y}^{k+1} = \mathbf{y}^k \oplus \Delta_{\mathbf{y}} L(\mathbf{y}^k, \lambda^k) \quad (40)$$

$$\lambda^{k+1} = \lambda^k + U(\mathbf{y}^k), \quad (41)$$

where \oplus is the vector-add operator ($x \oplus y = (x_1 + y_1, \dots, x_n + y_n)$).

8.3 An Implementation of a Discrete Lagrangian Algorithm

Figure 21 shows the pseudo code of \mathcal{A} , a generic discrete Lagrangian algorithm implementing (40) and (41). It performs descents in the original variable space of \mathbf{y} and ascents in the Lagrange-multiplier space of λ . In discrete space, $\Delta_{\mathbf{y}} L(\mathbf{y}, \lambda)$ is used in place of the gradient function in continuous space. We call one *iteration* as one pass through the while loop.

In the following, we describe some of the considerations in implementing DLM \mathcal{A} .

(a) *Initial Points and Restarts* (Lines 1-2). DLM is started from either the origin or from a random initial point generated by calling *drand48()* using a fixed random seed 101. Further, λ is always set to zero. The fixed initial points allow the results to be reproduced easily.

(b) *Descent and Ascent Strategies* (Line 4). There are two ways to calculate $\Delta_y L(\mathbf{y}, \lambda)$: greedy and hill-climbing, each involving a search in the range of Hamming distance one from the current \mathbf{y} (assignments with one variable flipped from the current assignment \mathbf{y}).

In a *greedy strategy*, the assignment leading to the maximum decrease in the Lagrangian-function value is selected to update the current assignment. Therefore, all assignments in the vicinity need to be searched every time, leading to computation complexity of $O(m)$, where m is the number of variables in the SAT problem. In *hill-climbing*, the first assignment leading to a decrease in the Lagrangian-function value is selected to update the current assignment. Depending on the order of search and the number of assignments that can be improved, hill-climbing strategies are generally less computationally expensive than greedy strategies.

A comparison of the two strategies show that hill-climbing is orders of magnitude faster and results in solutions of comparable quality.

(c) *Conditions for updating λ* (Line 5). The frequency in which λ is updated affects the performance of a search. The considerations here are different from those of continuous problems. In a discrete problem, descents based on discrete gradients usually make small changes in $L(\mathbf{y}, \lambda)$ in each update of \mathbf{y} because only one variable changes. Hence, λ should not be updated in each iteration of the search to avoid biasing the search in the Lagrange-multiplier space of λ over the original variable space of \mathbf{y} .

Experimentally, λ should only be updated when $\Delta_x L(x, \lambda) = 0$. At this point, a local minimum in the original variable space is reached, and the search can only escape from it by updating λ . This strategy amounts to pure descents in the original \mathbf{y} variable space, while holding λ constant, until a local minimum is reached.

Note that this strategy is similar to Morris' “break out” strategy [398] and Selman and Kautz's *GSAT* [483, 484] that applies adaptive penalties to escape from local minima. One problem that is overlooked in these strategies is the growth of penalty terms. In solving a difficult SAT problem, penalty terms may grow to become very large as the search progresses, causing large swings in the objective function and delaying convergence of the search.

Solutions to this issue are discussed next.

(d) *Amount of update of λ* (Line 6). A parameter c controls the magnitude of changes in λ . In general, c can be a vector of real numbers, allowing non-uniform updates of λ across different dimensions and possibly across time. For simplicity, $c = 1$ has been found to work well for most of the benchmarks tested. However, for some larger and more difficult problems, a smaller c can result in shorter search time.

The update rule in Line 6 results in nondecreasing λ . This is true because $U(x)$ is either 0 or 1: when a clause is not satisfied, its corresponding λ is increased; and when a clause is satisfied, its corresponding λ is not changed. In contrast, in applying Lagrangian methods to solve continuous problems with equality constraints, the Lagrange multiplier λ_i of constraint $g_i(x) = 0$ increases when $g_i(x) > 0$ and decreases when $g_i(x) < 0$.

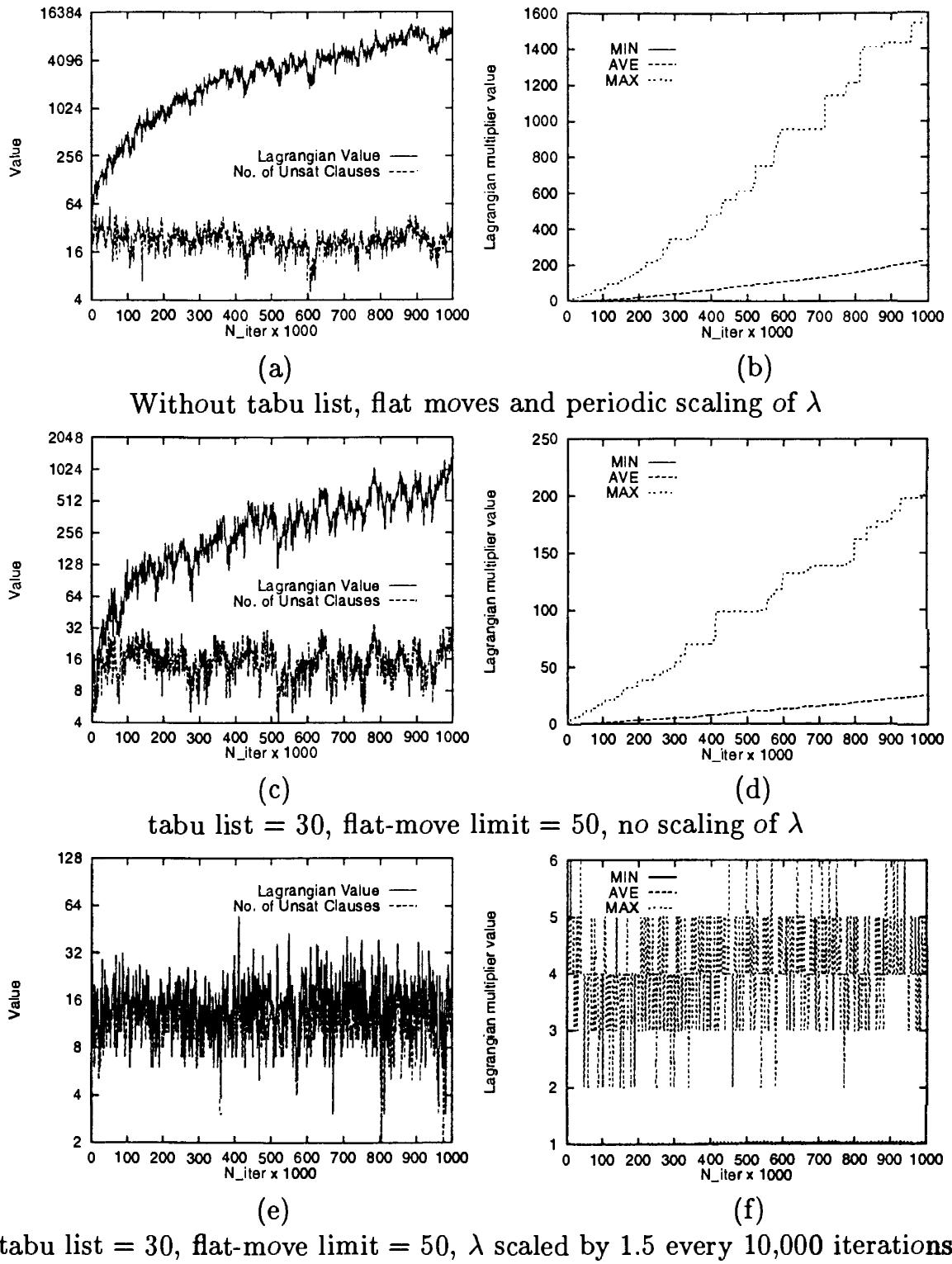
When there is no mechanism to reduce the Lagrange multipliers, they can grow without bound, causing large swings in the Lagrangian-function value and making the search terrain more rugged. Although this strategy does not worsen the search time for most of the benchmark problems tested, λ values can become very large as time goes on for a few difficult problems requiring millions of iterations. When this happens, the search has difficulty in identifying an appropriate direction to move.

This situation is illustrated in the first two graphs of Figure 22 that show the behavior of DLM when it was applied to solve one of the more difficult DIMACS SAT benchmark problems. Here, the search is stuck in a sub-optimal basin in the space of the objective function where the number of unsatisfied clauses fluctuates around 20. Since the search terrain modeled by L becomes more rugged as the Lagrange multipliers increase, the search will have difficulty to escape from this region.

To overcome this problem, λ should be reduced periodically. For instance, in the last two graphs of Figure 22, λ was scaled down by a factor 1.5 every 10,000 iterations. This strategy, when combined with other strategies to be discussed next, restricts the growth of Lagrange multipliers, leading to the solution of some of the more difficult benchmark problems.

(e) *Plateaus in the Search Space*. In binary problems like SAT, a search may find a very small subset of variables that can lead to no degradation in the objective function. Flipping variables in this small subset successively may likely lead to a cycle in the search space. To avoid such an undesirable situation, variables that have been flipped in the recent past can be stored in a tabu list [202, 246] and will not be flipped until they are out of the list.

Further, for large SAT problems formulated as discrete optimization



tabu list = 30, flat-move limit = 50, λ scaled by 1.5 every 10,000 iterations

Figure 22: Execution profiles of DIMACS "g125-17" (a), (c), and (e) plot Lagrangian values and number of unsatisfied clauses. (b), (d), and (f) plot the min., avg., and max. Lagrange-multiplier values.

problems, the search may encounter large and flat, but suboptimal, basins. Here, gradients in all directions are the same and the search may wander forever. The discrete gradient operator $\Delta_y L(y, \lambda)$ may have difficulties in basins/plateaus because it only examines adjacent points of $L(y, \lambda)$ that differ in one dimension. Hence, it may not be able to distinguish a plateau from a local minimum.

One way to escape is to allow uphill moves. For instance, in *GSAT*'s random walk strategy [485], uphill walks are allowed based on probability p . However, the chance of getting a sequence of uphill moves to get out a deep basin is small since each walk is independent.

There are two effective strategies that allow a plateau to be searched.

(a) *Flat-move strategy.* We need to determine the time to change λ when the search reaches a plateau. As indicated earlier, updating λ when the search is in a plateau changes the surface of the plateau and may make it more difficult for the search to find a local minimum somewhere inside the plateau. To avoid this situation, a strategy called *flat move* [548] can be employed. This allows the search to continue for some time in the plateau without changing λ , so that the search can traverse states with the same Lagrangian-function value. How long should flat moves be allowed is heuristic and possibly problem dependent. Note that this strategy is similar to Selman's "sideway-move" strategy [484].

(b) *Tabu list.* This search strategy aims to avoid revisiting the same set of states in a plateau. In general, it is impractical to remember every state the search visits in a plateau due to the large storage and computational overheads. A tabu list [202, 246] can be kept to maintain the set of variables flipped in the recent past and to avoid flipping a variable if it is in the tabu list.

The last four graphs of Figure 22 illustrate the performance of DLM when the search maintains a tabu list of size 30, when it is allowed to stay in a basin within 50 moves (flat-move limit), and when all Lagrange multipliers are periodically scaled down. These graphs show significant reduction in the growth of Lagrangian-function values and Lagrange multipliers.

By using these strategies, DLM can solve successfully many of the hard problems in the DIMACS benchmark suite [548]. These results are presented in Section 13.

9 Integer Programming Method

In this section, we first give an integer program (IP) formulation of SAT. Then we describe some traditional techniques of using the integer programming approach to solve SAT.

9.1 An Integer Programming Formulation for SAT

In order to represent SAT inputs in the framework of mathematical programming, we identify logic value *true* with integer 1 and *false* with -1. Similar in *UniSAT* models (Section 7.1), all Boolean \vee and \wedge connectives are transformed into + and \times of ordinary addition and multiplication operations, respectively. Using a standard transformation, the i th clause C_i is transformed into a linear inequality [310, 308]:

$$\sum_{j=1}^n q_{i,j}(w_j) \geq 2 - |C_i| \quad (42)$$

where

$$q_{i,j}(w_j) = \begin{cases} w & \text{if literal } x_j \text{ is in clause } C_i \\ -w & \text{if literal } \bar{x}_j \text{ is in clause } C_i \\ 0 & \text{if neither } x_j \text{ nor } \bar{x}_j \text{ is in } C_i \end{cases} \quad (43)$$

where w_j is the j th integer variable.

To restrict $w_j = \pm 1$, $j = 1, 2, \dots, n$, requires that extra constraints be added to insure that each w_j be in the closed interval $[-1, 1]$, i.e., $-1 \leq w_j \leq 1$ for $j = 1, 2, \dots, n$.

Following the above formulation, for example, a CNF \mathcal{F}

$$(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3)$$

is translated into

$$w_1 - w_2 \geq 0$$

$$-w_1 + w_2 + w_3 \geq -1$$

$$w_2 + w_3 \geq 0$$

so an integer programming formulation is obtained for SAT as: finding $w_j = \pm 1$, such that

$$\begin{pmatrix} 1 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \geq \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} \quad (44)$$

or

$$\begin{pmatrix} -1 & 1 & 0 \\ 1 & -1 & -1 \\ 0 & -1 & -1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad (45)$$

While the Simplex method is effective for solving linear programs (LP), there is no single technique that is fast for solving integer programs. Therefore, approaches developed try to solve the integer program as an integer linear program (ILP).⁵ If the solution is non-integer, one rounds off the values to the nearest integers and checks whether this corresponds to a solution of the original input. If the rounded off values do not correspond to a solution, adds a new constraint and computes a solution of the modified linear program. So far most methods developed to solve the integer programs for SAT indirectly work on the corresponding integer linear programs.

Researchers have observed that the optimal integer-programming solution is usually not obtained by *rounding* the linear-programming solution although this is possible in certain cases (see Section 10). The closest point to the optimal linear-program may not even be feasible. In some cases, the nearest feasible integer point to the linear-program solution is far removed from the optimal integer point. Thus, when using an integer linear program to solve the integer program for SAT, it is not sufficient simply to round linear-programming solutions.

In the following sections, we describe existing integer programming methods to solve SAT.

9.2 Linear Program Relaxation

A basic method to solve an integer program is the linear program relaxation. In this approach, the LP relaxation is achieved by replacing $x_i \in \{0,1\}$ with $0 \leq x_i \leq 1$. The LP relaxation can be solved efficiently with some sophisticated implementations of Dantzig's Simplex method, such as MINOS [399], or some variations of Karmarkar's interior point method [312].

Hooker early reported that by solving a linear programming of SAT, one frequently produces an integer solution [266]. Kamath *et al.* used MINOS 5.1 to solve linear programming relaxation [310, 308]. They tried some small SAT inputs and found that the Simplex method failed to find integral

⁵An integer linear program (ILP) is a linear program further constrained by integrality restrictions.

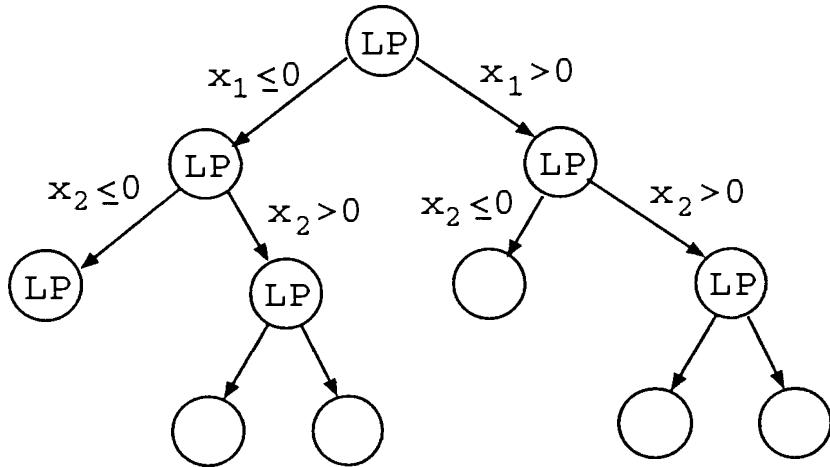


Figure 23: An example of a branch-and-bound tree.

solutions to the linear programming relaxations in majority of instances tested.

9.3 Branch and Bound Method

Branch-and-bound is essentially a strategy of “divide and conquer.” It is a straightforward and the most successful way to solve the integer programming problem. The idea is to systematically partition the linear-programming feasible region into manageable subdivisions and make assessments of the integer-programming problem based on these subdivisions. When moving from a region to one of its subdivisions, we add one constraint that is not satisfied by the optimal linear-programming solution over the parent region. So the linear programs corresponding to the subdivisions can be solved efficiently. In general, there are a number of ways to divide the feasible region, and as a consequence there are a number of branch-and-bound algorithms.

We show the basic procedures of branch-and-bound with a simple example shown in Figure 23. The method starts with the fractional solution given by its corresponding LP relaxation. Then a variable of fractional solution is selected. For example, let x_1 be a variable, and set $x_1 \leq 0$ as an additional constant; *i.e.*, branch on x_1 with constraint $x_1 \leq 0$. Resolve the LP relaxation with this augmented constraint. If it still produces a non-integer solution, branch on another non-integer variable, say x_2 , first with constraint $x_2 \leq 0$, and resolve the LP with extra constraint $x_1 \leq 0$ and $x_2 \leq 0$. This

process continues until solving the augmented LP yields an integer solution, *i.e.*, an incumbent solution, so there is no need to branch further at that node. Since we do not know this to be optimal, a backtracking procedure is required to search with extra constraints $x_1 \leq 0$ and $x_2 \geq 0$ and resolve the augmented LP and continue the process until an integer solution is obtained.

The above process produces a binary tree as shown in Figure 23. In this way, we implicitly exhaust all possibilities and conclude with an optimal solution. Note that each time we obtain an incumbent solution we get a new upper bound on the minimum value of the objective function. If at the same node the LP yields an objective function with value that exceeds the best upper bound obtained so far, then we can *fathom* that node, since any solution obtained at its successors can only be worse.

9.4 Cutting-Plane Method

Unlike partitioning the feasible region into subdivisions, as in branch-and-bound approaches, the cutting-plane algorithm solves integer programs by modifying linear-programming solutions until an integer solution is obtained. It works with a single linear program, which it refines by adding new constraints. The new constraints successively reduce the feasible region until an integer optimal solution is found.

The idea of the cutting plane method can be illustrated from a simple geometric interpretation (Figure 24). The feasible region for the integer program, *i.e.*, an integer polytope, consists of those integer lattice points satisfying all constraints. A *cut* is an inequality satisfied by all the feasible solutions of the integer program. A cutting plane is a hyperplane defined by that inequality and it conflicts with the solution X^* of the linear-programming relaxation. The cutting plane passes between X^* and the integer polytope and cuts off a part of the relaxed polytope containing the optimal linear-programming solution X^* without excluding any feasible integer points. After the cut, the resulting linear program is solved again. If the optimal values for the decision variables in the linear program are all integer, they are optimal; otherwise, a new cut is derived from the new optimal linear-programming tableau and appended to the constraints.

In practice, the branch-and-bound procedures almost always outperform the cutting-plane algorithm. Nevertheless, the algorithm has been important to the evolution of integer programming. Historically, it was the first algorithm developed for integer programming that could be proven to converge in a finite number of steps. In addition, even though the algorithm

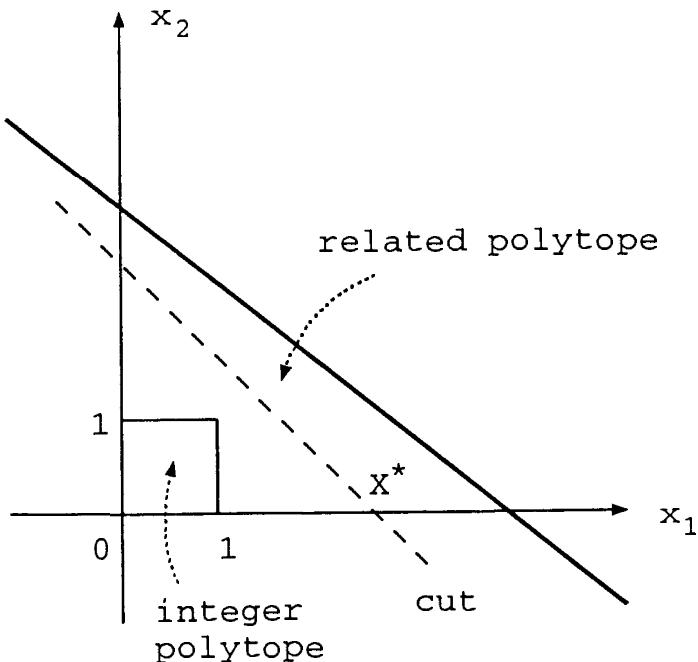


Figure 24: An illustration of the cutting-plane method.

generally is considered very inefficient, it has provided insights into integer programming that have led to other, more efficient algorithms.

9.5 Interior Point Method

The most important advance in linear programming solution techniques was recently introduced by Karmarkar [312]. As shown in Figure 25, compared to the Simplex method which jumps from a corner point to another corner point of the LP polytope until the optimal solution is found, Karmarkar's algorithm constructs an ellipsoid inside the polytope and uses nonlinear transformations to project better solution guesses in the interior of the polytope. Unlike the Simplex method which approaches the optimal solution indeed by step-by-step searching and has an exponential worst-case complexity, Karmarkar's algorithm has been proven to be a polynomial time algorithm.

To apply Karmarkar's algorithm on integer programming, first the 0/1 integer program is transformed to a ± 1 integer program. Then the potential function x^2 is used, and obviously the optimal integer solution to the original IP problem is at the point that the potential function achieves a maximum. However, using Karmarkar's algorithm on integer programming may get

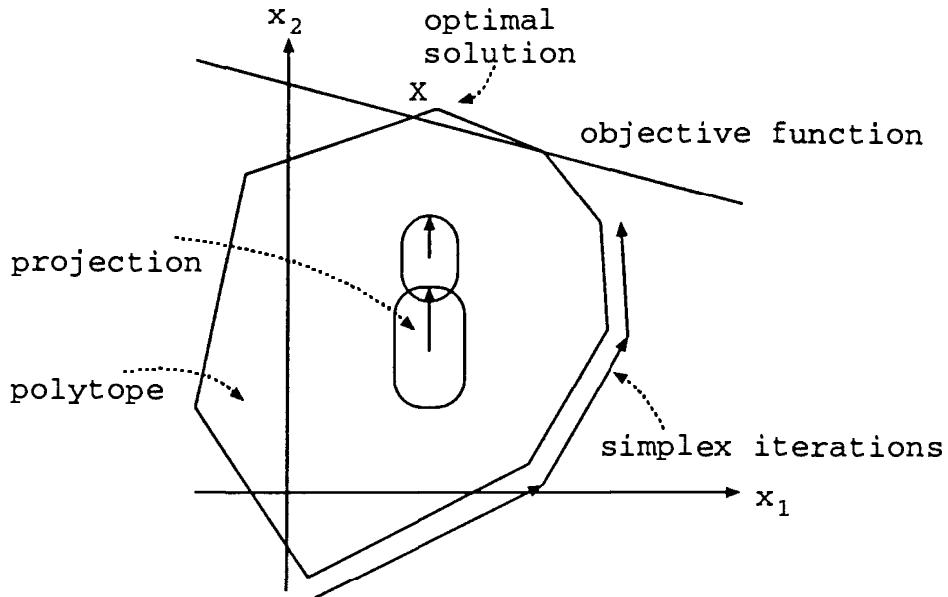


Figure 25: The basic ideas of the Simplex method and Karmarkar's method.

stuck at a local minimum, *i.e.*, it does not guarantee to find the optimal solution by projection. Therefore, it is an incomplete algorithm.

9.6 Improved Interior Point Method

It is expected that a sequence of interior points

$$w^{k+1} = w^k + \alpha \Delta w^* \quad (46)$$

is generated such that the potential function in Karmarkar's algorithm is minimized. It is crucial to determine the descent direction Δw^* of the potential function around w^k and the step size α .

In the original Karmarkar's algorithm, the step size α is assumed with $(0,1]$. They used $\alpha = 0.5$ in their experiments to solve SAT inputs. If the potential function is well represented by the quadratic approximation around the given point, then if we move along the Newton direction and have the appropriate values for certain parameters, we will reach the minimum; otherwise, recall that the step size is chosen so that it reaches a minimum of the objective function on that line of the given descent direction. So there is no reason to restrict α within $(0,1]$.

This suggests the necessity to use line search to choose optimal step size. Following this idea, Shi, Vannelli, and Vlach have recently given an

improved interior point algorithm [489]. In their algorithm, the step size α is determined by a golden-section search [367]. Experiments show significant improvements on Karmarkar's algorithm.

10 Special Classes of SAT

Certain classes of CNF formulas are known to be solved in polynomial time by special purpose algorithms. There are at least three reasons for discussing such classes in this section. First, a given formula can be preprocessed and examined to determine whether it is a member of a polynomial time solvable class. If so, a special purpose, fast algorithm can be brought to bear on the formula. Second, a portion of a given formula may be a member of such a class and its solution may make solving the formula easier. Third, study of such classes reveals, in part, the nature of "easy" SAT formulas. On the other hand, as reported in Section 12, studies of random formulas indicate that these known classes contain only a small fraction of the formulas that can be solved rapidly by known methods with high probability.

Below, we mention some of the more notable polynomial time solvable classes. Often, efficient algorithms for solving formulas of such classes iteratively assign values to unassigned variables, removing newly satisfied clauses and falsified literals at each step, until either all clauses are satisfied or a substructure that is impossible to satisfy is revealed.

A crucial component of many of these algorithms is *unit resolution*. At any iteration, one or more clauses that have not been satisfied may contain exactly one unassigned variable. Such a clause is regarded as a *unit clause*. Call the set of unit clauses at the current iteration the *current unit set*. Assuming no clause has yet been falsified, one of the unassigned variables from the current unit set can be assigned a value that causes a clause containing it to take the value *true*. This is what we call an application of the *unit clause rule* or *unit resolution*.

Express a clause as a set of literals, a formula as a set of clauses, and a truth assignment T as a set of mutually non-complementary literals that each have value *true* (variables associated with literals not specified in T take either value *false* or are unassigned). Let $\text{complement}(x)$ be the literal that is the complement of literal x . The following algorithm **UR** applies unit resolution to a given formula \mathcal{F} . It returns a partial truth assignment

T and another formula with no unit clauses.

UR(\mathcal{F}, T):

```
/* Applies unit resolution to CNF formula  $\mathcal{F}$  until exhaustion */
/*  $T$  is passed by reference */
/* Returns a CNF formula */

Repeat the following
    Let  $\{x\} \in \mathcal{F}$  be a unit clause
    If  $x$  is a positive literal Set  $T \leftarrow T \cup \{x\}$ 
     $\mathcal{F} \leftarrow \{C - \{\text{complement}(x)\} : C \in \mathcal{F}, x \notin C\}$ 
    While  $\emptyset \notin \mathcal{F}$  and there is a unit clause in  $\mathcal{F}$ .
    Output ( $\mathcal{F}$ );
```

10.1 2-SAT

All clauses of a 2-SAT formula contain at most two literals. A given 2-SAT formula \mathcal{F} may be solved efficiently because each of its clauses, say $(x \vee y)$, is functionally equivalent to a pair of implications, in this case $\bar{x} \rightarrow y$ and $\bar{y} \rightarrow x$. The first of these implications means that an assignment of *false* to x requires an assignment of *true* to y if the clause $(x \vee y)$ is to be satisfied. A similar meaning holds for the second implication.

A 2-SAT formula may be solved using a limited form of backtracking. Start by applying unit resolution to get rid of all the unit clauses in \mathcal{F} . If clauses remain, assign a value to an arbitrarily chosen variable v . This may result in a unit clause. Then, the application of unit resolution forces some implications to become active. If two complementary literals are in activated implications then a contradiction arises and \mathcal{F} cannot be satisfied using the value given to v . So, reverse the value and try again. If two complementary literals are in activated implications this time, no assignment will satisfy \mathcal{F} so output this fact. Otherwise, unit resolution has completed and no unit clauses remain. From this point on, never change the value of v . If \mathcal{F} is not empty, choose another variable v' and continue as above until either \mathcal{F} is determined to be unsatisfiable or until \mathcal{F} is empty in which case report a solution. Algorithm 2-SAT Solver below expresses these ideas and

determines whether a given 2-SAT formula is satisfiable.

2-SAT Solver(\mathcal{F}):

```

/*  $\mathcal{F}$  must be a 2-SAT formula */
second ← true
 $T \leftarrow \emptyset$ ;  $T' \leftarrow \emptyset$ ;  $\mathcal{F}' \leftarrow \emptyset$ 
Repeat the following
   $\mathcal{F} \leftarrow \text{UR}(\mathcal{F}, T')$ 
  If  $\emptyset \in \mathcal{F}$ 
    If  $\text{second} = \text{true}$  or  $\mathcal{F}' = \emptyset$  Output ("unsatisfiable")
     $\text{second} \leftarrow \text{true}$ 
     $T' \leftarrow \emptyset$ ;  $\mathcal{F} \leftarrow \mathcal{F}'$ ;  $x \leftarrow x'$ 
    If  $x$  is a negative literal Set  $T' \leftarrow \{\text{complement}(x)\}$ 
     $\mathcal{F} \leftarrow \{C - \{x\} : C \in \mathcal{F}, \text{complement}(x) \notin C\}$ 
  Otherwise If  $\mathcal{F} \neq \emptyset$ 
     $\text{second} \leftarrow \text{false}$ 
    Choose a literal  $x$  arbitrarily from  $\mathcal{F}$ 
     $T \leftarrow T \cup T'$ ;  $T' \leftarrow \emptyset$ ;  $\mathcal{F}' \leftarrow \mathcal{F}$ ;  $x' \leftarrow x$ 
    If  $x$  is a positive literal Set  $T' \leftarrow \{x\}$ 
     $\mathcal{F} \leftarrow \{C - \{\text{complement}(x)\} : C \in \mathcal{F}, x \notin C\}$ 
  Otherwise Output ( $T \cup T'$ );

```

A satisfying assignment can be constructed from $T \cup T'$ if it is output: namely, all variables in $T \cup T'$ take value *true* and all others take value *false*. This algorithm is adapted from [156]. A good implementation has complexity linear in the size of the given formula.

10.2 Horn Formulas

A *CNF* formula is Horn if every clause in it has at most one positive literal. This class is widely studied, in part because of its close association with Logic Programming. Namely, a Horn clause $(\bar{x}_1 \vee \bar{x}_2 \vee \dots \vee \bar{x}_i \vee y)$ is equivalent to the rule $x_1 \wedge x_2 \wedge \dots \wedge x_i \rightarrow y$ or the implication $x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_i \rightarrow y$ (where association is from right to left). However, the notion of causality is generally lost when translating from rules to Horn formulas.

Horn formulas can be solved in linear time using unit resolution [145, 286, 479]. The following Algorithm *Horn Solver* is more efficient than using

UR because only unit clauses containing positive literals are considered.

Horn Solver(\mathcal{F}):

```
/*  $\mathcal{F}$  must be a Horn formula */
 $T \leftarrow \emptyset$ 
Repeat the following
  If there is a unit clause  $\{x\} \in \mathcal{F}$ ,  $x$  a positive literal
     $T \leftarrow T \cup \{x\}$ 
     $\mathcal{F} \leftarrow \{C - \{\bar{x}\} : C \in \mathcal{F}, x \notin C\}$ 
  If  $\emptyset \in \mathcal{F}$  Then Output ("unsatisfiable")
Until there are no unit clauses in  $\mathcal{F}$ .
Output ( $T$ );
```

If T is output, the assignment such that all variables in T are given the value *true* and all other variables are given the value *false* satisfies \mathcal{F} . If "unsatisfiable" is output, then no satisfying truth assignment exists for \mathcal{F} . The unsatisfiable Horn formula $(\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_1) \wedge (x_1)$ demonstrates that such an output is possible. In words, Algorithm *Horn Solver* applies *positive* unit resolution to exhaustion or until some clause becomes falsified by the assignment implied by T .

An important property of Horn formulas is that a satisfying assignment, if one exists, which is minimal in the number of variables set to *true* is a *unique* minimum assignment with respect to *true*. This property has important implications in, among other things, efficient algorithms for some other classes of SAT (e.g. see Section 10.6). The Algorithm *Horn Solver* finds a unique minimum satisfying assignment with respect to *true*, if one exists,

Algorithm *Horn Solver* is useful only if the input formula is known to be Horn. This is a property that, it is easy to see, can be checked in linear time.

A formula may be a Horn formula in *disguise*. Such a formula, after a suitable renaming of variables (for a renamed variable, positive literals become negative literals and negative literals become positive) is Horn. Renamable or disguised Horn formulas are treated in Section 10.7.

10.3 Extended Horn Formulas

There are several extensions to Horn formulas. One class, called the class of extended Horn formulas, was introduced by Chandru and Hooker [80] who were looking for conditions under which a Linear Programming relaxation could be used to find solutions to propositional formulas. A theorem of Chandrasekaran [85] characterizes sets of linear inequalities for which 0-1 solutions can always be found (if one exists) by rounding a real solution obtained using an LP relaxation. Extended Horn formulas can be expressed as linear inequalities that belong to this family of 0-1 problems.

Chandru and Hooker [80] showed that unit resolution alone can determine whether or not a given extended Horn formula is satisfiable. This is due to the following two properties of an extended Horn formula:

1. If \mathcal{F} is extended Horn and has no unit clauses then \mathcal{F} is satisfiable.
2. If \mathcal{F} is extended Horn and v is any variable in \mathcal{F} then
 $\mathcal{F}_1 = \{C - \{v\} : C \in \mathcal{F}, \bar{v} \notin C\}$ and $\mathcal{F}_2 = \{C - \{\bar{v}\} : C \in \mathcal{F}, v \notin C\}$
 are both extended Horn.

A satisfying truth assignment for a satisfiable formula may be found by applying unit resolution, setting values of unassigned variables to 1/2 when no unit clauses remain, and rounding the result by a matrix multiplication [80]. This algorithm cannot, however, be reliably applied unless it is known that a given formula is extended Horn. Unfortunately, the problem of recognizing extended Horn formulas is not known to be solved in polynomial time. However, the extended Horn formulas are a subclass of SLUR formulas (see Section 10.5) which can be solved in linear time without the need for a recognition algorithm. Although the class of extended Horn formulas has numerous interesting properties, we defer discussion of solving formulas in this class to Section 10.5 since our primary focus is on fast, reliable solution methods.

10.4 Formulas from Balanced $(0, \pm 1)$ Matrices

The class of formulas from balanced $(0, \pm 1)$ matrices, which we call CC-balanced formulas here, has been studied by several researchers (see [101] for a detailed account of balanced matrices and a description of CC-balanced formulas). The motivation for this class is the question, for SAT, when do Linear Programming relaxations have integer solutions?

Express a *CNF* formula of m clauses and n variables as an $m \times n$ $(0, \pm 1)$ -matrix M where the rows are indexed on the clauses, the columns are indexed on the variables, and a cell $M(i, j)$ has the value $+1$ if clause i has variable j as an unnegated literal, the value -1 if clause i has variable j as a negated literal, and the value 0 if clause i does not have variable j as a negated or unnegated literal. A *CNF* formula is a *CC-balanced formula* if in every submatrix of M with exactly two nonzero entries per row and per column, the sum of the entries is a multiple of four [520].

CC-balanced formulas have the property that, if every clause contains more than one literal, then for every variable v there are two satisfying truth assignments: one with v set to *true* and one with v set to *false*. Algorithm *Balanced Solver* below is a simple linear-time algorithm for finding solutions to known CC-balanced formulas based on this property [101]. It assigns *true* values to variables only when forced to do so by *UR*.

Balanced Solver(\mathcal{F}):

```
/*  $\mathcal{F}$  must be a CC-balanced formula as defined here */
 $T \leftarrow \emptyset$ 
Repeat the following
   $\mathcal{F} \leftarrow \text{UR}(\mathcal{F}, T)$ 
  If  $\emptyset \in \mathcal{F}$  Output ("unsatisfiable")
  If  $\mathcal{F} = \emptyset$  Output ( $T$ )
  Choose arbitrarily a variable  $x \in \mathcal{F}$ 
   $\mathcal{F} \leftarrow \{C - \{x\} : C \in \mathcal{F}, \bar{x} \notin C\}$ 
  While  $\mathcal{F} \neq \emptyset$ .
    Output ( $T$ );
```

Unlike extended Horn formulas, CC-balanced formulas can be recognized in polynomial time [101].

10.5 Single-Lookahead Unit Resolution

This class is an extension of Horn, extended Horn, and CC-balanced formulas [476]. It is peculiar in that it is defined based on an algorithm rather than on properties of formulas. The algorithm, called *SLUR*, selects variables sequentially and arbitrarily and considers a one-level lookahead, under unit resolution, of both possible values that the selected variable can take. If unit resolution does not result in an empty clause in one direction the

assignment corresponding to that value choice is made permanent and variable selection continues. If all clauses are satisfied after a value is assigned to a variable (and unit resolution is applied), the algorithm returns a satisfying assignment. If unit resolution, applied to the given formula or to both sub-formulas created from assigning values to the selected variable on the first iteration, results in a clause that is falsified, the algorithm reports that the formula is unsatisfiable. If unit resolution results in falsified clauses as a consequence of both assignments of values to the selected variable on any iteration except the first, the algorithm reports that it has given up.

SLUR(\mathcal{F}):

```

/*  $\mathcal{F}$  can be any CNF formula */
 $T \leftarrow \emptyset$ 
 $\mathcal{F} \leftarrow \text{UR}(\mathcal{F}, T)$ 
If  $\emptyset \in \mathcal{F}$  Output ("unsatisfiable")
 $level \leftarrow 0$ 
Repeat the following
    Choose arbitrarily a variable  $x \in \mathcal{F}$ 
     $T_1 \leftarrow T; T_2 \leftarrow T$ 
     $\mathcal{F}_1 \leftarrow \text{UR}(\{C - \{x\} : C \in \mathcal{F}, \bar{x} \notin C\}, T_1)$ 
     $\mathcal{F}_2 \leftarrow \text{UR}(\{C - \{\bar{x}\} : C \in \mathcal{F}, x \notin C\}, T_2)$ 
    If  $\emptyset \in \mathcal{F}_1$  and  $\emptyset \in \mathcal{F}_2$ 
        If  $level = 0$  Output ("unsatisfiable")
        Otherwise Output ("give up")
    Otherwise
        Arbitrarily choose  $i$  so that  $\emptyset \notin \mathcal{F}_i$ 
         $\mathcal{F} \leftarrow \mathcal{F}_i; T \leftarrow T_i$ 
        If  $i = 2$  then  $T \leftarrow T \cup \{x\}$ 
         $level \leftarrow 1$ 
    While  $\mathcal{F} \neq \emptyset$ 
        Output ( $T$ )

```

A formula is in the class SLUR if, for all possible sequences of selected variables, Algorithm *SLUR* does not give up on that formula. Note that due to the definition of this class, the question of class recognition is avoided.

Algorithm *SLUR* takes linear time with the modification, due to Truemper [523], that unit resolution be applied simultaneously to both branches of

a selected variable, abandoning one branch if the other finishes first without falsifying a clause.

All Horn, extended Horn, and CC-balanced formulas are in the class SLUR. Thus, an important outcome of the results on the SLUR class is the observation that no special preprocessing or testing is needed for some of the special polynomial time solvable classes of SAT when using a reasonable variant of the DPL algorithm.

A limitation of all the classes considered so far is they do not represent many interesting unsatisfiable formulas. There are several possible extensions to the SLUR class which improve this situation. One is to add a 2-SAT solver to the unit resolution (*UR*) steps of Algorithm *SLUR*. This extension is at least able to handle all 2-SAT formulas which is something *SLUR* cannot do. This extension can be elegantly incorporated due to an observation of Truemper: “Whenever *SLUR* completes a sequence of unit resolutions, and if at that time the remaining clauses are nothing but a subset of the original clauses (which they would have to be if all clauses have at most two literals), then effectively the *SLUR* algorithm can start all over. That is, if fixing of a variable to both values leads to an empty clause, then the formula has been proved to be unsatisfiable. Thus, one need not augment *SLUR* by the 2-SAT algorithm, because the 2-SAT algorithm (at least one version of it) does exactly what the modified *SLUR* does.” Another extension of SLUR is to allow a polynomial number of backtracks, giving up if at least one branch of the search tree does not terminate at a leaf where a clause is falsified. Thus, unsatisfiable formulas with short search trees can be solved efficiently. However, such formulas are uncommon among the set of all formulas.

10.6 q-Horn Formulas

This class is another extension of Horn formulas [44] and [45]. Formulas in the class q-Horn are thought to be close to what might be regarded as the largest easily definable class of polynomially solvable propositional formulas because of a result due to Boros, Crama, Hammer, and Saks [45]. Let $\{x_1, x_2, \dots, x_n\}$ be a set of Boolean variables, and P_k and N_k , $P_k \cap N_k = \emptyset$, be subsets of $\{1, 2, \dots, n\}$ such that the k th clause in a CNF formula is given by $\vee_{i \in P_k} x_i \vee_{i \in N_k} \bar{x}_i$. Construct the following system of inequalities:

$$\sum_{i \in P_k} \alpha_i + \sum_{i \in N_k} (1 - \alpha_i) \leq Z, \quad (k = 1, 2, \dots, m), \text{ and} \quad (47)$$

$$0 \leq \alpha_i \leq 1 \quad (i = 1, 2, \dots, n). \quad (48)$$

where $Z \in R^+$. The minimum value of Z that satisfies (47) and (48) for a formula is the *satisfiability index* for that formula. If a formula's satisfiability index is no greater than 1, then it is q-Horn. However, the class of all formulas with a satisfiability index greater than $1 + 1/n^\epsilon$, for any fixed $\epsilon < 1$, is NP-complete.

An important question to ask at this time is whether the q-Horn class is contained in the SLUR class or vice versa. The answer is that the two classes are incomparable. The following is a q-Horn formula that is not in the SLUR class:

$$(x_1 \vee \bar{x}_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_6) \wedge (\bar{x}_1 \vee x_3 \vee x_7).$$

This formula is not SLUR because the variable choices of $x_4 - x_7$ in Algorithm *SLUR* result in $\emptyset \in \mathcal{F}_1$ and $\emptyset \in \mathcal{F}_2$ at the bottom level, causing the algorithm to give up. It is easy to check that the satisfiability index for this formula is no greater than 1. On the other hand, the following formula from the SLUR class is not q-Horn:

$$(x_1 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_3 \vee x_5) \wedge (x_3 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (x_4 \vee \bar{x}_1).$$

The satisfiability index for this formula is $5/4$ but any variable selection sequence in Algorithm *SLUR* leads to a satisfying assignment. It is left to the reader to extend these examples to infinite families with the same properties.

10.7 Renamable Formulas

Suppose clauses of a *CNF* formula \mathcal{F} are constructed from a set V of variables and let $V' \subset V$. Define $switch(\mathcal{F}, V')$ to be the formula obtained from \mathcal{F} as follows: for every variable $v \in V'$, reverse all unnegated occurrences of v in \mathcal{F} to negated occurrences and all negated occurrences of v to unnegated occurrences. For a given formula \mathcal{F} , if there exists a $V' \subset V$ such that $switch(\mathcal{F}, V')$ is Horn, then the formula is said to be renamable Horn. Renamability for the other above mentioned classes is superfluous since the algorithms given above for the q-Horn and SLUR class work even if a given formula is renamable to a q-Horn or SLUR formula. Therefore, at least one of these algorithms will work for any formula that is renamable to any class considered so far in this section. Additional classic references to Horn renamability are [347] and [19].

10.8 Formula Hierarchies

Some sets of clauses not falling into one of the polynomially solvable classes defined above may be reduced to “equivalent” formulas that are members of at least one of these classes. If such reductions are efficient, these sets can be solved in polynomial time. Such reductions can take place in stages where each stage represents a class of polynomially solved formulas, and lower stages represent classes of perhaps lower time complexity than classes represented by higher stages. The lowest stage is a polynomially solved base class, such as one of the classes above.

An example of such a hierarchy is found in [185]. The base class, at stage 0, is Horn. Consider a stage 1 formula that is not Horn. By definition of the hierarchy, there is a variable v which, if set to *true*, leaves a set of non-satisfied clauses and non-falsified literals that is Horn. If this Horn formula is found to be satisfiable, we can conclude the original formula is. Otherwise, setting v to *false* leaves a set of clauses that is a stage 1 formula (empty formulas are considered to belong to every stage). Thus, the above process can be repeated (on stage 1 formulas) to exhaustion. Since it takes linear time to solve Horn formulas and in the worst-case a linear number of Horn systems must be considered, the process for solving formulas at stage 1 has quadratic complexity. The above concept can be expanded to higher stages to form a hierarchy: at stage i , when setting v to *true*, a sub-formula is at stage $i - 1$, and when setting v to *false*, a sub-formula is at stage i . Thus, solutions to stage i formulas are carried out recursively leading to a time complexity that is bounded by m^i . An alternative way to solve formulas at stage i in the hierarchy is to use i -resolution (resolution is not applied unless at least one clause has at most i literals) [63].

The only remaining question is to determine whether a given formula is a stage i formula. This can be done with a bottom-up approach described in [185].

For other information on such Hierarchies see, for example, [114, 187].

10.9 Non-linear Formulations

An optimization problem with 0-1 variables can be reduced to a constrained nonlinear 0-1 program. Such programs are expressed as follows:

$$\max_{\mathbf{x} \in \{0,1\}^n} F(\mathbf{x}) = \sum_{k=1}^p c_k T_k \quad (49)$$

subject to

$$g_i(\mathbf{x}) = \sum_{k=1}^{p_i} a_{ik} T_{ik} < b_i; \quad i = 1, 2, \dots, m$$

where

$$T_k = \prod_{j \in N_k} x_j; \quad N_k \subseteq N = \{1, 2, \dots, n\}, \quad k = 1, 2, \dots, p \quad (50)$$

and

$$T_{ik} = \prod_{j \in N_{ik}} x_j; \quad N_{ik} \subseteq N, \quad k = 1, 2, \dots, p_i, \quad i = 1, 2, \dots, m$$

Problems in propositional logic, originating, for example, in graph theory, can be expressed this way by associating 0 to *false*, 1 to *true*, $1 - a$ to \bar{a} , and $a \cdot b$ to $a \wedge b$.

Several methods for solving the above formulation have been proposed [242, 244]. In some restricted cases these methods have polynomial time complexity. Thus, the rich literature on this subject can be carried over to the domain of propositional satisfiability to provide low complexity algorithms for SAT under corresponding conditions.

A notable example involves functions for which the co-occurrence graph is a partial k -tree [108]. The *DNF* formulation expressed by equations (9)-(10) is in the form of equations (49) and (50). Let $F(\mathbf{x})$ be such a *DNF* function. The co-occurrence graph of $F(\mathbf{x})$ has a vertex set corresponding to the variables $\{x_1, x_2, \dots, x_n\}$ with an edge between x_i and x_j ($i \neq j$) if these variables occur simultaneously in at least one product term of $F(\mathbf{x})$. A simple, undirected graph G is a k -tree if there is an ordering $\{x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_n}\}$ of all its vertices such that, for all $j = 1, 2, \dots, n - k$, in the subgraph G_j induced by vertices $\{x_{\pi_j}, x_{\pi_{j+1}}, \dots, x_{\pi_n}\}$ the vertex x_{π_j} has degree k and its neighbors induce a complete subgraph of G_j . A partial k -tree is any graph obtained by deleting edges from a k -tree. If the co-occurrence graph of $F(\mathbf{x})$ is a partial k -tree, then $F(\mathbf{x})$ can be solved in linear time [108]. Since the maximization problem for *DNF* formulas is the same as the minimization problem for *CNF* formulas (by using $1 - x$ for literal x and x for literal \bar{x}), *CNF* formulas can be solved in linear time if their corresponding co-occurrence graph is a partial k -tree.

Another example is a linear time algorithm for determining whether a 2-SAT formula has exactly one solution, that is, uniquely solvable. The

question of determining unique solvability is a tough one in general and it is even hard to determine whether linear time algorithms exist for special classes of SAT.⁶ However, one is presented for 2-SAT in [243] using the framework of pseudo-boolean functions (that is, of the form (49) and (50)). Finally, we mention the result of [107] where a polynomial time algorithm for producing a parametric representation of all solutions to a 2-SAT formula is presented.

11 Advanced Techniques

In this section, we describe a number of advanced optimization techniques for satisfiability testing. They have been used in practical engineering applications and have proven to be effective for certain classes of SAT.

11.1 General Boolean Representations

In practice, many problems in integrated circuit design, such as logic verification, test pattern generation, asynchronous circuit design, logic optimization, sequential machine reduction, and symbolic simulation, can be expressed as Boolean satisfiability problems with arbitrary Boolean functions. Each representation has corresponding algorithms for satisfiability testing. A Boolean representation affects the performance of Boolean manipulation methods accordingly. Thus, efficient representation and manipulation of Boolean functions is crucial to many practical applications. Many different representations have been proposed for manipulating Boolean functions. However, many Boolean functions derived from practical circuit design problems suffer from an exponential size in their representations, making satisfiability testing infeasible.

Most SAT algorithms work on *conjunctive normal form (CNF)* formulas, i.e., input formulas must be expressed as a product of sums of literals. The *CNF* formula is a canonical formula used in most analytical studies but is not an efficient representation in practical application problems. Many real engineering design problems use non-clausal representations rather than the *CNF* formula. Algorithms in this category may be regarded as *non-clausal* inference algorithms for satisfiability testing [220]. Compared to

⁶An almost linear time algorithm for unique Horn-SAT has been obtained by Berman et al. [30] and improved into a linear time algorithm by a modification due to Pretolani [431] (Minoux developed a quadratic time algorithm in [387]).

CNF formulas, a non-clausal, general Boolean representation is much more compact and efficient, although the transformation of an arbitrary non-clausal expression into *CNF* can be done in polynomial time by introducing new variables. This will result in clause-form representation of substantially larger sizes [195, 425]. While this is not critical in complexity theory, it will have serious impact in solving practical application problems.

In practice, a SAT algorithm can be made much more efficient if it works directly on problems represented in a compact number of general Boolean formulas rather than a large collection of *CNF* clauses. For a non-clausal SAT algorithm, the evaluation of arbitrarily large, complex Boolean functions is a key to its efficiency [234].

The next two subsections describe a sequential and a parallel Boolean representation and manipulation methods.

11.2 Binary Decision Diagram (BDD)

Ordered Binary Decision Diagrams (OBDDs) [60, 61] is an efficient representation and manipulation method for arbitrary Boolean functions. This representation is defined by imposing restrictions on the Binary-Decision-Diagram (BDD) representation introduced by Lee [342] and Akers [9], such that the resulting form is canonical. The OBDD representation and its manipulation method are an extremely powerful technique in various practical applications. It is particularly useful with formulas where one needs to consider every solution, such as cases where one must search for optimal solutions. Although the OBDD representation of a function may have size exponential in the number of variables, many useful functions have more compact representations in practice.

A BDD gives a graphical representation of Boolean functions. It is a directed acyclic graph with two types of leaf nodes, **0** and **1**. Each non-leaf node is labeled with a Boolean variable v and has two out-going edges labeled 0 (the left edge) and 1 (the right edge). A BDD can be utilized to determine the output value of the function by examining the input values. Every path in a BDD is unique, *i.e.*, no path contains nodes with the same variables. This means that if we arbitrarily trace out a path from the function node to the leaf node 1, then we have automatically found a value assignment to function variables for which function will be 1 regardless of the values of the other variables.

Given a simple example Boolean function $F = (a + b) \cdot (a + c)$, the BDD of function F can be constructed to determine its binary value, given the

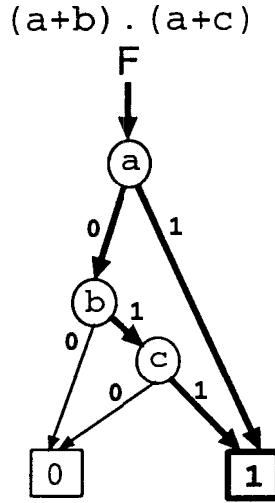


Figure 26: A simple BDD example for $F = (a + b) \cdot (a + c)$.

binary values of variables a , b , and c . At the root node of BDD, we begin at the value of variable a . If $a = 1$, then $F = 1$ and we are finished. If $a = 0$, we look at b . If $b = 0$, then $F = 0$ and again we are finished. Otherwise, we look at c , its value will be the value of F . The complete BDD for function F is shown in Figure 26, where all the paths from the root function node F to the leaf node 1 are highlighted. Each highlighted path yields a satisfiable assignment. For F , the satisfiable assignments are $a = 1$, $b = -$, $c = -$ and $a = 0$, $b = 1$, $c = 1$, where ' $-$ ' denotes a don't care assignment.

It is well known that the BDD size for a given function depends on the variable order chosen for the function (e.g., $\{a,b,c\}$ in Figure 26). Since the early introduction of BDDs, several extensions have been proposed to reduce BDD sizes in practical applications. In an ordered BDD [60, 61], the input variables are ordered, and every path from the root node to the leaf node visits the input variables in an ascending order. In practice, a simple topological based ordering heuristic [372] yields small size BDDs for practical Boolean instances. A reduced ordered BDD is an ordered BDD where each node represents a unique logic function. Bryant showed that the reduced ordered BDD of a Boolean function is well-defined and is a canonical representation of the function; *i.e.*, two functions are equivalent if their reduced ordered BDDs are isomorphic [60, 61].

The DBDD is efficient to search for optimal solutions for arbitrarily complicated Boolean expressions. In VLSI circuit design, many practical problems require the enumeration of *all* possible assignments for a given

Boolean formula. The best assignment that yields the minimum cost (e.g., minimal circuit structure, minimum chip area, and maximum circuit speed) is then selected from these possible assignments. Since most algorithms for satisfiability testing are designed for finding one truth assignment, they are impractical for selecting an optimal assignment. BDDs are very useful in such situations, since a simple and incremental enumeration of all possible paths from the root node to the leaf node 1 yields all the truth assignments. Thus, once the BDD for a Boolean function has been constructed, it is straightforward to enumerate all assignments or find an optimal solution.

The BDD method can effectively handle small and medium size formulas. For larger size formulas, a partitioning into a set of smaller sub-formulas before applying the BDD algorithms has been suggested. This approach works well for asynchronous computer circuit design problems [221, 448].

11.3 Multispace Search

Many search and optimization methods have been developed in combinatorial optimization, operations research, artificial intelligence, neural networks, genetic algorithms, and evolution programming. An optimization algorithm seeks a value assignment to variables such that all the constraints are satisfied and the performance objective is optimized. The algorithm operates by changing values to the variables in the value space. Because value changing does not affect the formula structure and the search space, it is difficult for a value search algorithm to handle the pathological behavior of local minima.

Multispace search is a new optimization approach developed in recent years [227, 234, 235]. The idea of multispace search was derived from principles of non-equilibrium thermodynamic evolution that structural changes are more fundamental than quantitative changes, and that evolution depends on the growth of new structure in biological system rather than just information transmission. A search process resembles the evolution process, and structural operations are important to improve the performance of traditional value search methods [227, 234, 235].

In multispace search, any active component related to the given input structure can be manipulated, and thus, be formulated as an independent search space. For a given optimization problem, for its variables, values, constraints, objective functions, and key parameters (that affect the input structure), we define the variable space, the value space (*i.e.*, the traditional search space), the constraint space, the objective function space, the parameter space, and other search spaces, respectively. The totality of all the

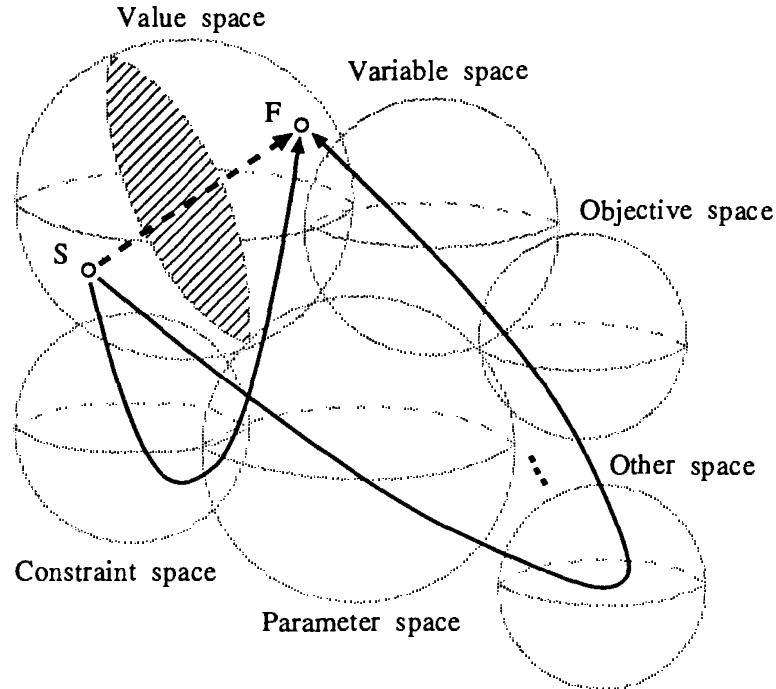


Figure 27: In the value space, a traditional search process (dashed line) cannot pass a “wall” of high cost search states (hatched region). It fails to reach the final solution state, F . A multispace search process (solid lines) scrambles across different search spaces. It could bypass this “wall” through the other search spaces.

search spaces constitutes a *multispace*.

The basic idea of multispace search is simple. Instead of being restricted in the value space, the multispace is taken as the search space. In the multispace, components other than value can be manipulated and optimized as well. During the search, a multispace search algorithm not only alters values in the value space; as shown in Figure 27, it also walks across the variable space and other active spaces, changes dynamically the input structure in terms of variables, parameters, and other components, and constructs systematically a sequence of structured, intermediate instances. Each intermediate instance is solved by an optimization algorithm, and the solution found is used as the initial solution to the next intermediate instance. By interplaying value optimization with structured operations, multispace search incrementally constructs the final solution to the search instance through a sequence of structured intermediate instances. Only at the *last* moment of the search, the reconstructed instance structure approaches the original in-

stance structure, and thus the final value assignment represents the solution of the given search input.

Multispace search algorithm combines traditional optimization algorithms with structural multispace operations. In each search step, multispace search performs two fundamental operations: a traditional value search and the structural reconfiguration of the intermediate instance during each individual search phase. According to the active event in the scrambling schedule [227, 228], the search process enters a specified search space and performs structural operations to the intermediate instance structures, followed by a traditional value search that optimizes the constructed intermediate instance. The resulting intermediate solution is then used as the initial instance to the next phase of multispace search.

The major structural operations in multispace search [227, 228] include multispace scrambling [228, 234], extradimension transition (e.g., air bridge, real dimension, and extra dimension) [217, 219, 218], search space smoothing [231], multiphase search [498, 501, 217, 446, 555, 213, 224], local to global passage [213, 213], tabu search [202], and perturbations (e.g., jumping, tunneling, climbing, and annealing) [215, 216, 217, 218, 315].

Two important preprocessing methods for satisfiability testing are partitioning input size and partitioning variable domain.

Partitioning Input Size.

Due to excessive computing time, a large size NP-hard problem is difficult to solve. Partitioning a large input into a set of smaller sub-instances may give efficient solution of the input. There are two partitioning methods, each consisting of a partitioning, a conquer, and an integration procedure. For constructive partitioning (e.g., divide and conquer), partitioning, conquer, and integration procedures are well defined and easy to implement. For destructive partitioning, it is difficult to design the partitioning and integration procedures.

Gu and Puri have recently developed an efficient input size partitioning technique for satisfiability testing and applied it to asynchronous circuit design [228, 221, 448]. Asynchronous circuits are used in low power and high performance digital systems such as mobile, portable, and military communications. The core problem in asynchronous circuit synthesis can be formulated as an instance of SAT to satisfy the complete state coding (CSC) constraints, *i.e.*, the SAT-Circuit problem [539]. This partitioning preprocessor decomposes a large size SAT formula that represent the given

asynchronous circuit design into a number of smaller, disjoint SAT formulas. Each small size SAT formula can be solved efficiently. Eventually, the results of these sub-formulas are integrated together and contribute to the solution of the original formula. This preprocessor avoids the problem of solving very large SAT formulas and guarantees to finding one best solution in practice. This partitioning preprocessing is destructive since, during the search, extra variables are introduced to resolve the critical CSC constraints. Furthermore, they built a complete, incremental SAT solver based on binary decision diagrams (BDD). Their system is able to find an optimal solution to the asynchronous circuit design problem efficiently.

Partitioning Variable Domain.

A variable domain contains values to be assigned to variables. The size of a variable domain, along with the number of variables, determine the computational complexity of an optimization algorithm. From a theoretical point of view, even a small reduction in the variable domain would result in significant improvements in computing efficiency. It is, however, difficult to make use of variable-domain reduction techniques in solving optimization problems. Recently, Wang and Rushforth have studied mobile cellular network structures and developed a novel variable-domain reduction technique for channel assignment in these networks [555, 556]. This application provides practical benchmarks for satisfiability testing.

Wang and Rushforth's channel assignment algorithm was developed based on the structure of cellular frequency reuse patterns. Using their variable domain partitioning technique, they partition a mobile cellular network with larger variable domain into two networks: a minimum network with a fixed and small variable domain (due to the known frequency reuse patterns) and a difference network with an even smaller variable domain [555, 556]. Channels are assigned separately to the minimum network and to the difference network, and the superposition of these two assignments constitutes an assignment to the original network.

Because this variable domain partitioning approach decomposes an instance of a channel assignment problem with a large number of assignments into two separate channel assignment sub-instances with considerably smaller numbers of assignments, it dramatically reduces the computational complexity and thus the computing efficiency for solving given inputs, in addition to the significantly improved solution results. This novel partitioning technique can be applied to solve the channel assignment problem with

any existing channel assignment algorithms. During numerous channel assignment experiments, this algorithm outperformed all available algorithms for solving the practical channel assignment problem benchmarks. Experimental evidence suggests that this partitioning approach is both efficient and effective.

11.4 Parallel SAT Algorithms and Architectures

Many parallel SAT and CSP algorithms have been developed. In a recent survey [220], the following parallel algorithms for solving SAT were discussed:

- 1987: Parallel DP algorithms
- 1986: Parallel discrete relaxation chips
- 1987: Parallel backtracking architecture
- 1987: Parallel local search algorithm
- 1989: Parallel interior point method
- 1990: Parallel, differential, non-clausal inference
- 1991: Parallel $\alpha\beta$ relaxation
- 1991: Parallel unconstrained optimization
- 1992: Neural network approach
- 1993: Multiprocessor local search

For the following two reasons, algorithms running on loosely-coupled, multiprocessor parallel computers offer limited performance improvements for solving SAT. First, in the worst case, a SAT algorithm may suffer from the exponential growth in computing time. In order to solve a SAT formulas effectively, we will need a computer that has much larger speedup than what is available today. This computer will require the integration of at least a few million processors in a tightly-coupled manner. This is infeasible in the current computer system integration technology.

Second, as the processor gets much faster, the communication overhead among processors in a parallel machine becomes a bottleneck, which may often take 70 % to even 90 % of the total computing time [300]. Ideally one would expect the speedup on a parallel computer to increase linearly with increasing number of processors. Due to serious off-processor communication delays, after certain saturation point, adding processors does not increase speedup on a loosely-coupled parallel machine. Processor communication delay also makes process creation, process synchronization, and remote memory access very expensive in a loosely-coupled multiprocessor

system. For this reason, the speedup on a multiprocessor is normally less than the number of processors used. With simple SAT algorithms, however, speedup is sometimes greater than the number of processors [396]. Variable settings similar to those that are already known not to lead to a solution are also unlikely to lead to a solution. The obvious methods of parallelizing simple SAT algorithms break up the tendency to search similar settings at about the same time.

From our experience, tightly coupled parallel computing, which effectively reduces off-processor communication delays, is a key to the parallel processing of SAT formulas [220]. In order to use a tightly-coupled parallel architecture for SAT computation, one must map a computing structure to the input structure and must reduce the total number of sequential computing steps through a large number of symmetrical interactions among simple processors [211, 220]. Several different approaches, e.g., special-purpose parallel VLSI architectures [232, 233], bit-parallel programming on sequential machines [212, 217, 503, 502], and tight programming on parallel computer systems, are promising alternatives in this direction. These approaches are capable of providing a tight mapping between a formula structure and a computing structure, resulting in faster computation. The computational power of these approaches are orders of magnitude greater than standard sequential algorithms running on uniprocessor machines or parallel algorithms running on loosely coupled multiprocessors.

Parallel processing does not change the worst-case complexity of a SAT algorithm unless one has an exponential number of processors. Parallel processing, however, does delay the effect of exponential growth of computing time, allowing one to solve larger size instance of SAT.

11.5 The *Multi-SAT* Algorithm

The problem structures of real world practical applications vary significantly, making it difficult to develop an efficient SAT algorithm to solve a wider range of the practical application problems. Many efficient algorithms have been developed for the SAT problem. They each can solve a class of problem instances efficiently. Backtracking algorithms can handle some small size, hard problem instances, providing complete solutions. Local search could handle fairly large-size satisfiable problem instances quickly. BDD SAT solver is able to solve practical problem instances with performance criteria. Lagrangian-base global search method can provide solutions to a wide range of SAT problem instances. Problem size and domain partitioning techniques

can further enhance the existing SAT algorithms, so they can solve much larger size practical problem instances. If we combine the *niches* of several efficient algorithms together, they may handle a much wider range of SAT problem instances efficiently. Another school of concern for the *Multi-SAT* algorithm comes from the existing challenge for SAT algorithm's design and testing. A *Multi-SAT* algorithm can relieve the load of this task, facilitating quick design and testing of the algorithm.

Two algorithm integration approaches (hybrid algorithm and algorithm clustering) have been proposed for algorithm integration. In the *hybrid algorithm* approach, algorithms in different classes are integrated in a single algorithm. The algorithm would make use of different algorithmic niches according to some decision procedures. Early examples of this approach include combining local search with backtracking [217] and combining non-linear optimization with backtracking [211, 233, 218]. The effectiveness of a hybrid algorithm may be limited due to the overheads of decision making and algorithmic context switching.

In the *algorithm clustering* approach ("Future Work" in [218]), algorithms in different classes are optimized individually to achieve the best performance. Each algorithm is executed on a computer. A cluster of computers is used to execute several algorithms selected from different classes. The individual results of the algorithms' executions are integrated together, producing the final result. The algorithm clustering approach does not suffer from any performance degradation due to direct algorithm integration. Computer hardware prices continue to decrease, a cluster of computers can be built in a cost-effective way (a PC can now be purchased with around \$1,000). The only requirement for clustering computation is a multi-tasking integration software.

In our first implementation of the *Multi-SAT* algorithm, we select several efficient SAT algorithms. These include, for examples, *DPL* and *CSAT* from backtracking algorithm, *SAT1*, *GSAT*, and *SAT3* from local search algorithm, *BDD* SAT solver from binary decision diagram algorithm, and *DLM* from Lagrangian-base global search method. Combining problem size and domain partitioning techniques, they together support an effective satisfiability testing for problem instances with uncertain structures, using "many stones" to shoot "one bird." By changing parameters and initial starting points, *Multi-SAT* can track an algorithm structure, allow a detailed study of the entire problem spectrum, and provide a cost-effective tool kit for practical satisfiability testing.

A basic software system for the *Multi-SAT*, *Clustor*, is shown in Figure

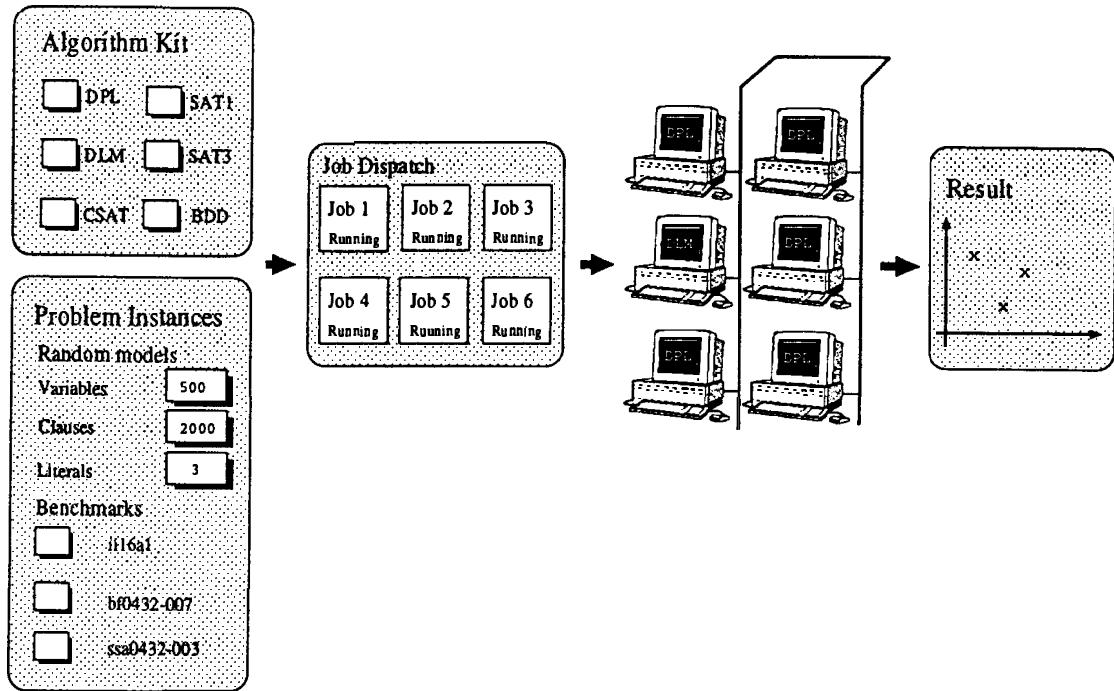


Figure 28: A software system, *Clustor*, for *Multi-SAT* algorithm.

28. We have an *algorithm tool kit* collecting candidate algorithms from different algorithm classes, a *problem instance database* for user to select the problem instances, a distributed system software, *job dispatcher*, for remote job execution control and execution result collection, and a network of computers executing the selected algorithms. The software system can be run on a PC platform or a UNIX platform under an interactive graphical interface.

12 Probabilistic and Average-Case Analysis

Probabilistic and average-case analysis can give useful insight into the question of what SAT algorithms might be effective and why. Under certain circumstances, one or more structural properties shared by each of a collection of formulas may be exploited to solve such formulas efficiently; or structural properties might force a class of algorithms to require superpolynomial time. Such properties may be identified and then, using probabilistic analysis, one may argue that these properties are so common that the performance of an algorithm or class of algorithms can be predicted for most of a family of formulas.

The main drawbacks of this approach are: 1) some distribution of input formulas must be assumed and a chosen distribution may not represent reality very well; 2) results are usually sensitive to the choice of distribution; 3) the state of analytical tools is such that distributions yielding to analysis are typically symmetric with independent components; 4) few algorithms have yielded to analysis. Despite these drawbacks, probabilistic results can be a useful supplement to worst-case results, which can be overly pessimistic for NP-complete problems, in understanding algorithmic behavior.

This section reviews some notable probabilistic and average-case results for certain SAT algorithms. The results we present are based on one CNF distribution, the l -SAT distribution, mainly because most interest seems to be in results using this distribution. Other works such as [210] have information about the average l -SAT and other distributions.

For convenience, a clause C is represented as a set of literals and a formula \mathcal{F} as a set of clauses. If x is a literal, then $\text{complement}(x)$ is the complement of x . If x is a literal and \mathcal{C} is a set of clauses then $\text{occurrences}(x, \mathcal{C})$ is the number of times x appears in clauses of \mathcal{C} . When we say variable $v \in C$ (alternatively, variable v is in clause C) we mean either literal $v \in C$ or literal $\bar{v} \in C$. When we say variable $v \in \mathcal{F}$ (alternatively, variable v is in formula \mathcal{F}) we mean there is a clause $C \in \mathcal{F}$ such that $v \in C$. When we say there is a *path* through a set of clauses $\{C_1, C_2, C_3, \dots, C_p\}$ we mean there is a permutation π on clause indices and a set $\{v_1, v_2, \dots, v_{p-1}\}$ of variables such that $v_i \in C_{\pi_i}$ and $v_i \in C_{\pi_{i+1}}$, $i = 1, 2, 3, \dots, p - 1$; this path becomes a *cycle* if $v_1 \in C_{\pi_p}$. All formulas are constructed from a set V of n variables.

12.1 The l -SAT Distribution

The parameters of this distribution are the number of clauses m , the number of variables n , and the number of literals l in each clause. Each clause in a random formula is constructed as follows: uniformly choose a size l subset of V and negate each variable in the subset independently with probability $1/2$. A random l -SAT formula is a collection of m such clauses, independently generated.

This distribution does not generate unrealistic clauses as does the average l -SAT distribution (see, for example, [210]). That is, null clauses, and tautological clauses (complementary literals in the same clause) do not exist in a random l -SAT formula.

The probabilistic analysis of SAT algorithms under the l -SAT distribution is often difficult. This is partly due to the structure of subformulas when

assigning a value to a variable on an iteration of a particular algorithm. If subformulas are distributed in the same way as the original formula, the analysis can usually proceed easily. But, statistical dependence between clauses and literals often crops up and prevents this. A notable exception, however, is the analysis of variants of the unit clause rule as described in *Algorithms for satisfiable formulas* below.

In what follows, when we refer to l -SAT, we assume $l \geq 3$ unless specifically stated (as in, for example, 2-SAT).

Satisfiable and Unsatisfiable Formulas.

A random l -SAT formula is satisfiable with probability tending to 1 when $m/n < 2^l/4l$. This was proven by showing that a certain unit resolution based algorithm called *SC* finds solutions to random l -SAT formulas with high probability for any ratio of m/n in that range (see *Algorithms for satisfiable formulas* below for details).

It is straightforward to show that a random l -SAT formula is unsatisfiable with probability tending to 1 when $m/n > (.69)2^l > -1/\log_2(1 - 2^{-l})$ [169]. The probability that at least one satisfying truth assignment exists for a random formula is less than the expected number of such truth assignments. The probability that a given truth assignment satisfies a random formula is $((2^l - 1)/2^l)^m$ so the expected number of satisfying assignments is

$$\left(1 - \frac{1}{2^l}\right)^m 2^n$$

which tends to 0 if $m/n > -1/\log_2(1 - 2^{-l})$. This is not a tight bound, however, since the variance of the number of satisfying assignments is high.

There are at least two reasons for interest in the division of values of m/n into primarily satisfiable and unsatisfiable regions:

1. Quite a few algorithms perform differently in both regions. In fact, some, for example local search variants, are designed to work well only in one region or the other. Therefore, knowing the boundaries of a region helps to understand the performance limitations of some algorithms.
2. Experiments have shown that formulas become harder to solve as m/n increases toward the point where about 50% of formulas are satisfiable then become easier as m/n increases beyond that point.

Unfortunately, we do not know precisely where this 50% point is, only that it is somewhere in the gap $2^l/4l < m/n < (.69)2^l$ for $l \geq 3$. The search for this “threshold” has generated a succession of ever improving results [181, 311, 370, 151, 318] which have focused primarily on 3-SAT formulas. On the lower end, the current best result, due to Frieze and Suen [181], shows a random 3-SAT formula is satisfiable with probability tending to 1 if $m/n < 3.003$ (see *Algorithms for satisfiable formulas* below). On the upper end, the current best reported result shows a random 3-SAT formula is unsatisfiable with probability tending to 1 if $m/n > 4.64$.

The latter bound is still not close to the experimentally observed upper boundary of approximately $m/n < 4.24$ on the predominantly satisfiable region (see, for example, [485]). It is based on identifying, for a given formula \mathcal{F} , a set of satisfying “critical” truth assignments that is far smaller than the complete set of assignments satisfying \mathcal{F} . The expected number of critical assignments is a better upper bound on the probability that a formula is satisfiable than the expected number of satisfying assignments and it can be found with some cleverness. The reader is referred to [318] for details.

We remark that the case $l = 2$ has been decided in two papers [87, 203]. The result is that random 2-SAT formulas are satisfiable with probability tending to 1 if $m/n < 1$ and unsatisfiable with probability tending to 1 if $m/n > 1$. This is proven, on the upper end, by finding a structure, not too large, that implies unsatisfiability if it is present (the structure amounts to a cycle of implications that forces a variable to contradict any value it is given), and then showing the structure is present in a random formula with probability tending to 1 if $m/n > 1$. Since the structure is not too large, the second moment method can reliably be used to get a tight bound on m/n . On the lower end, a structure, not too large, that must be in any unsatisfiable formula is shown to exist in a random 2-SAT formula with probability tending to 0 for $m/n < 1$. The second moment method is again used in this case.

Polynomial Time Solvable Classes.

Formulas that are members of certain polynomial time solvable classes, discussed in Section 10, are not generated frequently enough, for interesting values of m/n , to assist in determining satisfiability of random l -SAT formulas. This is unlike the situation for the average l -SAT distribution. We illustrate with a few examples.

First consider the frequency with which a random l -SAT formula is Horn

or hidden Horn. The probability that a clause is Horn is $(l+1)/2^l$. Therefore, the probability that a random l -SAT formula is Horn is $((l+1)/2^l)^n$ which tends to 0 for any fixed l . A formula is hidden Horn if there is a subset of variables (a *switch set*) whose literals can all be reversed to yield a Horn formula. Regardless of switch set, there are only $l+1$ out of 2^l ways (negation patterns) that a random clause can become Horn. Therefore, the expected number of successful switch sets is $2^n((l+1)/2^l)^m$ which tends to 0 if $m/n > 1/(l - \log_2(l+1))$. Thus, random l -SAT formulas are not hidden Horn, with probability tending to 1, if $m/n > 1/(l - \log_2(l+1))$. Even when $l = 3$, this is $m/n > 1$. But this bound can be improved considerably by finding complex structures that imply a formula cannot be hidden Horn. Such a structure is presented next.

The following result for q-Horn formulas is taken from [174]. For $p = \lfloor \ln n \rfloor \geq 4$, call a set of p clauses a *c-cycle* if all but two literals can be removed from each of $p - 2$ clauses, all but three literals can be removed from two clauses, the variables can be renamed, and the clauses can be reordered in the following sequence

$$\{x_1, \bar{x}_2\}, \{x_2, \bar{x}_3\} \dots \{x_i, \bar{x}_{i+1}, x_{p+1}\} \dots \{x_j, \bar{x}_{j+1}, \bar{x}_{p+1}\} \dots \{x_p, \bar{x}_1\} \quad (51)$$

where $x_i \neq x_j$ if $i \neq j$. Given a c-cycle $\mathcal{C} \subset \mathcal{F}$, if no two literals removed from \mathcal{C} are the same or complementary, then \mathcal{C} is called a *q-blocked c-cycle*.

If a formula \mathcal{F} has a q-blocked c-cycle then it is not q-Horn. Let a q-blocked c-cycle in \mathcal{F} be represented as above. Develop inequalities (47) and (48) for \mathcal{F} . After rearranging terms in each, a subset of these inequalities is as follows

$$\alpha_1 \leq Z - 1 + \alpha_2 - \dots \quad (52)$$

...

$$\alpha_i \leq Z - 1 + \alpha_{i+1} - \alpha_{p+1} - \dots$$

...

$$\alpha_j \leq Z - 1 + \alpha_{j+1} - (1 - \alpha_{p+1}) - \dots$$

...

$$\alpha_p \leq Z - 1 + \alpha_1 - \dots \quad (53)$$

From inequalities (52) to (53) we deduce

$$\alpha_1 \leq pZ - p + \alpha_1 - (1 - \alpha_{p+1} + \alpha_{p+1}) - \dots$$

or

$$0 \leq pZ - p - 1 - \dots$$

where all the terms in \dots are non-negative. Thus, all solutions to (52) through (53) require $Z > (p + 1)/p = 1 + 1/p = 1 + 1/\lfloor \ln^2 n \rfloor > 1 + 1/n^\beta$ for any fixed $\beta < 1$. This violates the result of [45] that requires $Z \leq 1$ in order for \mathcal{F} to be q-Horn. We remark that the class of formulas for which $Z > 1 + 1/n^\beta$ for any fixed $\beta < 1$ is NP-complete [45].

The expected number of q-blocked c-cycles can be found and the second moment method applied to give the result that a random l -SAT formula is not q-Horn, with probability tending to 1, if $m/n > 4/(l^2 - l)$. For $l = 3$ this is $m/n > 2/3$.

A similar analysis yields the same results for hidden Horn, SLUR, CC-balanced, or extended Horn formulas.

Most of the polynomial time solvable classes considered in Section 10 suffer from the same problem: vulnerability to some “cycle”-like structure, such as (51), among the clauses of a given formula. Since the l -SAT distribution plays no favorites, if the probability that any “cycle”s appear tends to 1, the probability that a “killer cycle” for a particular class appears usually tends to 1 as well. Since cycles begin to appear with high probability when formulas are still extremely sparse (that is, when $m/n \approx 1/l^2$) and one “killer cycle” is enough to prevent a formula from being a member of a special class, these classes cover little of the m/n spectrum.

Resolution.

Variants of resolution based algorithms have been the most extensively studied using the l -SAT distribution. Analysis is divided into two regions of the parameter space: where random formulas are satisfiable with high probability and where they are unsatisfiable with high probability.

Algorithms for Satisfiable Formulas.

Where random formulas are satisfiable with high probability, the behavior of some resolution based search algorithms can be understood best by means of performance results on corresponding *linear algorithms*. A linear algorithm iteratively extends a partial assignment until either satisfiability is proven because the partial assignment is a satisfying assignment, or until it is found that no extension will lead to a satisfying assignment. In the former case the algorithm reports a satisfying assignment and in the latter case the algorithm gives up. Because they have no backtracking compo-

nent, linear algorithms terminate in polynomial time. The computational path taken by a linear algorithm can be thought of as the first explored branch in the search space of a corresponding search algorithm: that is, at each iteration of the linear algorithm, choices of variable and value are made based on heuristics used by a search algorithm. The performance of linear algorithms is measured by the frequency with which a solution is returned and is sensitive to the heuristics used. Positive results carry over to the corresponding search algorithm.

The most successful results to date on linear algorithms have been obtained for a class grounded in the unit clause rule. We say successful because 1) analysis is possible; 2) the results are quite good compared to other heuristics; and 3) the results illuminate the mechanics that cause good behavior. We next present an informal discussion of the analysis of several such linear algorithms. The main intention is to focus on motivation and intuition with a minimum of details.

A clause-flow model can be used to analyze the mechanics of many linear algorithms. Let \mathcal{F} be an l -SAT formula. The linear algorithms discussed below choose an unassigned literal on each iteration (omitted from the algorithms are the actual assignments which set the chosen literals to *true*). Such a choice causes some clauses of \mathcal{F} to disappear because they have become satisfied and some clauses have a literal removed because they have become falsified by the implicit assignment. Therefore, define $C_i^{\mathcal{F}}(j)$ to be the set of clauses of \mathcal{F} that have exactly i literals at the start of the j th iteration (henceforth, the superscript will be dropped for simplicity). Define $w_i(j)$ to be the number of i -literal clauses added to $C_i(j)$ as a result of choosing a literal on the j th iteration. That is, $w_i(j)$ is the flow of clauses into $C_i(j)$ due to picking a literal on the j th iteration. Define $z_i(j)$ to be the number of clauses eliminated from $C_i(j)$ (satisfied) as a result of choosing a literal on the j th iteration. That is, $z_i(j)$ is the flow out of $C_i(j)$ due to picking a literal on the j th iteration. Let $m_i(j) = |C_i(j)|$.

The success of a linear algorithm depends critically on what is happening to $w_0(j)$. If $w_0(j) > 0$ for any j , any linear algorithm stops and gives up because some clause has just had all its literals falsified by the current partial truth assignment. In turn, $w_0(j)$ can be controlled by keeping complementary pairs of clauses out of $C_1(j)$, for all j , since, if a complementary pair exists in $C_1(j)$ for some $j = j'$, then eventually $w_0(j) > 0$ for some $j = j^* > j'$. One way to keep complementary pairs out of $C_1(j)$, for all j , is to prevent an accumulation of unit clauses over time since such an accumulation tends to raise the probability that a complementary pair exists at some

point. The unit clause rule does this by acting like a “pump” that attempts to immediately discharge all clauses which flow into $\mathcal{C}_1(j)$. Unfortunately, it is not usually the case that more than one unit clause can be discharged at a time. Therefore, the unit clause rule is unable to prevent an accumulation in $\mathcal{C}_1(j)$ when $w_1(j) > 1$ over a significant range of j .

One approach to the analysis of a linear unit clause based algorithm is model the clause flows and accumulations as a system of differential equations and determine under what conditions $\max_j \{w_1(j)\} = 1$. Those conditions mark the good performance boundary of the algorithm. This has been tried for the following, most elementary, linear unit clause based algorithm.

UC(\mathcal{F}):

$j \leftarrow 0$

Let $L = \{x, \bar{x} : \exists C \in \mathcal{F} \text{ such that either } x \in C \text{ or } \bar{x} \in C\}$

Repeat the following

If $\mathcal{C}_1(j) \neq \emptyset$ Then randomly choose literal $x \in \mathcal{C}_1(j)$

Otherwise choose x randomly from L

$L \leftarrow L - \{x, \text{complement}(x)\}$

Remove from \mathcal{F} all clauses containing x

Remove from \mathcal{F} all occurrences of $\text{complement}(x)$

$j \leftarrow j + 1$

Until $\mathcal{F} = \emptyset$ or there exist two complementary unit clauses in \mathcal{F}

If $\mathcal{F} = \emptyset$ Then Output(“a solution exists”)

Otherwise Output(“cannot determine whether a solution exists”)

Whether the flows can be modeled according to the approach stated above depends on two things: 1) the clauses in $\mathcal{C}_i(j)$, for any i and j , should be statistically independent and uniformly distributed; 2) conditions whereby markovian processes may be modeled as differential equations should be satisfied. In the first case, clauses entering $\mathcal{C}_i(j)$ are independent of clauses existing in $\mathcal{C}_i(j)$ and of each other, and are uniformly distributed since they were so in $\mathcal{C}_{i+1}(j-1)$ and the chosen literal is selected randomly from the appropriate set of free literals. Also, conditioned on the event that at least one unit clause leaves $\mathcal{C}_1(j)$ when $|\mathcal{C}_1(j)| > 0$, clauses leave $\mathcal{C}_1(j)$ independently as well. In the second case, even as n grows, the flows $w_i(j)$ and $z_i(j)$ are binomially distributed with means that are less than lm/n which is bounded by a constant. This is enough to satisfy the conditions of

Proposition 4.1 in [331] for $m_i(j)$ so there is no loss in modeling the discrete flows and accumulations by a system of differential equations.

With a formal argument we can conclude that Algorithm *UC* determines a solution exists for a given random l -SAT formula with probability bounded from below by a constant if

$$\frac{m}{n} < \frac{2^{l-1}}{l} \left(\frac{l-1}{l-2} \right)^{l-2}.$$

A feature of flow analysis is it reveals mechanisms that suggest other, improved heuristics. For example, since increasing z flows decreases w flows, the following adjustment to Algorithm *UC* is suggested: if there are no unit clauses, choose a literal x randomly from \mathcal{F} , then compare the number of occurrences of x with the number of occurrences of $\text{complement}(x)$ and choose the literal that occurs most often (this tends to increase z flows at the expense of w flows). This isn't quite good enough, however, since $C_i(j)$ clauses are approximately twice as influential as $C_{i+1}(j)$ clauses. This is because, roughly, one clause is accounted for in $z_i(j)$ for every two clauses in $z_{i+1}(j)$. Thus, it is better to compare *weights* of literals where the weight of a literal is given by:

$$\omega(l) = \sum_{C \in \mathcal{F}: l \in C} 2^{-|C|}.$$

An analysis of such a heuristic is not known to us but the following algorithm for 3-SAT, using the original idea of counting clauses, comes fairly close.

UCL(\mathcal{F}):

$j \leftarrow 0$

Let $L = \{x, \bar{x} : \exists C \in \mathcal{F} \text{ such that either } x \in C \text{ or } \bar{x} \in C\}$

Repeat the following

If $\mathcal{C}_1(j) \neq \emptyset$ Then randomly choose literal $x \in \mathcal{C}_1(j)$

Otherwise Do the following

Choose literal y randomly from L

If $\text{occurrences}(\text{complement}(y), \mathcal{C}_3(j)) > \text{occurrences}(y, \mathcal{C}_3(j))$

Then $x \leftarrow \text{complement}(y)$ Otherwise $x \leftarrow y$

$L \leftarrow L - \{x, \text{complement}(x)\}$

Remove from \mathcal{F} all clauses containing x

Remove from \mathcal{F} all occurrences of $\text{complement}(x)$

$j \leftarrow j + 1$

Until $\mathcal{F} = \emptyset$ or there are two complementary unit clauses in \mathcal{F}

If $\mathcal{F} = \emptyset$ Then Output("a solution exists")

Otherwise Output("cannot determine whether a solution exists")

Whereas in the case of Algorithm *UC* applied to random 3-SAT formulas a satisfying assignment can be found with bounded probability when $m/n < 2.66$, Algorithm *UCL* determines a solution exists for a given random 3-SAT formula with probability bounded from below by a constant if $m/n < 2.9$.

There is another improved heuristic suggested by flow analysis. If "pumping" clauses at the bottom level by means of the unit clause rule is effective, putting "pumps" at all levels should be more effective. This amounts to adopting the following strategy for literal selection which is a generalization of the unit clause rule called the *smallest clause rule*: choose a literal from a clause of smallest size. Thus, if there is at least one unit clause, choose from one of them; otherwise, if there is at least one 2-literal clause, choose a literal from a 2-literal clause; otherwise, if there is at least one 3-literal clause, choose a literal from a 3-literal clause, and so on. Putting this heuristic into Algorithm *UC* gives the following:

GUC(\mathcal{F}):

$j \leftarrow 0$

Repeat the following

Let $k = \min\{i : \mathcal{C}_i(j) \neq \emptyset\}$

Let $L = \{x : \exists C \in \mathcal{C}_k(j) \text{ such that } x \in C\}$

Choose x randomly from L

Remove from \mathcal{F} all clauses containing x

Remove from \mathcal{F} all occurrences of *complement*(x)

$j \leftarrow j + 1$

Until $\mathcal{F} = \emptyset$ or there exist two complementary unit clauses in \mathcal{F}

If $\mathcal{F} = \emptyset$ Then Output("a solution exists")

Otherwise Output("cannot determine whether a solution exists")

The effectiveness of "pumping" at all levels is revealed by a flow analysis applied to Algorithm *GUC*. The results are

1. Algorithm *GUC* determines a solution exists for a given random $l - SAT$ formula with probability bounded from below by a constant when

$$\frac{m}{n} < \frac{3.09 * 2^{l-2}}{l+1} \left(\frac{l-1}{l-2} \right)^{l-2} \quad \text{and } 4 \leq l \leq 40.$$

2. Algorithm *GUC* determines a solution exists for a given random $l - SAT$ formula with probability tending to 1 when

$$\frac{m}{n} < \frac{1.845 * 2^{l-2}}{l+1} \left(\frac{l-1}{l-2} \right)^{l-2} - 1 \quad \text{and } 4 \leq l \leq 40.$$

But it is not necessary to "pump" at all levels to get this kind of result nor is it necessary to solve differential equations to determine the accumulation of clauses in $\mathcal{C}_i(j)$, particularly for low values of i . The following algorithm is a slightly stripped-down version of Algorithm *GUC* where literals are taken from one- or two-literal clauses with highest priority.

SC(\mathcal{F}):

$j \leftarrow 0$

Let $L = \{x, \bar{x} : \exists C \in \mathcal{F} \text{ such that either } x \in C \text{ or } \bar{x} \in C\}$

Repeat the following

Let $k = \min\{i : \mathcal{C}_i(j) \neq \emptyset\}$

Let $L' = \{x : \exists C \in \mathcal{C}_k(j) \text{ such that } x \in C\}$

If $k \leq 2$ Then choose x randomly from L'

Otherwise choose x randomly from L

$L \leftarrow L - \{x, \text{complement}(x)\}$

Remove from \mathcal{F} all clauses containing x

Remove from \mathcal{F} all occurrences of $\text{complement}(x)$

$j \leftarrow j + 1$

Until $\mathcal{F} = \emptyset$ or there exist two complementary unit clauses in \mathcal{F}

If $\mathcal{F} = \emptyset$ Then Output("a solution exists")

Otherwise Output("cannot determine whether a solution exists")

It was shown by Chvátal and Reed [92] that Algorithm *SC* determines a solution exists for a given random l -SAT formula with probability tending to 1 when

$$\frac{m}{n} < \frac{2^l}{8l} \left(\frac{l-1}{l-3} \right)^{l-3} \frac{l-1}{l-2}.$$

By adding a limited amount of backtracking to *GUC*, Frieze and Suen proposed an algorithm, called *GUCB*, for 3-SAT that finds a satisfying assignment, with probability tending to 1, when $m/n < 3.003$ [181]. Probabilistic analysis of a backtracking algorithm under the l -SAT distribution can be exceedingly difficult because statistical dependences can easily show up when returning to subformulas after a failure to locate a satisfying assignment. However, Frieze and Suen showed it is possible to carefully manage a limited amount of backtracking so that this does not happen. The following explains how the backtracking in *GUCB* is accomplished. The initial operation of *GUCB* is the same as that of *GUC*. Suppose *GUCB* has successfully completed k iterations and has chosen the sequence of literals $\{x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_k}\}$ and set them to *true*. Suppose $|\mathcal{C}_1(k')| = 0$, $k' < k$ and $|\mathcal{C}_1(j)| > 0$ for all $k' < j \leq k$ so the last iteration that saw no unit clauses was iteration k' . Suppose further that choosing and setting literal $x_{\pi_{k+1}}$ to *true* results in the existence of complementary unit clauses in $\mathcal{C}_1(k+1)$. Then

GUCB backtracks by setting $x_{\pi_{k'}} = x_{\pi_{k'+1}} = \dots = x_{\pi_k} = x_{\pi_{k+1}} = \text{false}$. Corresponding adjustments are made to the clauses and literals of \mathcal{F} . That is, clauses now satisfied are removed, removed clauses now *not* satisfied are reinstated, literals now falsified are removed, and removed literals now *not* falsified and in *non* satisfied clauses are reinstated. After backtracking, *GUCB* continues choosing and setting literals to *true* as before. The algorithm succeeds if all clauses are eliminated. The algorithm fails in two ways: 1) the resetting of literals from *true* to *false* results in a null clause; 2) a complementary pair of unit clauses is encountered before $|\mathcal{C}_1|$ has become 0 after a backtrack. The analysis of *GUCB* is possible because the effect on the distribution of $\mathcal{C}_i(j)$ is slight and because, with probability tending to 1, *GUCB* backtracks at most $\ln^5(n)$ times when $m/n < 3.003$.

Frieze and Suen also apply limited backtracking to *SC* and call the resulting algorithm *SCB*. They show, for $l \geq 4$, *SCB* succeeds, with probability tending to 1, when $m/n < \eta_l 2^l/l$ where $\eta_4 \approx 1.3836$, $\eta_5 \approx 1.504$, and $\lim_{l \rightarrow \infty} \eta_l \approx 1.817$.

This may be compared to the above result for *GUC* which, at $l = 40$, has a performance bound of $m/n < (1.2376)2^{40}/40$ (probability tending to 1), and the above result for *SC* which has a performance bound of $\lim_{l \rightarrow \infty} m/n < (0.9236)2^l/l$ (probability tending to 1).

Although we have concentrated on unit clause elimination algorithms, others have been analyzed as well. Perhaps the most interesting result is on a linear pure literal elimination algorithm, which we call *PL*, and is due to Broder, Frieze, and Upfal [50].

PL(\mathcal{F}):

Repeat the following

Let $L = \{x : \exists C \text{ such that } x \in C, \text{ and } \forall C \in \mathcal{F}, \text{complement}(x) \notin C\}$

If $L = \emptyset$ break out of the loop

Repeat the following for all $x \in L$

 Remove from \mathcal{F} all clauses containing x

If $\mathcal{F} = \emptyset$ Then Output("a solution exists")

Otherwise Output("cannot determine whether a solution exists")

If Algorithm *PL* determines a solution exists, then there is a satisfying assignment for \mathcal{F} : set all the literals x chosen by *PL* to *true*. Algorithm *PL*

cannot determine whether a solution exists for a random 3-SAT formula, with probability tending to 1, if $m/n > 1.63$ ([393]) and determines a solution exists for a random 3-SAT formula, with probability tending to 1, if $m/n < 1.63$ ([50]).

The result is interesting for several reasons.

1. Application of the unit clause rule or smallest clause rule to a random formula preserves a distribution on sets of clauses of the same size that is uniform and random. This property, which simplified the analysis of *UC*, *GUC*, and *SC*, among other algorithms, does not hold when the pure literal rule is applied due to statistical dependency that develops in the resulting subformulas. The analysis of *PL* requires the use of some mathematical tools to show that the dependency is not strong enough to change the outcome much. In particular, Martingale arguments are used to help “squeeze” some of the probability out of the analysis by showing that the number of clauses, number of variables, and number of distinct pure literals that remain after all clauses containing the initial set of pure literals are removed from a random formula is essentially the expected number.
2. This is an example of an algorithm that has a 0-1 threshold property, at least for random 3-SAT formulas. That is, there appears to be some constant c such that *PL* determines a solution exists with probability tending to 1 if $m/n < c$, and cannot determine whether a solution exists with probability tending to 1 if $m/n > c$. This is in contrast to the unit clause based algorithms above for which there are a c_1 and c_2 , with $c_1 < c_2$, such that an algorithm determines a solution exists with probability tending to 1 if $m/n < c_1$, with probability bounded from above and below by a constant if $c_1 < m/n < c_2$, and with probability tending to 0 if $c_2 < m/n$. Exactly why some algorithms have thresholds and some do not is not well understood.
3. Algorithm *PL* is not based on the unit clause rule but is based on the pure literal rule which is an important component of the Davis-Putnam Procedure. The analysis provides some basis for comparison of the relative effectiveness of both rules.

Algorithms for Unsatisfiable Formulas.

The algorithms discussed above are useless for verifying the unsatisfiability of a formula where it is required to prove that no truth assignment is a

satisfying one. Resolution is one popular method capable of doing this and can be extremely effective when certain clause patterns are present, with high probability. The following resolution algorithm has polynomial time complexity:

U-Solve(\mathcal{F})

Repeat the following

If there are two clauses in \mathcal{F} that resolve to $C \notin \mathcal{F}$ with $|C| \leq l$
Then add C to \mathcal{F} .

Until no more clauses can be added to \mathcal{F} .

If $\emptyset \in \mathcal{F}$, Output (“unsatisfiable”)

Otherwise Output (“cannot determine whether \mathcal{F} is unsatisfiable”)

Clearly, Algorithm *U-Solve* cannot be mistaken if it outputs unsatisfiable.

It can be shown that Algorithm *U-Solve* is effective on random l -SAT formulas up to a point. Particular patterns of clauses that *U-Solve* can exploit to produce a null clause are developed as follows:

Let $S_{i,j}$ denote the family of clause sets defined as follows over a set of variables $X = \{x_{11}, x_{21}, \dots, x_{12}, x_{22}, \dots\}$:

$$S_{i,j} = \left\{ \begin{array}{ll} \{\{x_{i1}\}\{\bar{x}_{i1}\}\} & \text{if } j = 1; \\ \{P \cup \{x_{ij}\} : P \in S_{2i-1,j-1}\} \cup \{P \cup \{\bar{x}_{ij}\} : P \in S_{2i,j-1}\} & \text{if } j > 1. \end{array} \right\}$$

Thus,

$$\begin{aligned} S_{1,1} &= \{\{x_{11}\}, \{\bar{x}_{11}\}\} \\ S_{1,2} &= \{\{x_{12}, x_{11}\}, \{x_{12}, \bar{x}_{11}\}, \{\bar{x}_{12}, x_{21}\}, \{\bar{x}_{12}, \bar{x}_{21}\}\} \\ S_{1,3} &= \{\{x_{13}, x_{12}, x_{11}\}, \{x_{13}, x_{12}, \bar{x}_{11}\}, \{x_{13}, \bar{x}_{12}, x_{21}\}, \{x_{13}, \bar{x}_{12}, \bar{x}_{21}\}, \\ &\quad \{\bar{x}_{13}, x_{22}, x_{31}\}, \{\bar{x}_{13}, x_{22}, \bar{x}_{31}\}, \{\bar{x}_{13}, \bar{x}_{22}, x_{41}\}, \{\bar{x}_{13}, \bar{x}_{22}, \bar{x}_{41}\}\} \end{aligned}$$

Over variable set $X \cup \{y_0, y_1, y_2, \dots\}$ define

$$\mathcal{R}_{l,r} = \left\{ \begin{array}{l} \{P \cup \{y_0, \bar{y}_1\} : P \in \mathcal{S}_{1,l-2}\} \cup \\ \left\{ \cup_{i=1}^{r-2} \{P \cup \{y_i, \bar{y}_{i+1}\} : P \in \mathcal{S}_{i+1,l-2}\} \right\} \cup \\ \{P \cup \{y_{r-1}, y_0\} : P \in \mathcal{S}_{r,l-2}\} \cup \\ \{P \cup \{\bar{y}_0, \bar{y}_r\} : P \in \mathcal{S}_{r+1,l-2}\} \cup \\ \left\{ \cup_{i=r+1}^{2r-2} \{P \cup \{y_{i-1}, \bar{y}_i\} : P \in \mathcal{S}_{i+1,l-2}\} \right\} \cup \\ \{P \cup \{y_{2r-2}, \bar{y}_0\} : P \in \mathcal{S}_{2r,l-2}\} \end{array} \right\}$$

For example,

$$\mathcal{R}_{3,3} = \left\{ \begin{array}{l} \{y_0, \bar{y}_1, x_{11}\}, \{y_0, \bar{y}_1, \bar{x}_{11}\}, \{y_1, \bar{y}_2, x_{21}\}, \{y_1, \bar{y}_2, \bar{x}_{21}\}, \\ \{y_2, y_0, x_{31}\}, \{y_2, y_0, \bar{x}_{31}\}, \{\bar{y}_0, \bar{y}_3, x_{41}\}, \{\bar{y}_0, \bar{y}_3, \bar{x}_{41}\}, \\ \{y_3, \bar{y}_4, x_{51}\}, \{y_3, \bar{y}_4, \bar{x}_{51}\}, \{y_4, \bar{y}_0, x_{61}\}, \{y_4, \bar{y}_0, \bar{x}_{61}\} \end{array} \right\}$$

Algorithm *U-Solve*, applied to any l -SAT formula \mathcal{F} that contains a subset of clauses which can be mapped to $\mathcal{R}_{l,r}$ ($r \geq 2$) after renaming of variables, always results in a certificate of unsatisfiability for \mathcal{F} . Therefore, if a random l -SAT formula has such a subset with high probability, then *U-Solve* will conclude unsatisfiability with high probability.

It is straightforward to obtain the probability that there is a subset of clauses that maps to $\mathcal{R}_{l,r}$ for some r . Formula $\mathcal{R}_{l,r}$ contains $r2^{l-1}$ clauses and $r2^{l-1} - 1$ distinct variables, and is minimally unsatisfiable: that is, it is unsatisfiable but removal of any clause produces a satisfiable set. This property is important to the analysis since a minimally unsatisfiable formula can be padded with additional clauses to get other unsatisfiable clause patterns with a much higher ratio of variables to clauses, but the probability that such a padded pattern exists is not greater than the probability that one of its base minimally unsatisfiable sets exist. The existence probability crosses from tending to 0 to tending to 1 at a particular ratio of m/n . This can be estimated by finding conditions that set the average number of minimally unsatisfiable sets to ∞ in the limit. The probability that a particular subset of $r2^{l-1}$ clauses matches the pattern $\mathcal{R}_{l,r}$ for a particular choice of variables is $(r2^{l-1})! / (2^l \binom{n}{l})^{r2^{l-1}}$. There are $\binom{n}{2r^{l-1}-1}$ ways to choose the variables and each of $2^{2r^{l-1}-1}$ ways to complement

the chosen variables presents a pattern that *U-Solve* can handle. Furthermore, any of the $2r^{l-1} - 1$ permutations of those variables also presents a solvable pattern. Finally, there are $\binom{m}{2r^{l-1}}$ ways to choose clauses. Therefore, the average number of $\mathcal{R}_{l,r}$ patterns in \mathcal{F} is the product of the above terms which is about $(l!m/(2^{l-1}n^{(l-1)+1/r2^{l-1}}))^{r2^{l-1}}$. This tends to ∞ when $m/n^{(l-1)} > (n\omega(n))^{1/r2^{l-1}}$ where $\omega(n)$ is any slowly growing function of n . Setting $r = \lfloor \ln^{1+\epsilon}(n) \rfloor$, where ϵ is a small constant (in this case patterns are small enough for a second moment analysis to carry through) is sufficient to give the result that Algorithm *U-Solve* succeeds, with probability tending to 1, on a random l -SAT formula if $m/n^{(l-1)} > 2^{l-1}/l!$.

This analysis shows the importance of the ratio of the number of variables to the number of clauses in a minimally unsatisfiable formula. A higher ratio means a lower cutoff for m/n^{l-1} . We remark that the first results of the above nature appearing in print, as far as we know, are due to Xudong Fu [182]. Fu used a minimally unsatisfiable pattern which he called a *Flower* to achieve the above result. But Flowers have $2^{l-1}(r + l - 1)$ clauses and $2^{l-1}r + l - 1$ variables. Since the difference between clauses and variables in $R_{l,r}$ is 1, whereas the difference is $(2^{l-1} - 1)(l - 1)$ in the case of Flowers, we chose to present $R_{l,r}$ instead.

Improving on the above result is hard. An initial attempt consists of finding minimally unsatisfiable clause patterns where the ratio of variables to clauses is higher than 1 (recall, a ratio of 1 is roughly what can be achieved now), and hopefully close to l . But, the following result, which can be proved by induction on the number of variables in a minimally unsatisfiable set, is a consequence of a theorem of Beame and Pitassi [25], and has also been proven by Kleine Büning is discouraging: If \mathcal{R} is a minimally unsatisfiable formula with r clauses, then \mathcal{R} has less than r variables.

Notice that this does not say anything about the number of literals in a clause: it could be fixed at any l or different for each clause and the result still holds. Moreover, the results above do not say that resolution fails to prove unsatisfiability in polynomial time with high probability when $m/n^{(l-1)} \rightarrow 0$. They suggest only that looking for clause patterns which cause an algorithm such as *U-Solve* to succeed often for such values of m/n is likely not to bear fruit. However, a deeper and illuminating argument shows that resolution can't be successful, with high probability, for some values of m/n .

A number of papers consider the probabilistic performance of resolution for random unsatisfiable l -SAT formulas [25, 87, 182]. The current best (most pessimistic) result is, for $l > 3$, a random unsatisfiable l -SAT formula

has only exponential size resolution proofs, with probability tending to 1, if $m/n^{(l+2)/4-\epsilon} < 1$ for any $\epsilon > 0$. Also, a random 3-SAT formula has only exponential size resolution proofs, with probability tending to 1, if $m/n^{6/5-\epsilon} < 1$, for any $\epsilon > 0$.

Let $\mathcal{P}_{\mathcal{F}}$ be the minimum set of clauses that is the result of repeated applications of the resolution rule starting from an unsatisfiable formula \mathcal{F} and ending with the null clause. A proof fragment $\mathcal{P}_{\mathcal{F}}(C)$ is the minimum subset of clauses in $\mathcal{P}_{\mathcal{F}}$ needed to imply a resolvent $C \in \mathcal{P}_{\mathcal{F}}$. The *complexity* of a resolvent C , referred to as $\text{complexity}(C)$, is the number of clauses that are in both \mathcal{F} and $\mathcal{P}_{\mathcal{F}}(C)$.

The pessimistic result above is due to the fact that resolution does poorly on formulas that are sparse. If $V' \subset V$ is a subset of variables and $\mathcal{C} \subset \mathcal{F}$ is a subset of clauses such that each clause in \mathcal{C} contains at least one variable in V' and all clauses in $\mathcal{F} - \mathcal{C}$ contain no variable in V' , then we say V' encompasses \mathcal{C} . Let $n' < n''$. A CNF formula \mathcal{F} in n variables is called (n', n'', y) – *sparse* if every subset of s variables, $n' < s < n''$, encompasses at most ys clauses in \mathcal{F} .

Formulas that are sparse with both $n' = o(n)$ and $n'' = o(n)$ force a superpolynomial number of resolution steps for the following reason. Any particular size subset \mathcal{C} of clauses taken from such a sparse formula \mathcal{F} must contain a large number of variables that occur exactly once in \mathcal{C} . This forces at least one large clause to exist in $\mathcal{P}_{\mathcal{F}}$. But, almost all “short” resolution proofs contain no long clauses after eliminating all clauses satisfied by a particular small random partial assignment ρ . Moreover, resolution proofs for almost all random unsatisfiable l -SAT formulas, with clauses satisfied by ρ removed are sparse and, therefore, must have at least one long clause. Consequently, almost all unsatisfiable l -SAT formulas have long resolution proofs.

13 Performance Evaluation

The most important measure of a SAT algorithm’s performance remains its practical problem-solving ability. For inputs requiring only one solution, both complete algorithms and incomplete algorithms are applicable. For inputs requiring all solutions or an optimal solution, only complete algorithms will work. The past two decades have seen the proliferation of different algorithms for solving SAT: resolution, local search, nonlinear unconstrained optimization, lagrange method, BDD SAT solver, and multispace search,

among others. Previous experience indicates that these techniques complement rather than exclude each other by being effective for particular instances of SAT.

In this section, we summarize the experimental performance of several typical SAT algorithms on some random instances, DIMACS benchmarks, structured instances, and practical industrial benchmarks. A fuller version of SAT algorithms' benchmarking results will appear in a forthcoming paper, "Algorithms for the Satisfiability (SAT) Problem: Benchmarking," by the same authors.

1. Experiments on Random Formulas

In this section, we give experimental results for the following SAT algorithms in solving random l -SAT formulas and random average l -SAT formulas:

- *SAT1.3*: a sequential *CNF* local search algorithm [212, 226, 214, 217].
- *SAT1.7*: a parallel *CNF* local search algorithm [212, 217].
- *SAT1.13*: a complete *CNF* local search algorithm [212, 217].
- *SAT1.4*: a sequential *DNF* local search algorithm [212].
- *SAT1.8*: a parallel *DNF* local search algorithm [212].
- *SAT1.18*: a complete *DNF* local search algorithm [212].
- *SAT14.6*: a discrete, nonlinear, unconstrained optimization algorithm [212, 214, 218].
- *SAT14.16*: a complete, nonlinear, unconstrained optimization algorithm [212, 214, 218].
- *SAT14.7*: a continuous, nonlinear, unconstrained optimization [212, 214, 223].
- *SAT14.11*: a complete, continuous, nonlinear unconstrained optimization [212, 214, 216].
- *DPL*: a Davis-Putnam algorithm in Loveland form [118].

Table 1: Performance comparison averaged over ten runs between a DPL and some nonlinear unconstrained optimization algorithms on a SUN SPARC 2 workstation for solving 3-SAT problem instances. Time units: seconds. Symbol “S/F” in Column 4 stands for DPL’s *success/failure* to give an answer within a time limit of $120 \times m/n$ seconds, whereas symbol “G/L” stands for the number of times that all the remaining nonlinear unconstrained optimization algorithms hit *global/local* minimum points.

Problems			Execution Time						
m	n	l	S/F	DPL	G/L	SAT1.7	SAT1.13	SAT1.8	SAT14.6
500	500	3	10/0	2.159	10/0	0.013	0.021	0.011	0.013
750	500	3	10/0	2.916	10/0	0.015	0.030	0.013	0.021
1000	500	3	10/0	3.657	10/0	0.035	0.048	0.032	0.031
1250	500	3	9/1	5.797	10/0	0.049	0.064	0.044	0.067
1500	500	3	6/4	9.147	10/0	0.108	0.117	0.089	0.115
1000	500	4	10/0	4.684	10/0	0.026	0.040	0.024	0.024
1500	500	4	10/0	7.960	10/0	0.040	0.075	0.043	0.042
2000	500	4	8/2	10.27	10/0	0.066	0.105	0.062	0.069
2500	500	4	2/8	15.96	10/0	0.074	0.130	0.085	0.109
3000	500	4	1/9	46.33	10/0	0.118	0.201	0.115	0.153
3000	500	5	10/0	16.90	10/0	0.094	0.137	0.082	0.074
4000	500	5	5/5	28.39	10/0	0.119	0.204	0.103	0.144
5000	500	5	0/10	>1200	10/0	0.180	0.253	0.170	0.196
6000	500	5	0/10	>1440	10/0	0.313	0.370	0.267	0.313
7000	500	5	0/10	>1680	10/0	0.591	0.575	0.478	0.604
10000	1000	10	10/0	101.8	10/0	0.047	0.382	0.038	0.049
12000	1000	10	10/0	124.3	10/0	0.073	0.458	0.053	0.052
14000	1000	10	10/0	145.2	10/0	0.077	0.562	0.058	0.063
16000	1000	10	10/0	167.1	10/0	0.098	0.596	0.073	0.078
18000	1000	10	10/0	188.6	10/0	0.136	0.716	0.102	0.095

- *GSAT*: a sequential, greedy local search algorithm [482].
- *IP*: a parallel interior point zero-one integer programming algorithm [310, 308].

Performance Comparison with the DP Algorithm. The execution results of the DPL algorithm and some nonlinear unconstrained optimization algorithms for solving l -SAT instances are given in Table 1. We executed each algorithm ten times and report the average execution times. Because DPL was slow for large size instances, we set a maximum execution time of $120 \times m/n$ seconds as the time limit of its execution. Symbol “S/F” in Column 4 stands for DPL’s *success/failure* in giving an answer within such a time limit. For DPL, the average execution time does not include the

Table 2: Performance comparison between nonlinear unconstrained optimization algorithms running on a SUN SPARC 2 workstation and the **GSAT algorithm** running on a MIPS computer with comparable computing power for the 3-SAT problem instances. Time units: seconds.

Problems			Execution Time					
m	n	l	GSAT	SAT1.7	SAT1.13	SAT1.8	SAT14.6	SAT14.16
215	50	3	0.400	0.019	0.100	0.006	0.020	0.030
301	70	3	0.900	0.054	0.010	0.020	0.020	0.020
430	100	3	6.000	0.336	0.040	0.420	0.050	0.370
516	120	3	14.00	0.596	0.810	1.136	0.410	0.040
602	140	3	14.00	0.260	8.060	0.750	1.990	0.170
645	150	3	45.00	0.102	0.190	0.120	0.040	0.870
860	200	3	168.0	1.776	0.970	0.070	6.710	0.490
1062	250	3	246.0	3.106	20.71	0.070	12.43	0.090
1275	300	3	720.0	8.822	19.66	3.750	19.14	4.510

maximum execution time limit if some of the ten executions were successful; the average execution time was taken as the maximum execution time limit only if all ten executions failed. Symbol “G/L” in Column 6 stands for the number of times that all the remaining nonlinear unconstrained optimization algorithms hit the *global/local* minimum points.

From numerous algorithm executions, we observe that, for random l -SAT instances listed in Table 1, DPL was slower than the rest of the nonlinear unconstrained optimization algorithms. As the input size increases, the number of failures, F , increased quickly. For some slightly large inputs, such as $m = 5000$, $n = 500$, and $l = 5$, all ten algorithm executions failed after a reasonably long time limit. Due to its $O(m^{O(n/l)})$ average run-time complexity, even for some fairly easy instances, such as $m = 10000$, $n = 1000$, and $l = 10$, DPL took an excessive amount of time to find a solution. In comparison, local search and nonlinear unconstrained optimization algorithms were successful for all ten executions. They were able to find a solution to the given instances efficiently.

Table 1 suggests that DPL may not be a suitable candidate for large size random l -SAT instances. This observation should not be generalized to other application cases. In many other applications, as observed by others [70, 71, 127], DPL performed very well.

Performance Comparison with GSAT. Table 2 compares the perfor-

Table 3: Performance comparison between nonlinear unconstrained optimization algorithms running on a SUN SPARC 2 workstation and an interior point zero-one integer programming algorithm running on a *KORBX(R)* parallel/vector computer for solving average 3-SAT problem instances. Time units: seconds. Symbol “S/F” stands for the number of times that IP hits the *global/local* minimum points, whereas symbol “G/L” stands for the number of times that the remaining SAT algorithms hit the *global/local* minimum points.

Problems			Execution Time						
m	n	l	S/F	IP	G/L	SAT1.7	SAT1.13	SAT1.8	SAT14.6
100	50	5	52/0	0.7	10/0	0.001	0.004	0.001	0.001
200	100	5	70/0	1.1	10/0	0.006	0.010	0.006	0.005
400	200	7	69/0	3.5	10/0	0.007	0.014	0.007	0.007
800	400	10	31/0	5.6	10/0	0.009	0.034	0.009	0.003
800	400	7	20/0	7.8	10/0	0.014	0.032	0.014	0.009
1000	500	10	49/0	7.4	10/0	0.012	0.037	0.012	0.006
2000	1000	10	10/0	18.5	10/0	0.032	0.091	0.032	0.019
2000	1000	7	50/0	21.5	10/0	0.056	0.099	0.056	0.055
2000	1000	3	49/1	50.4	10/0	2.657	0.162	2.657	3.917
4000	1000	4	1/1	1085.4	10/0	10.63	11.07	10.63	6.826
4000	1000	10	10/0	25.1	10/0	0.055	0.189	0.055	0.044
8000	1000	10	10/0	38.0	10/0	0.219	0.456	0.219	0.254
16000	1000	10	10/0	66.4	10/0	0.603	1.042	0.603	0.625
32000	1000	10	10/0	232.4	10/0	1.701	2.720	1.701	1.611

mance between some local search and nonlinear unconstrained optimization algorithms running on a SUN SPARC 2 workstation and *GSAT* [482] running on a MIPS computer with comparable computing power [481]. Since *GSAT* is essentially a version of sequential local search (*i.e.*, *SAT1*) algorithm, for solving 3-SAT instances generated from the same input model used in [392], local search and nonlinear unconstrained optimization algorithms performed approximately tens to hundreds times faster than *GSAT*. Among them, parallel DNF local search (*SAT1.8*) algorithm and complete nonlinear unconstrained optimization (*SAT14.16*) were the best.

Performance Comparison with Interior Point Zero-One Integer Programming Algorithm. Recently, Kamath *et al.* used an interior point zero-one integer programming algorithm to solve SAT [310, 308]. They implemented their algorithm in FORTRAN and C languages and ran the algorithm on a *KORBX(R)* parallel/vector computer with instances generated

Table 4: **WSAT** (*GSAT* with random walk)'s real execution performance for hard random 3-SAT problem instances on an SGI Challenge with a 70 MHz MIPS R4400 processor. Time unit: seconds [485].

Problems		<i>GSAT</i>			Walk		
<i>n</i>	<i>m</i>	time	flips	R	time	flips	R
100	430	.4	7554	8.3	.2	2385	1.0
200	860	22	284693	143	4	27654	1.0
400	1700	122	2.6×10^6	67	7	59744	1.1
600	2550	1471	30×10^6	500	35	241651	1.0
800	3400	*	*	*	286	1.8×10^6	1.1
1000	4250	*	*	*	1095	5.8×10^6	1.2
2000	8480	*	*	*	3255	23×10^6	1.1

from the average 3-SAT input model. The *KORBX(R)* parallel computer operates in scalar mode at approximately 1 MFlops and at 32 MFlops with full vector concurrent mode. Their execution results are given in Columns 4 and 5 of Table 3.

We ran local search and nonlinear unconstrained optimization algorithms for the same instances (listed in [310, 308]) on a SUN SPARC 2 workstation. The results are given in Table 3. Apparently, as compared to the interior point zero-one integer programming algorithm running on a parallel computer, in addition to improved global convergence, local search and nonlinear unconstrained optimization algorithms were much simpler and achieved several orders of magnitude of performance improvements in terms of computing time.

2. Experiments on Hard Random Formulas

We compare the performance of two local search algorithms and a tabu search algorithm for the hard random 3-SAT problem instances generated from the **mwff** generator. All three programs were written in *C*. Table 4 give the real execution performance of the **WSAT** (*GSAT* with random walk) on an SGI Challenge with a 70 MHz MIPS R4400 processor [485, 484].

Table 5: **WSAT** (*GSAT* with random walk)'s real execution performance for hard random 3-SAT problem instances on a PC. Time unit: seconds [379].

<i>n</i>	<i>m</i>	inst.	time	flips	solved	ratio
100	430	500	0.18	2803	88%	31,85
200	860	500	1.99	18626	73%	255,85
400	1700	500	15.03	204670	100%	2046,70
600	2550	500	19.59	250464	62%	4013,85
800	3400	500	140.61	1809986	67%	26854,39
1000	4250	500	369.88	4633763	57%	81009,84
2000	8240	50	3147.26	26542387	16%	1658899,19

Tables 5 and 6 show the experimental results of *WSAT* and *TSAT* (Tabu search for SAT) programs written in *C* under Linux 1.1.59 for PC [379]. On a same machine, Mazure, Sais, and Gregoire compared the *GSAT* with *TSAT* and found that *TSAT* was more efficient in most cases. In addition, *TSAT* was able to solve more problem instances compared to the *GSAT*. The testing for *TSAT* for large size example with $n = 2000$ and $m = 8240$, however, was terminated at $n/m = 4.12$, before entering into the hard region of the random 3-SAT instances.

The performance of the *SAT1.5* algorithm [214, 224] (Section 6.6) is shown in Table 7. For hard problem instances in the transition region [392], *SAT1.5* can solve large-size SAT problem instances efficiently. It took *WSAT* on average 3,255 seconds to solve the $n = 2,000$ and $m = 8,480$ instances on an SGI Challenge with a 70 MHz MIPS R4400 processor. On a SUN SPARC 20 workstation, the *SAT1.5* algorithm was able to solve the same problem instance in some 530 seconds on average [224, 225]. For hard, large size SAT problem instances with $n > 5,000$, *SAT1.5* algorithm was able to handle the problems comfortably.

3. Experiments on Structured Instances

We now take a look at the performance of SAT algorithms for some structured instances.

Table 6: TSAT's real execution performance for hard random 3-SAT problem instances on a PC. Time unit: seconds [379].

<i>n</i>	<i>m</i>	inst.	time	flips	solved	ratio
100	430	500	0.11	1633	93%	17,60
200	860	500	0.73	9678	74%	130,78
400	1700	500	11.51	145710	100%	1457,10
600	2550	500	13.92	167236	65%	2580,80
800	3400	500	99.45	1143444	71%	16150,34
1000	4250	500	292.10	3232463	62%	51802,29
2000	8240	50	3269.15	29415465	40%	735386,63

Instances Generated from the *N*-Queens Problem. To assess the performance of local search and nonlinear unconstrained optimization algorithms with *non-binary* instances, we also tested SAT instances generated from instances of the *n*-queens problem. Figure 29 compares the performance between DP and nonlinear unconstrained optimization algorithms. It also compares the performance between DP and *SAT14.11* [216], a complete, continuous unconstrained optimization algorithm. Due to expensive floating point computations, the execution time of *SAT14.11* is higher than those of other discrete local search and nonlinear unconstrained optimization algorithms.

DIMACS Instances. For the same SAT formulas generated from instances of the Boolean inference problem [309], the performance of *SAT1.7* [217], a parallel local search algorithm, and a simple backtracking algorithm [336] is shown in Tables 8 and 9, respectively. An algorithm may be effective for only one type of input. The results suggest that it can be much more efficient if we use several different types of algorithms to handle the same inputs simultaneously.

In Table 10, we compare *A*₂ [548] with *WSAT*, *GSAT*, and Davis-Putnam's algorithm in solving the circuit diagnosis benchmark problems. We present average execution times and average number of iterations of *A*₂ as well as published average execution times of *WSAT*, *GSAT* and Davis-Putnam's method [485]. We did not attempt to reproduce the reported results of *GSAT* and *WSAT*, since the results may depend on initial conditions, such

Table 7: Real execution performance of the **SAT1.5 algorithm** for hard random 3-SAT problem instances on a SUN SPARC 20 workstation. For each problem, 30 random instances were tested. The minimum (T_{min}), maximum (T_{max}), and average (T_{mean}) execution times were recorded. “S” indicates the number of success cases of finding solutions within the time limit ($T\text{-limit}$). Time unit: second.

n	m	m/n	S/30	T_{min}	T_{mean}	T_{max}	T-limit
1000	4230	4.2300	20/30	6.04	206.90	956.15	1000
1000	4240	4.2400	15/30	0.55	223.26	891.11	1000
1000	4250	4.2500	14/30	1.69	88.370	454.79	1000
1000	4260	4.2600	10/30	24.4	243.25	914.35	1000
2000	8460	4.2300	12/30	115.8	779.44	2069.9	3000
2000	8480	4.2400	14/30	17.64	530.32	1360.9	3000
2000	8500	4.2500	7/30	58.09	789.35	1677.6	3000
2000	8510	4.2550	9/30	59.08	840.33	2322.8	3000
3000	12690	4.2300	12/30	430.6	1787.2	2876.4	5000
3000	12700	4.2333	18/30	122.5	2101.5	4479.4	5000
3000	12720	4.2400	11/30	270.6	1503.6	3840.7	5000
3000	12740	4.2467	12/30	229.1	2062.9	4807.4	5000
3000	12680	4.2267	15/30	356.1	2788.3	9510.2	10000
3000	12700	4.2333	11/30	503.3	3681.1	8247.9	10000
3000	12720	4.2400	15/30	30.09	2300.3	7002.3	10000
3000	12740	4.2467	8/30	563.3	2620.5	5330.9	10000
4000	16920	4.2300	11/30	739.83	4064.5	11498.2	12000
4000	16930	4.2325	10/30	1733.5	5472.0	10187.8	12000
4000	16940	4.2350	7/30	571.20	1948.9	4768.92	12000
4000	16960	4.2400	10/30	294.80	3709.0	9921.77	12000
5000	21150	4.2300	8/30	2024.7	3867.9	8134.81	9000
5000	21175	4.2350	6/30	1640.1	2982.7	4193.68	9000
5000	21200	4.2400	3/30	2935.8	4435.7	6357.65	9000
5000	21225	4.2450	4/30	3883.5	6025.6	10980.9	15000
10000	41000	4.1000	30/30	294.44	1315.6	3849.38	20000
10000	41800	4.1800	8/18	4294.5	8387.9	16654.8	20000
10000	42000	4.2000	4/30	963.53	5877.8	12020.3	20000
10000	42200	4.2200	2/30	9270.6	14241.9	19213.4	20000

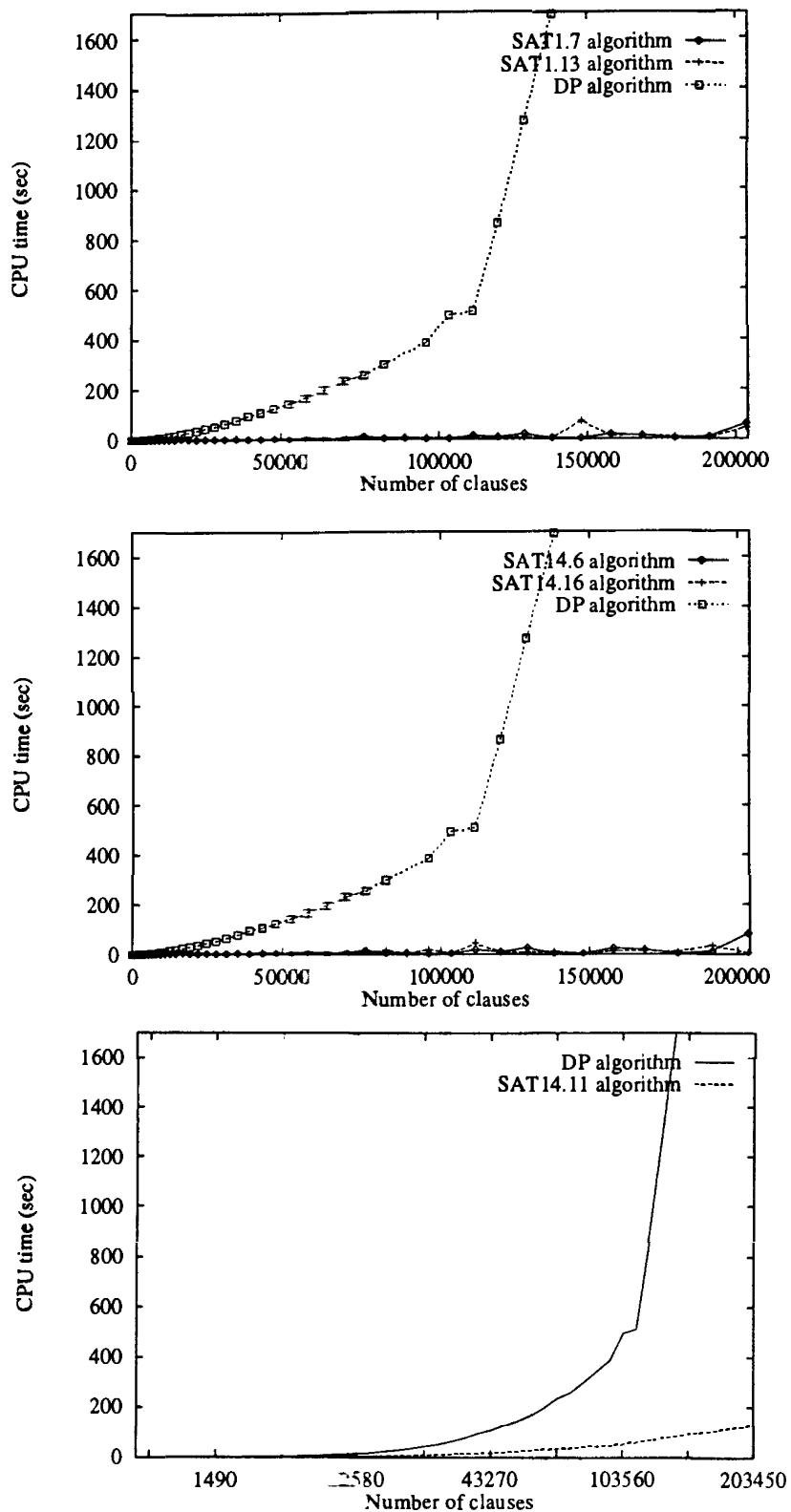


Figure 29: Comparison of $\square\Box$ with SAT1.7, SAT1.13, SAT14.6, SAT14.16, and SAT14.11 for solving SAT instances generated from CSP instances

Table 8: Performance of **SAT1.7** on a SUN SPARC 10 workstation. Time Units: seconds.

Problems			Ten Trials		Execution Time		
Name	<i>m</i>	<i>n</i>	Global	SAT	Min	Mean	Max
ii16a1.sat	1650	19368	10/10	YES	10.320	125.51	417.42
ii16b1.sat	1728	24792	10/10	YES	0.6100	6.1130	28.760
ii16c1.sat	1580	16467	10/10	YES	0.5400	1.8740	3.7500
ii16d1.sat	1230	15901	10/10	YES	0.4900	1.3810	2.8200
ii16e1.sat	1245	14766	10/10	YES	0.5300	0.9720	1.4800
ii16a2.sat	1602	23281		N/A			
ii16b2.sat	1076	16121	10/10	YES	1.9000	39.118	102.60
ii16c2.sat	924	13803	10/10	YES	0.3500	14.109	41.650
ii16d2.sat	836	12461	10/10	YES	0.3300	19.840	52.410
ii16e2.sat	532	7825	10/10	YES	0.5000	6.8830	21.980
ii32a1.sat	459	9212	10/10	YES	0.3600	3.5740	10.330
ii32b1.sat	228	1374	10/10	YES	0.1100	0.7390	1.6700
ii32b2.sat	261	2558	10/10	YES	0.1000	1.9040	4.4700
ii32b3.sat	348	5734	10/10	YES	1.6400	10.559	19.330
ii32b4.sat	381	9618	10/10	YES	0.5100	2.3060	4.7800
ii32c1.sat	225	1280	10/10	YES	0.0100	0.1150	0.4800
ii32c2.sat	249	2182	10/10	YES	0.0600	0.3980	0.9000
ii32c3.sat	279	3272	10/10	YES	0.6900	5.4900	16.850
ii32c4.sat	759	20862	10/10	YES	5.5200	361.80	1496.3
ii32d1.sat	332	2703	10/10	YES	0.2200	1.0680	3.1000
ii32d2.sat	404	5153	10/10	YES	0.2100	0.9140	2.1800
ii32d3.sat	824	19478	10/10	YES	1.7100	49.522	109.52
ii32e1.sat	222	1186	10/10	YES	0.0200	0.3260	1.0700
ii32e2.sat	267	2746	10/10	YES	0.0400	0.1130	0.3400
ii32e3.sat	330	5020	10/10	YES	0.4500	5.2700	13.910
ii32e4.sat	387	7106	10/10	YES	0.2700	10.734	46.750
ii32e5.sat	522	11636	10/10	YES	0.4900	23.424	84.470

Table 9: Performance of a simple backtracking algorithm on a SUN SPARC 10 workstation. Time Units: seconds.

Name	<i>m</i>	<i>n</i>	SAT	Time	Name	<i>m</i>	<i>n</i>	SAT	Time
ii16a1.sat	1650	19368	YES	1.285	ii16b1.sat	1728	24792	YES	1.490
ii16c1.sat	1580	16467	N/A	1.956	ii16d1.sat	1230	15901	YES	1.660
ii16e1.sat	1245	14766	N/A	2.125					
ii16a2.sat	1602	23281	YES	1.430	ii16b2.sat	1076	16121	YES	1.505
ii16c2.sat	924	13803	YES	2.016	ii16d2.sat	836	12461	YES	1.665
ii16e2.sat	532	7825	N/A	2.051					
ii32a1.sat	459	9212	YES	1.160	ii32b1.sat	228	1374	YES	1.035
ii32b2.sat	261	2558	YES	1.035	ii32b3.sat	348	5734	YES	1.240
ii32b4.sat	381	9618	YES	1.285					
ii32c1.sat	225	1280	YES	0.000	ii32c2.sat	249	2182	YES	1.325
ii32c3.sat	279	3272	YES	1.240	ii32c4.sat	759	20862	YES	1.695
ii32d1.sat	332	2703	YES	1.035	ii32d2.sat	404	5153	YES	1.525
ii32d3.sat	824	19478	YES	1.755					
ii32e1.sat	222	1186	YES	0.000	ii32e2.sat	267	2746	YES	1.035
ii32e3.sat	330	5020	YES	1.565	ii32e4.sat	387	7106	YES	1.615
ii32e5.sat	522	11636	YES	1.655					

Table 10: Comparison of A_2 's execution times in seconds averaged over 10 runs with respect to published results on some of the circuit diagnosis problems in the DIMACS archive, including the best known results obtained by WSAT, GSAT, and Davis-Putnam's algorithm [485].

Problem Id	<i>n</i>	<i>m</i>	A_2			WSAT	GSAT	DP
			SS 10/51	SGI	# Iter.			
ssa7552-038	1501	3575	0.228	0.235	7970	2.3	129	7
ssa7552-158	1363	3034	0.088	0.102	2169	2	90	*
ssa7552-159	1363	3032	0.085	0.118	2154	0.8	14	*
ssa7552-160	1391	3126	0.097	0.113	3116	1.5	18	*

- A_2 : Sun SparcStation 10/51 and a 150-MHz SGI Challenge with MIPS R4400;
- GSAT, WSAT and DP: SGI Challenge with a 70 MHz MIPS R4400.

Table 11: Comparison of A_2 's execution times in seconds averaged over 10 runs with published results on circuit synthesis problems from the DIMACS archive, including the best known results obtained by GSAT, integer programming, and simulated annealing [485].

Problem Id.	n	m	A_2			GSAT	Integer Prog.	SA
			SS 10/51	SGI	# Iter.			
ii16a1	1650	19368	0.122	0.128	819	2	2039	12
ii16b1	1728	24792	0.265	0.310	1546	12	78	11
ii16c1	1580	16467	0.163	0.173	797	1	758	5
ii16d1	1230	15901	0.188	0.233	908	3	1547	4
ii16e1	1245	14766	0.297	0.302	861	1	2156	3

- A_2 : Sun SparcStation 10/51 and a 150-MHz SGI Challenge with MIPS R4400;
- GSAT and SA: SGI Challenge with a 70 MHz MIPS R4400;
- Integer Programming: VAX 8700.

as the seeds of the random number generator and other program parameters. We ran A_2 on an SGI Challenge⁷ so that our timing results can be compared to those of *GSAT* and *WSAT*. Our results show that A_2 is approximately one order of magnitude faster than *WSAT*.

In Table 11, we compare A_2 [548] with the published results of *GSAT*, integer programming and simulated annealing on the circuit synthesis problems [485]. Our results show that A_2 performs several times faster than *GSAT*.

In Table 12, we compare the performance of the three versions of *DLM* with some of the best known results of *GSAT* on circuit-synthesis, parity-learning, some artificially generated 3-SAT, and some of the hard graph coloring problems. The results on *GSAT* are from [486], which are better than other published results. Our results show that *DLM* is consistently faster than *GSAT* on the “*ii*” and “*par*” inputs, and that A_1 is an order-of-

⁷Based on a single-CPU 150-MHz SGI Challenge with MIPS R4400 at the University of Illinois National Center for Supercomputing Applications, we estimate empirically that it is 15.4% slower than a Sun SparcStation 10/51 for executing A_2 to solve SAT benchmark problems. However, we did not evaluate the speed difference between a 150-MHz SGI Challenge and a 70-MHz SGI Challenge on which *GSAT* and *WSAT* were run.

Table 12: Comparison of DLM's execution times in seconds averaged over 10 runs with the best known results obtained by *GSAT* [486] on the circuit-synthesis, parity-learning, artificially generated 3-SAT instances, and graph coloring problems from the DIMACS archive.

Problem Identification	<i>n</i>	<i>m</i>	<i>A</i> ₁		GSAT	
			SS 10/51	Success Ratio	Time	Success Ratio
aim-100-2_0-yes1-1	100	200	0.19	10/10	1.96	9/10
aim-100-2_0-yes1-2	100	200	0.65	10/10	1.6	10/10
aim-100-2_0-yes1-3	100	200	0.19	10/10	1.09	10/10
aim-100-2_0-yes1-4	100	200	0.10	10/10	1.54	10/10
			<i>A</i> ₂		GSAT	
ii32b3	348	5734	0.31	10/10	0.6	10/10
ii32c3	279	3272	0.12	10/10	0.27	10/10
ii32d3	824	19478	1.05	10/10	2.24	10/10
ii32e3	330	5020	0.16	10/10	0.49	10/10
par8-2-c	68	270	0.06	10/10	1.33	10/10
par8-4-c	67	266	0.09	10/10	0.2	10/10
			<i>A</i> ₃		GSAT	
g125.17	2125	66272	1390.32	10/10	264.07	7/10
g125.18	2250	70163	3.197	10/10	1.9	10/10
g250.15	3750	233965	2.798	10/10	4.41	10/10
g250.29	7250	454622	1219.56	9/10	1219.88	9/10

- *A*₁, *A*₂, *A*₃: Sun SparcStation 10/51
- GSAT: SGI Challenge (model unknown)

Table 13: Execution times in CPU seconds over 10 runs of A_3 to solve some of the more difficult DIMACS benchmark problems.

Prob. Id.	Succ. Ratio	Sun SS 10/51 (Seconds)		
		Avg.	Min.	Max.
par8-1	10/10	4.780	0.133	14.383
par8-2	10/10	5.058	0.100	13.067
par8-3	10/10	9.903	0.350	21.150
par8-4	10/10	5.842	0.850	16.433
par8-5	10/10	14.628	1.167	34.900
par16-1	5/10	11172.8	4630.6	20489.1
par16-2	1/10	856.9	856.9	856.9
par16-3	1/10	20281.6	20281.6	20281.6
par16-4	3/10	3523.1	1015.0	7337.9
par16-5	1/10	13023.4	13023.4	13023.4
par16-1-c	10/10	398.1	11.7	1011.9
par16-2-c	10/10	1324.3	191.0	4232.3
par16-3-c	10/10	987.2	139.8	3705.2
par16-4-c	10/10	316.7	5.7	692.66
par16-5-c	10/10	1584.2	414.5	3313.2
hanoi4	1/10	476.5	476.5	476.5
f1000	10/10	126.8	4.4	280.7
f600	10/10	16.9	2.1	37.2
f2000	10/10	1808.6	174.3	8244.7
Program parameters				
Flat region limit = 50; λ reset interval = 10,000; operation: $\lambda = \lambda/1.5$.				
Problem group	par-16-[1-5]	test par problems	f	hanoi4
Tabu length	100	50	50	50
Increment of λ	1	1/2	1/16	1/2

magnitude faster than *GSAT* on some “aim” inputs.

We are designing new strategies to improve A_3 ’s [548] performance. Tables 13 shows some preliminary but promising results of A_3 on some of the more difficult but satisfiable DIMACS benchmark inputs.

4. Experiments on Practical Industrial Benchmarks

Performance of the SAT-Circuit Solver with Partitioning Preprocessing. We compared Gu and Puri’s SAT solver (having a partitioning preprocessing) [221] with Lavagno’s algorithm [339] and Vanbekbergen’s al-

gorithm [539] for solving industrial asynchronous circuit design benchmarks, including the HP and Philips benchmarks.

The experimental results indicate that, as compared to the previous methods [339, 539], the SAT-Circuit solver with partitioning preprocessing achieves many orders of magnitude of performance improvement in terms of computing time, in addition to a reduced implementation area. For example, in a large circuit *mr0*, SAT-Circuit took 2.80 seconds to solve the problem and yielded a two-level implementation area with 41 literals.⁸ In contrast, Lavagno *et al.*'s algorithm took 1,084.5 seconds and an area of 86 literals. For this example, Vanbekbergen *et al.*'s algorithm could not yield a solution within 3,600 seconds and aborted due to backtracking limit. For another benchmark circuit *mmu0*, SAT-Circuit solved it in 0.87 seconds, as compared to a pre-aborted 406.3 seconds for Vanbekbergen *et al.*'s approach [539].

Performance of a BDD SAT Solver with Partitioning Preprocessing. The BDD SAT-Circuit solver was implemented in C language. In this case, Gu and Puri tested their BDD SAT-Circuit solver with its ability to find all solutions (therefore, an optimal solution) for a large number of industrial asynchronous circuit benchmarks including the HP and Philips benchmarks [221, 448]. They also compared the performance of their BDD SAT-Circuit solver with the well known Lavagno *et al.*'s [339] asynchronous circuit design technique. The results of these experiments are given in Table 14 and Table 15.

Table 14 compares the execution time of the BDD SAT solver with the execution time of a simple backtracking SAT algorithm of [336]. The experimental results are given for SAT instances generated from Gu and Puri's SAT formula partitioning preprocessor [221]. Since the BDD SAT-Circuit solver yielded all the solutions, they normalized the execution time of the backtracking algorithm for all the truth assignment. The experimental results (Table 14) show that the BDD SAT solver outperforms the backtracking SAT technique for the practical SAT instances representing asynchronous circuit design.

They also calculated the implementation area of the designed circuits. Table 15 compares their BDD SAT solver with the well known Lavagno *et al.*'s asynchronous circuit design technique [339]. The BDD SAT-Circuit solver yielded reduced circuit implementation area than Lavagno *et al.*'s

⁸Literal here is a standard unit measuring layout area.

Table 14: Experimental results comparing the **BDD SAT-Circuit solver** and a backtracking SAT algorithm, both with SAT formula partitioning pre-processing, on practical asynchronous circuit benchmarks on a SUN SPARC-2 workstation. Time unit: second.

STG Benchmark Name	BDD SAT Solver	Backtracking satisfiability testing	STG Benchmark Name	BDD SAT Solver	Backtracking satisfiability testing
Mr0	58.3	>3,600	Mmu1	28.1	>3,600
SbufRamWr	32.7	>3,600	Vbe4a	1.95	>3,600
NakPa	0.53	5.4	RamRdSbuf	0.25	76.8
AlexNonFc	0.37	0.96	SbufSndPkt2	0.37	88.06
SbufSndCtl	18.27	353.6	AtoD	0.15	11.88
Pa	0.05	4.50	WrData	0.14	0.24
Fifo	0.05	0.10	SbufRdCtl	0.09	0.10
NoUsc	0.09	0.16	VbeEx2	3.94	0.80
NoUscSer	0.06	0.07	SendrDone	0.05	0.16
VbeEx1	0.03	0.04			

algorithm for almost all the circuits in the benchmark set [339]. Lavagno *et al.*'s method yields a total area of 449 literals in 1298.5 seconds. In comparison, for the same benchmarks, the BDD SAT solver achieved an area of 379 literals in 145.7 seconds. In addition, Lavagno *et al.*'s method was unable to solve some benchmark circuits, such as *Pa* and *AlexNonFc*. These results show that, as compared to existing techniques, the BDD SAT solver is capable of achieving an average of 20% reduction in implementation area for all the benchmarks. According to critical industrial evaluations, this BDD SAT solver offers a practical solution for complex industrial asynchronous circuit design problems.

14 Applications

Practical application problems are the driving forces for SAT research. They provide the ultimate benchmarks to test SAT algorithms and techniques. An effective SAT algorithm in one application problem will shed light on solving problems in other application areas.

The SAT problem has *direct* applications in mathematical logic, artificial intelligence, VLSI engineering, and computing theory. It also has *indirect* applications through other transferable problems, e.g., constraint satisfaction

Table 15: Comparison of Implementation area and design time of the BDD SAT-Circuit solver (with SAT formula partitioning preprocessing) and Lavagno *et al.*'s technique for practical asynchronous circuit benchmarks on a SUN SPARC-2 workstation. Time unit: second.

Benchmark Name	Benchmark		BDD SAT Solver			Lavagno and Moon [339]		
	Initial no. of states	Initial no. of signals	Final no. of signal	Circuit Area (literal)	CPU time sec.	Final no. of signal	Circuit Area (literal)	CPU time sec.
Mr0	302	11	15	41	58.36	13	86	1084.5
Mmul	82	8	10	38	28.16	10	37	47.8
SbufRamWr	58	10	12	47	32.79	12	35	54.6
Vbe4a	58	6	8	30	1.95	8	41	5.5
NakPa	56	9	10	25	0.53	10	41	20.8
RamRdSbuf	36	10	11	25	0.25	11	23	65.2
SbufSndPkt2	24	6	7	21	0.37	7	14	8.6
SbufSndCtl	21	6	7	17	0.37	8	43	3.4
AtoD	20	6	8	30	18.27	7	19	2.9
Pa	20	6	7	14	0.15	Internal State Error		
WrData	16	4	5	18	0.05	5	21	0.9
Fifo	16	4	5	15	0.14	5	15	0.7
SbufRdCtl	14	6	7	16	0.05	7	15	1.5
NoUsc	12	3	4	12	0.09	4	14	0.5
VbeEx2	12	3	4	12	0.09	4	21	0.5
NoUscSer	8	2	4	18	3.94	4	11	0.4
AlexNonFc	8	3	4	9	0.06	Non-Free-Choice STG		
SendrDone	7	3	4	8	0.05	4	6	0.4
VbeEx1	5	2	3	6	0.03	3	7	0.3

problems and constrained optimization problems [234]. Due to the *UniSAT* models, some application problems in the real space are related to SAT as well. In the following, we list some applications that can be formulated as solved as instances of SAT.

- *Mathematics*: finding n -ary relations such as transitive closure [68], detecting graph and subgraph isomorphisms [106, 382, 384, 409, 452, 529, 568], the graph coloring problem [58, 250, 378, 384], mathematical cryptology [418, 457], the automata homomorphism problem [201], finding spanning trees and Euler tours in a graph [406], solving the traveling salesman problem [294, 295, 340, 410], and logical arithmetic [94].
- *Computer science and artificial intelligence*: the constraint satisfaction problem [13, 194, 211, 369, 461], the n -queens problem [194, 249, 495], extended inference [22], logical programming [97, 99, 140, 337], abductive inference for synthesizing composite hypotheses [304], semantic information processing [22, 162, 407], puzzles and cryptoarithmetic [192, 249, 282, 376, 377, 404], truth maintenance [123, 125, 128, 139, 380], production system [285, 390, 391], the soma cube and instant insanity problem [194], theorem proving [276, 323, 437, 567], and neural network computing [13, 14, 130, 258, 362, 279].
- *Machine vision*: image matching problem [22, 89, 460, 563], line and edge labeling problems [77, 160, 525, 551, 571], stereopsis, scene analysis and semantics-based region growing [22, 77, 159, 160, 161, 525, 551], the shape and object matching problem [68, 116, 254], syntactic shape analysis [117, 253, 317], shape from shading problem [12, 51, 175, 269, 270, 272, 271, 281, 373, 422], and image restoration [196].
- *Robotics*: related vision problem [89, 280], packing problem [134], and trajectory and task planning problems [47, 153].
- *Computer-aided manufacturing*: task planning [403], design [401, 402], solid modeling, configuring task [176], design cellular manufacturing system, scheduling [165, 364], and 3-dimensional object recognition [237, 271].
- *Database systems*: operations on objects [528, 531], database consistency maintenance, query-answering and redundancy-checking, query optimization [79, 528], concurrency control [32, 155, 368], distributed

database systems [188], truth and belief maintenance [123, 125, 128, 139, 380], the relational homomorphism problem [249, 528], and knowledge organization for recognition system [251].

- *Text processing:* optical character recognition [91, 397, 512], character constraint graph model [277], printed text recognition [21, 277], handwritten text recognition [493], automatic correction of errors in text [530].
- *Computer graphics:* construction of 2-dimensional pictures and 3-dimensional graphical objects from constraints, reasoning of the geometrical features of 3-dimensional objects [56, 183].
- *Integrated circuit design automation:* circuit modeling [76, 519], logic minimization [261], state assignment [539, 540], state minimization [207, 451], asynchronous circuit synthesis [221, 448, 447, 449], I/O encoding for sequential machines [468], power dissipation estimation [136], logic partitioning [86, 144, 335, 408, 466], circuit layout and placement [11, 37, 48, 98, 113, 135, 241, 504], scheduling and high-level synthesis [49, 333, 419], pin assignment [46, 465], floorplanning [428, 513], interconnection analysis [142, 143], routing [1, 72, 132, 133, 240, 343, 417, 454, 458, 480], compaction [69, 141, 252, 313, 334, 354, 473, 490, 537], performance optimization [307, 324, 375, 463, 513], testing and test generation [137, 287, 336, 147, 429], and verification [322, 515, 526]. Please also see: Jun Gu, *Satisfiability Problems in VLSI Engineering* [222], 1996.
- *Computer architecture design:* instruction set optimization [4, 115, 208, 290, 446, 450], computer controller optimization [28, 34, 314, 366, 445, 451], arithmetic logic circuit design [78], compiler system optimization [7, 353], scheduling [38, 138, 189, 190, 238, 346], fault-tolerant computing [24, 268, 20], task partitioning and assignment [40, 41, 112, 283, 491], load balancing [357, 405, 570], real time systems [262, 289, 325, 507, 508, 517], data flow consistency analysis [7], data module assignment in memory system [7], and parallel and distributed processing [453, 478].
- *High-speed networking:* contact the authors.
- *Communications:* contact the authors.

- *Security:* contact the authors.

In other areas such as industrial (chemical, transportation, construction, nuclear) engineering, management, medical research, social sciences, there are numerous SAT /CSP applications.

15 Future Work

A number of future research directions for the satisfiability problem have been discussed recently. They are further emphasized in the 1996 DIMACS satisfiability workshop.

General Boolean Expressions and Evaluation. Many practical application problems are expressed as Boolean satisfiability problems by a compact set of general Boolean functions. Although the transformation of a general Boolean expression into *CNF* can be done in polynomial time, it will result in a substantially larger clause-form representation [195, 425]. While this may not be critical in complexity theory, it will have serious impact on the time to solve these problems. To this end, efficient representation and manipulation of general Boolean functions is crucial to solving practical application problems.

Theoretical Issues. Recent research on SAT has brought up some interesting theoretical problems, such as the average time complexity analysis [26, 217, 236, 276, 374, 433], determining satisfiable-unsatisfiable boundary [110, 316, 392], global convergence and local convergence rate [219, 223], and the structure and hardness of input models [103, 171, 199]. Some of the problems, e.g., the average time complexity analysis, are extremely difficult [345]. So far only some preliminary efforts based on simplified assumptions were given [50, 226, 219, 223].

One of the recent efforts to solve SAT formulas is to find subclasses for which the problem is solvable in polynomial time [154, 187]. Future work in this direction aims at building hierarchies of formulae classes, analyzing the properties of such hierarchies, and qualitative evaluation of the hierarchies.

SAT Algorithm Development. The development of new algorithms and improved techniques for satisfiability testing has been a long-term effort of the research community and the industry. From computation/efficiency point of view, specific data structures and implementation details of SAT

algorithms are crucial. The algorithm space shows a number of asymmetrical and irregular places, implying further opportunity for new SAT algorithm development.

From an experimental point of view, it is difficult to find a super algorithm that performs well for a wide range of SAT instances. Existing SAT algorithms complement rather than exclude each other by being effective for particular problem instances. One of the future directions is to continue the development of the *Multi-SAT* algorithm, integrating different algorithms using a cluster of computers [225] (Section 11.5). Computer hardware and memory space are becoming increasingly inexpensive. If one can trade hardware for improved performance, it can show a promising approach (in fact, trading memory space for speed was a basic design philosophy behind the RISC computer architectures).

For important practical applications, there may be significant problem domain information. Efficient SAT algorithms may be developed by exploring input- and application-specific structures (Section 11.3). Specialized algorithms tailored to particular applications, on the other hand, do provide key insights to general satisfiability testing.

Practical Application Case Study. It has been recognized by SAT researchers that practical application problems are the driving forces for SAT research; they are the ultimate benchmarks to test SAT algorithms. This direction was further addressed by the NSF, the advisory committee, and the organizing committee of the 1996 DIMACS Satisfiability workshop [148, 296, 297]. There has been a strong relationship between theory, algorithms, and applications of SAT. A major step in the future is to bring together theorists, algorithmists, and practitioners working on SAT and on industrial applications involving SAT, enhancing the interaction between the three research groups. It would be beneficial to research community and to industry if we can apply theoretical and algorithmic results on SAT to practical problems, while taking these practical problems for further theoretical/algorithmic study. In addition to theoretical/algorithmic study, in the future, we will also further concentrate on significant industrial case studies of SAT, practical applications of SAT algorithms, and practical and industrial SAT benchmarks.

Parallel Algorithms and Architectures. Implementing an algorithm on VLSI hardware architectures is a common practice to speed up algorithm

execution. Not only does it offer faster execution speed, certain sequential portions of the algorithm may be implemented in hardware architectures in parallel form. For SAT *per se*, it has certain granularity at the search tree level, clause level, and variable level that lend itself well to parallel processing. A number of parallel algorithms and architectures for solving SAT have been developed and have been found to perform well at different levels of granularity. Two basic approaches have been taken in this direction: implementing parallel SAT inference algorithms on special-purpose VLSI chips [232, 233], and implementing tightly-coupled, parallel SAT algorithms on existing sequential computer machines [212, 217, 217, 503, 502].

Algorithm Engineering Approach. Aho, Johnson, Karp, Kosaraju, McGeoch, Papadimitriou, and Pevzner have recently proposed an algorithm engineering approach for the experimental testing of algorithms [8]. They believe that “Within theoretical computer science algorithms are usually studied within highly simplified models of computation and evaluated by metrics such as their asymptotic worst-case running time or their competitive ratio. These metrics can be indicative of how algorithms are likely to perform in practice, but they are not sufficiently accurate to predict actual performance. The situation can be improved by using models that take into account more details of system architecture and factors such as data movement and interprocessor communication, but even then considerable experimentation and fine-tuning is typically required to get the most out of a theoretical idea. Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice.”

16 Conclusions

The SAT problem is at the *core* of the class of NP-complete problems and has many practical applications. In recent years, many optimization methods, parallel algorithms, and practical techniques have been developed for solving the SAT problem. The past two decades have seen the proliferation of many SAT algorithms: resolution, local search, nonlinear optimization, BDD SAT solver, and multispace search, among others. Existing methods complement rather than exclude each other by being effective for particular instances of SAT. In this survey, we present a general algorithm space that integrates existing SAT algorithms into a unified perspective. We describe several major

classes of SAT algorithms with the emphasis on introducing recent advances in SAT algorithms. We gave performance evaluation of some existing SAT algorithms. This chapter also provides a set of practical applications of SAT. The area of SAT research is a rich land of well-developed theory and methods. To apply theoretical/algorithmic results to practical problems seems the ultimate way to test and benchmark SAT algorithms. Not only will the end results of such an endeavor have a major scientific/industrial impact, but in the process it will push optimization technology to its limit.

Acknowledgments

The authors are grateful to many people for their help in preparing this article. Moshe Vardi suggested that this survey be written for the 1996 DIMACS satisfiability workshop. Bob Johnson, Steve Cook, David Johnson, and Christos Papadimitriou have provided valuable comments to this article. We thank for Ranan Banerji, Roberto Battiti, Max Böhm, Endre Boros, Randy Bryant, H. Kleine Bünning, Guo Liang Chen, James Crawford, Rina Dechter, Xiaotie Deng, Bin Du, Ding-Zhu Du, Olivier Dubois, Giorgio Gallo, Wen Gao, Allen Van Gelder, Ian Gent, Fred Glover, Qianping Gu, Vladimir Gurvich, Peter Hammer, Pierre Hansen, Peter Heusch, Jieh Hsiang, Frank Hsu, Harry B. Hunt III, Kazuo Iwama, Philip C. Jackson, Lewis Johnson, Henry Kautz, Leonid Khachiyan, Dennis Kibler, Scott Kirkpatrick, Lefteris Kirousis, Oliver Kullmann, Vipin Kumar, R.C.T. Lee, Theo Lettmann, Guojie Li, Ming Li, Wei Li, Arne Lokketangen, Hans Maaren, M.V. Marathe, David Mitchell, John Mitchell, Henri Morel, Katta Murty, Panos Pardalos, David Plaisted, Vaughan Pratt, D. Pretolani, Marco Protasi, Ruchir Puri, Mauricio Resende, Daniel Rosenkrantz, Craig Rushforth, Andy Sage, Ingo Schiermeyer, John Schlipf, Bart Selman, Sandeep Shukla, Ewald Speckenmeyer, Cosimo Spera, R.E. Stearns, Alasdair Urquhart, Peter Vanbekbergen, Anthony Vannelli, Stephen Vavasis, Toby Wals, Jinchang Wang, Wei Wang, David Yau, for their insightful comments on various early versions of this chapter.

References

- [1] L. Abel. On the order of connections for automatic wire routing. *IEEE Trans. on Computers*, pages 1227–1233, Nov. 1972.
- [2] B. Abramson and M. Yung. Divide and conquer under global constraints: A solution to the n-queens problem. *Journal of Parallel and Distributed Computing*, 6:649–662, 1989.
- [3] Leonard M. Adleman Molecular Computation of Solutions to Combinatorial Problems *Science*, 266:1021–1024, 1994.
- [4] T. Agerwala. Microprogram Optimization : A Survey. *IEEE Trans. on Computers*, C-25:962–973, Oct. 1976.
- [5] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, 1974.
- [6] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, 1985.
- [7] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers*. Addison-Wesley, Reading, 1986.
- [8] A.V. Aho, D.S. Johnson, R.M. Karp, S.R. Kosaraju, C.C. McGeoch, C.H. Papadimitriou, and P. Pevzner, *Theory of computing: Goals and directions*, March 15, 1996.
- [9] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, Jun. 1978.
- [10] W. Ahrens. *Mathematische Unterhaltungen und Spiele (in German)*. B.G. Teubner (Publishing Company), Leipzig, 1918-1921.
- [11] S.B. Akers. On the use of the linear assignment algorithm in module placement. In *Proc. of the 18th ACM/IEEE Design Automation Conference*, pages 137–144, 1981.
- [12] J. Aloimonos. Visual shape computation. *Proceedings of the IEEE*, 76:899–916, Aug. 1988.
- [13] J. A. Anderson and G. E. Hinton. *Models of Information Processing in the Brain*. In G. E. Hinton and J. A. Anderson, editors, *Parallel Models of Associative Memory*, chapter 1, pages 9–48. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1981.
- [14] J.A. Anderson and E. Rosenfeld, editors. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, 1988.

- [15] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings 33rd IEEE Symposium on the Foundations of Computer Science*, pages 14–23, 1992.
- [16] K. J. Arrow and L. Hurwicz. Gradient method for concave programming, I: Local results. In K. J. Arrow, L. Hurwicz, and H. Uzawa, editors, *Studies in Linear and Nonlinear Programming*. Stanford University Press, Stanford, CA, 1958.
- [17] P. Ashar, A. Ghosh, and S. Devadas. Boolean satisfiability and equivalence checking using general Binary Decision Diagrams. *Integration, the VLSI journal*, 13:1–16, 1992.
- [18] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–132, Mar. 1979.
- [19] B. Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms* 1, pages 97–103, 1980.
- [20] A. Bagchi, B. Servatius, and W. Shi. Fault tolerance in massively parallel computers. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [21] H. Baird. Anatomy of a versatile page reader. *Proceedings of the IEEE*, 80(7):1059–1065, Jul. 1992.
- [22] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
- [23] P. Banerjee, M.H. Jones, and J.S. Sargent. Parallel simulated annealing algorithms for cell placement on hypercube multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 1(1):91–106, Jan. 1990.
- [24] A.E. Barbour. Solutions to the minimization problem of fault-tolerant logic circuits. *IEEE Trans. on Computers*, 41(4):429–443, Apr. 1992.
- [25] P. Beame and T. Pitassi. Simplified and improved resolution lower bounds. In *Proceedings of the 38th Annual Symposium of the Foundations of Computer Science*, to appear, 1997.
- [26] S. Ben-Davis, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *J. of Computer and Systems Sciences*, 44:193–219, 1992.

- [27] A. Ben-Tal, G. Eiger, and V. Gershovitz. Global minimization by reducing the duality gap. *Mathematical Programming*, 63:193–212, 1994.
- [28] R. G. Bennetts. An Improved Method for Prime C-Class Derivation in the State Reduction of Sequential Networks. *IEEE Trans. on Computers*, C-20:229–231, Feb. 1971.
- [29] A. Beringer, G. Aschemann, H.H. Hoos, M. Metzger, and A. Wei. GSAT versus simulated annealing. In *Proceedings of ECAI'94*, pages 130–134, 1994.
- [30] K. Berman, J. Franco, and J. Schlipf. Unique satisfiability for Horn sets can be solved in nearly linear time. *Proc. Seventh Advanced Research Institute in Discrete Applied Mathematics* (ARIDAM), New Brunswick, New Jersey, May, 1992. In *Discrete Applied Mathematics* 60:77–91, 1995.
- [31] B. Bernhardsson. Explicit solutions to the n-queens problems for all n. *ACM SIGART Bulletin*, 2(2):7, Apr. 1991, ACM Press.
- [32] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publication, 1987.
- [33] Wolfgang Bibel. Automated theorem proving. *Vieweg*, 1982.
- [34] N. N. Biswas. State Minimization of Incompletely Specified Sequential Machines. *IEEE Trans. on Computers*, C-23:80–84, Jan. 1974.
- [35] J. R. Bitner and E. M. Reingold. Backtrack programming techniques. *Comm. of ACM*, 18(11):651–656, Nov. 1975.
- [36] C.E. Blair, R.G. Jeroslow, and J.K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 5:633–645, 1986.
- [37] J.P. Blanks. Near-optimal placement using a quadratic objective function. In *Proc. of the 22th ACM/IEEE Design Automation Conference*, pages 609–615, 1985.
- [38] J. Blazewicz. Scheduling dependent tasks with different arrival times to meet deadlines. In *Proc. of the International Workshop on Modeling and Performance Evaluation of Computer Systems*, pages 57–65, 1976.
- [39] M. Böhm and E. Speckenmeyer. A fast parallel SAT-solver — efficient workload balancing. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994. To appear in Annals of Mathematics and Artificial Intelligence.

- [40] S. H. Bokhari. Partitioning problems in parallel, pipelined, and distributed computing. *IEEE Trans. on Computers*, 37(1):48–57, Jan 1988.
- [41] F. Bonomi. On job assignment for a parallel system of processor sharing queues. *IEEE Trans. on Computers*, 39(7):858–869, July 1990.
- [42] E. Boros, P.L. Hammer, and A. Kogan. Computational experiments with an exact SAT solver. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [43] E. Boros, Y. Crama, P. L. Hammer. Polynomial-time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence* 1, pages 21–32, 1990.
- [44] E. Boros, P. L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics* 55, pages 1–13, 1994.
- [45] E. Boros, Y. Crama, P. L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journal on Computing* 23, pages 45–49, 1994.
- [46] H.N. Brady. An approach to topological pin assignment. *IEEE Trans. on CAD*, Vol. 3, pp. 250–255, July 1984.
- [47] M. Brady, J.M. Hollerbach, T.L. Johnson, T. Lozano-Perez, and M.T. Mason, editors. *Robot Motion: Planning and Control*. The MIT Press, Cambridge, 1982.
- [48] M.A. Breuer. Min-cut placement. *J. of Design Automation and Fault Tolerant Computing*, 1:343–362, 1976.
- [49] F. Brewer and D. Gajski. Chippe: A system for constraint driven behavioral synthesis. *IEEE Trans. on CAD*, 9(7):681–695, July 1990.
- [50] A.Z. Broder, A.M. Frieze, and E. Upfal. On the satisfiability and maximum satisfiability of random 3-CNF formulas. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 322–330, 1993.
- [51] M.J. Brooks and B.K.P. Horn. Shape and source from shading. In *Proc. of IJCAI'85*, pages 932–936, Aug. 1985.
- [52] C.A. Brown, L.A. Finklestein, and P.W. Purdom. Backtrack Searching in the Presence of Symmetry. *6th International Conference on Algebraic Algorithms and Error Correcting Codes (AAECC)*, Lecture Notes in Computer Science, Vol. 357, pp. 99-110.

- [53] C.A. Brown, L.A. Finklestein, and P.W. Purdom. Backtrack Searching in the Presence of Symmetry. *Nordic Journal of Computing*, to appear.
- [54] C.A. Brown and P.W. Purdom. An average time analysis of backtracking. *SIAM J. on Computing*, 10(3):583–593, Aug. 1981.
- [55] C.A. Brown and P.W. Purdom. An empirical comparison of backtracking algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-4(3):309–316, May 1982.
- [56] B. Bruderlin. Constructing three dimensional geometric objects defined by constraints. In *Proceedings of 1986 ACM SIGGRAPH Workshop in Interactive Graphics*, 1986.
- [57] M. Bruynooghe. Solving combinatorial search problems by intelligent backtracking. *Information Processing Letters*, 12(1):36–39, 1981.
- [58] M. Bruynooghe. Graph coloring and constraint satisfaction. Technical Report CW 44, Dept. of Computer Science, Katholieke Universiteit Leuven, Dec. 1985.
- [59] M. Bruynooghe and L.M. Pereira. *Deduction Revision by Intelligent Backtracking*, pages 194–215. Ellis Horwood Limited, 1984.
- [60] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, Aug. 1986.
- [61] R.E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, Sept. 1992.
- [62] R.E. Bryant. Binary Decision Diagrams Applied to SAT and Related Problems. Presented at 1996 DIMACS Workshop on Satisfiability Problem: Theory and Applications. March 11, 1996.
- [63] H. Kleine Büning. On generalized Horn formulas and k resolution. *Theoretical Computer Science* 116, pages 405–413, 1993.
- [64] H. Kleine Büning and T. Lettmann. Aussagenlogik: Deduktion und Algorithmen. B.G. Teubner, Stuttgart, 1993. English version to appear in 1996.
- [65] K. Bugrara and P. Purdom. Clause order backtracking. Technical Report 311, 1990.
- [66] K.M. Bugrara and C.A. Brown. On the average case analysis of some satisfiability model problems. *Information Sciences*, 40:21–37, 1986.

- [67] K.M. Bugrara, Y.F. Pan, and P. Purdom. Exponential average time for the pure literal rule. *SIAM J. on Computing*, 18:409–418, 1989.
- [68] A. Bundy, editor. *Catalogue of Artificial Intelligence Tools*. Springer-Verlag, Berlin, 1984.
- [69] J.L. Burns and A.R. Newton. Efficient constraint generation for hierarchical compaction. In *Proc. Int'l Conf. on Computer Design*, pages 197–200. IEEE Computer Society, Oct. 1987.
- [70] M. Buro and H.K. Büning. Report on a SAT competition. Technical report, FB-17 — Mathematik/Informatik, Universität Paderborn, Nov. 1992.
- [71] M. Buro and H.K. Büning. Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, Feb. 1993.
- [72] M. Burstein and R. Pelavin. Hierarchical channel router. In *Proc. of the 20th ACM/IEEE Design Automation Conference*, pages 591–597, Jun. 1983.
- [73] G. Butler. Computing in Permutation and Matrix Groups II: Backtrack Algorithm. *Math. Comp.* 39: 671–680, 1982.
- [74] G. Butler and C.W.H. Lam. A General Backtracking Algorithm for the Isomorphism Problem of Combinatorial Objects. *J. Symbolic Computation* 1:363–381, 1985.
- [75] V. Černy. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
- [76] S. Chakradhar, V. Agrawal, and M. Bushnell. Neural net and Boolean satisfiability model of logic circuits. *IEEE Design & Test of Computers*, pages 54–57, Oct. 1990.
- [77] I. Chakravarty. A generalized line and junction labeling scheme with applications to scene analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-1(2):202–205, Apr. 1979.
- [78] P.K. Chan, M.D.F. Schlag, C.D. Thomborson, and V.G. Oklobdzija. Delay optimization of carry-skip adders and block carry-lookahead adders using multidimensional dynamic programming. *IEEE Trans. on Computers*, 41(8):920–930, Aug. 1992.

- [79] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.
- [80] V. Chandru and J. N. Hooker. Extended Horn sets in propositional logic. *J. ACM* **38**, pages 205–221, 1991.
- [81] Y. Chang and B. W. Wah. Lagrangian techniques for solving a class of zero-one integer linear programs. In *Proc. Computer Software and Applications Conf.*, 1995.
- [82] Y.-J. Chang and B. W. Wah. Lagrangian techniques for solving a class of zero-one integer linear programs. In *Proc. Computer Software and Applications Conference*, pages 156–161, Dallas, TX, August 1995. IEEE.
- [83] M.T. Chao and J. Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. on Computing*, 15:1106–1118, 1986.
- [84] M. T. Chao and J. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k satisfiable problem. *Information Sciences*, 51:289–314, 1990.
- [85] R. Chandrasekaran. Integer programming problems for which a simple rounding type of algorithm works. In W. Pulleyblank, ed. *Progress in Combinatorial Optimization*. Academic Press Canada, Toronto, Ontario, Canada, pages 101–106, 1984.
- [86] H.R. Charney and D.L. Plato. Efficient partitioning of components. In *Proc. of the 5th Annual Design Automation Workshop*, pages 16–21, 1968.
- [87] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *J. of ACM*, 35:759–770, 1988.
- [88] W.T. Chen and L.L. Liu. Parallel approach for theorem proving in propositional logic. *Inform. Sci.*, 41(1):61–76, 1987.
- [89] R. T. Chin and C. R. Dyer. Model-based recognition in robot vision. *ACM Computing Surveys*, 18(1):67–108, Mar. 1986.
- [90] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, June 1987.

- [91] Y. Chu, editor. *Special Section on Chinese/Kanji Text and Data Processing*. *IEEE Computer*, volume 18, number 1. IEEE Computer Society Press, 1985.
- [92] V. Chvátal and B. Reed. Mick gets some (the odds are on his side). In *Proceedings on the Foundations of Computer Science*, 1992.
- [93] A. Cichocki and R. Unbehauen. Switched-capacitor artificial neural networks for nonlinear optimization with constraints. In *Proc. of 1990 IEEE Int'l Symposium on Circuits and Systems*, pages 2809–2812, 1990.
- [94] J. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2), 1987.
- [95] W. F. Clocksin and C. S. Mellish. *Programming in Prolog (2nd Edition)*. Springer-Verlag, Berlin, 1984.
- [96] M.B. Clowes. On seeing things. *Artificial Intelligence*, 2:79–116, 1971.
- [97] J. Cohen, editor. *Special Section on Logic Programming*. *Comm. of the ACM*, volume 35, number 3. 1992.
- [98] J.P. Cohoon and W.D. Paris. Genetic placement. In *Digest of the Int'l Conf. on Computer-Aided Design*, pages 422–425, 1986.
- [99] A. Colmerauer. Opening the Prolog III universe. *BYTE Magazine*, pages 177–182, Aug. 1987.
- [100] J.S. Conery. *Parallel Execution of Logic Programs*. Kluwer Academic Publishers, Boston, 1987.
- [101] M. Conforti, G. Cornuéjols, A. Kapoor, K. Vušković, and M. R. Rao. Balanced Matrices. Mathematical Programming: State of the Art. J. R. Birge and K. G. Murty, eds. Braun-Brumfield, United States. Produced in association with the 15th Int'l Symposium on Mathematical Programming, University of Michigan, 1994.
- [102] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third ACM Symposium on Theory of Computing*, pages 151–158, 1971.
- [103] S.A. Cook. Find hard instances of the satisfiability problem. Presented at the 1996 DIMACS Workshop on Satisfiability Problem: Theory and Applications. March 11, 1996.
- [104] P. R. Cooper and M. J. Swain. Parallelism and domain dependence in constraint satisfaction. Technical Report TR 255, Dept. of Computer Science, Univ. of Rochester, Dec. 1988.

- [105] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, 1990.
- [106] D.G. Corneil and D.G. Kirkpatrick. A theoretical analysis of various heuristics for the graph isomorphism problem. *SIAM J. on Computing*, 9(2):281–297, 1980.
- [107] Y. Crama, P. L. Hammer, B. Jaumard, and B. Simeone. Product form parametric representation of the solutions to a quadratic boolean equation. *R.A.I.R.O. Recherche opérationnelle/Operations Research* 21:287–306, 1987.
- [108] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics* 29:171–185, 1990.
- [109] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problem. In *Proc. of AAAI'93*, pages 21–27, Aug. 1993.
- [110] J. Crawford and L. Auton. Experimental results on the crossover point in satisfiability problems. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [111] J.M. Crawford. Solving satisfiability problems using a combination of systematic and local search. Submitted to the DIMACS Challenge II Workshop, 1994.
- [112] Z. Cvetanovic. The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. *IEEE Trans. on Computers*, C-36(4):421–432, Apr 1987.
- [113] W. Dai and E.S. Kuh. Hierarchical floorplanning for building block layout. In *Digest of Int'l Conf. on Computer-Aided Design*, pages 454–457, 1986.
- [114] M. Dalal, and D. W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing letters* 44, pages 173–180, 1992.
- [115] S. R. Das, D. K. Banerji, and A. Chattopadhyay. On Control Memory Minimization in Microprogrammed Digital Computers. *IEEE Trans. on Computers*, C-22:845–848, Sept. 1973.
- [116] L. S. Davis. Shape matching using relaxation techniques. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-1(1):60–72, Jan. 1979.

- [117] L. S. Davis and T. C. Henderson. Hierarchical constraint processes for shape analysis. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-3(3):265–277, May 1981.
- [118] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
- [119] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. of ACM*, 7:201–215, 1960.
- [120] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.
- [121] J. de Kleer. Problems with ATMS. *Artificial Intelligence*, 28:197–224, 1986.
- [122] J. de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of 11th IJCAI*, pages 290–296, 1989.
- [123] J. de Kleer. Exploiting locality in a TMS. In *Proceedings of AAAI'90*, pages 264–271, 1990.
- [124] R. Dechter. Learning while searching in constraint satisfaction problems. In *Proceedings of AAAI'86*, 1986.
- [125] R. Dechter. A constraint-network approach to truth maintenance. Technical Report R-870009, Computer Science Dept., UCLA, Los Angeles, 1987.
- [126] R. Dechter. Enhancement schemes for constraint processing: Back-jumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3), 1990.
- [127] R. Dechter. Directional resolution: The Davis-Putnam procedure revisited. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [128] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of AAAI'88*, 1988.
- [129] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.
- [130] J.S. Denker, editor. *Neural Networks for Computing*, volume 151 of *AIP (Snowbird, Utah) Conference Proceedings*. American Institute of Physics, New York, 1986.

- [131] N. Dershowitz, J. Hsiang, N. Josephson, and D. Plaisted. Associative-commutative rewriting. In *Proceedings of IJCAI*, pages 940–944, 1983.
- [132] D.N. Deutsch. A 'dogleg' channel router. In *Proc. of the 13th ACM/IEEE Design Automation Conference*, pages 425–433, Jun. 1976.
- [133] D.N. Deutsch. Compacted channel routing. In *Digest Int'l Conf. on Computer-Aided Design*, pages 223–225, Nov. 1985.
- [134] J. P. A. Deutsch. A short cut for certain combinational problems. In *British Joint Comput. Conference*, 1966.
- [135] S. Devadas. Optimal layout via Boolean satisfiability. In *Proceedings of ICCAD'89*, pages 294–297, Nov. 1989.
- [136] S. Devadas, K. Keutzer, and J. White. Estimation of power dissipation in CMOS combinational circuits using Boolean function manipulation. *IEEE Transactions on CAD*, 11(3):373–383, Mar. 1992.
- [137] S. Devadas, H.T. Ma, A.R. Newton, and A.S. Vincentelli. A synthesis and optimization procedure for fully and easily testable sequential machines. *IEEE Trans. on CAD*, 8(10):1100–1107, Oct. 1989.
- [138] S.K. Dhall and C.L. Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, Feb. 1978.
- [139] V. Dhar and A. Crocker. A problem-solver/TMS architecture for general constraint satisfaction problems. Technical report, Dept. of Information Systems, New York University, 1989.
- [140] M. Dincbas, H. Simonis, and P.V. Hentenryck. Solving a cutting-stock problem in constraint logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, 1988.
- [141] J. Doenhardt and T. Lengauer. Algorithm aspects of one-dimensional layout. *IEEE Trans. on CAD*, CAD-6(5):863–878, 1987.
- [142] W.E. Donath. Placement and average interconnection lengths of computer logic. *IEEE Trans. on Circuits and Systems*, CAS-26(4):272–277, Apr. 1979.
- [143] W.E. Donath. Wire length distribution for placements of computer logic. *IBM J. of Research and Development*, 25(3):152–155, May 1981.
- [144] W.E. Donath and A.J. Hoffman. Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin 15*, pages 938–944, 1972.

- [145] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming* 1, pages 267–284, 1984.
- [146] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.
- [147] B. Du and J. Gu. Sequential circuit test generation by Boolean satisfiability. 1996, to appear.
- [148] D.Z. Du, J. Gu, and P.M. Pardalos, editors. *The Satisfiability (SAT) Problem*. DIMACS Volume Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.
- [149] O. Dubois. Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science*, 81:49–64, 1991.
- [150] O. Dubois and J. Carlier. Probabilistic approach to the satisfiability problem. *Theoretical Computer Science*, 81:65–75, 1991.
- [151] O. Dubois and Y. Boufkhad. Analysis of the space of solutions for random instances of the satisfiability problem. *Proceedings of the fourth International Symposium on Artificial Intelligence and Mathematics*, Ft. Lauderdale, Florida, p. 175, 1996.
- [152] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. *DIMACS Series Volume: Clique, Graph Coloring, and Satisfiability — Second DIMACS Implementation Challenge*. Editors: D.S. Johnson and M.A. Trick, 26, American Mathematical Society, 1996.
- [153] C. Eastman. Preliminary report on a system for general space planning. *Comm. of ACM*, 15(2):76–87, Feb. 1972.
- [154] T. Eiter, P. Kilpelainen, and H. Mannila. Some remarks on renaming and satisfiability hierarchies. Summary Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [155] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger. The notion of consistency and predicate look in a database system. *Comm. of ACM*, 19(11), Nov. 1976.
- [156] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM J. on Computing*, 5(4):691–703, 1976.
- [157] B.-J. Falkowski and L. Schmitz. A note on the queens' problem. *Information Processing Letters*, Vol. 23:39–46, July 1986.

- [158] M.Y. Fang and W.T. Chen. Vectorization of a generalized procedure for theorem proving in propositional logic on vector computers. *IEEE Trans. on Knowledge and Data Engineering*, 4(5):475–486, Oct. 1992.
- [159] O. Faugeras and K. Price. Semantic description of aerial images using stochastic labeling. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-3(6):633–642, Nov. 1981.
- [160] O. D. Faugeras, editor. *Fundamentals in Computer Vision*. Cambridge University Press, London, 1983.
- [161] J. A. Feldman and Y. Yakimovsky. Decision theory and artificial intelligence: I. a semantics-based region analyzer. *Artificial Intelligence*, 5:349–371, 1974.
- [162] J. D. Findler. *Associative Networks: Representation and Use of Knowledge by Computers*. Academic Press, New York, 1979.
- [163] C.A. Floudas and P.M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer Verlag, New York, 1990.
- [164] C.A. Floudas and P.M. Pardalos, editors. *Recent Advances in Global Optimization*. Princeton University Press, New York, 1992.
- [165] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Pitman, London, 1987.
- [166] J. Franco. On the probabilistic performance of algorithms for the satisfiability problem. *Information Processing Letters*, 23:103–106, 1986.
- [167] J. Franco. Elimination of infrequent variables improves average case performance of satisfiability algorithms. *SIAM J. on Computing*, 20:1119–1127, 1991.
- [168] J. Franco and Y.C. Ho. Probabilistic performance of heuristic for the satisfiability problem. *Discrete Applied Mathematics*, 22:35–51, 1988/89.
- [169] J. Franco and M. Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.
- [170] J. Franco. 1993. On the occurrence of null clauses in random instances of satisfiability. *Discrete Applied Mathematics* 41, pp. 203–209.
- [171] J. Franco and R. Swaminathan. Toward a good algorithm for determining unsatisfiability of propositional formulas. 1996.

- [172] J. Franco, and R. Swaminathan. 1996. Average case results for Satisfiability algorithms under the random-clause-width model. To appear in *Annals of Mathematics and Artificial Intelligence*.
- [173] J. Franco, J. Goldsmith, J. Schlipf, E. Speckenmeyer, R. P. Swaminathan. 1996. An algorithm for the class of pure implicational formulas. Proceedings of the *Workshop on Satisfiability*, Siena, Italy.
- [174] J. Franco. Relative Size of Certain Polynomial Time Solvable Subclasses of Satisfiability. DIMACS Volume, Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1997.
- [175] R.T. Frankot and R. Chellappa. A method for enforcing integrability in shape from shading algorithms. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-10(4):439–451, Jul. 1988.
- [176] F. Frayman and S. Mittal. *Cossack: A Constraints-based Expert System for Configuration Tasks*. Computational Mechanics Publications, Nadel, 1987.
- [177] J.W. Freeman. Improvements to the davis-putnam procedure for satisfiability. Jan. 1994.
- [178] E. C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, Jan. 1982.
- [179] E. C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, Oct. 1985.
- [180] E.C. Freuder and M.J. Quinn. Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of 9th IJCAI*, pages 1076–1078, 1985.
- [181] A. M. Frieze, and S. Suen. 1993. Analysis of simple heuristics for random instances of 3-SAT.
- [182] X. Fu. The complexity of resolution proofs.
- [183] T. W. Fuqua. Constraint kernels: Constraints and dependencies in a geometric modeling system. Master's thesis, Dept. of Computer Science, Univ. of Utah, Aug. 1987.
- [184] Z. Galil. On the complexity of regular resolution and the Davis-Putnam procedure. *Theoretical Computer Science*, pages 23–46, 1977.
- [185] G. Gallo, and M. G. Scutella. Polynomially solvable satisfiability problems. *Information Processing Letters* 29, pages 221–227, 1988.

- [186] G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *J. of Logic Programming*, 7:45–61, 1989.
- [187] G. Gallo and D. Pretolani. Hierarchies of polynomially solvable SAT problems. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [188] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Trans. on Database Systems*, 8(2), Jun. 1983.
- [189] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4:397–411, 1975.
- [190] M.R. Garey and D.S. Johnson. Two-processors scheduling with start-times and deadlines. *SIAM J. on Computing*, 6:416–426, 1977.
- [191] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.
- [192] J. Gaschnig. A constraint satisfaction method for inference making. In *Proceedings of 12th Annual Allerton Conf. Circuit System Theory*, 1974.
- [193] J. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of 9th IJCAI*, page 457, 1977.
- [194] J. Gaschnig. *Performance Measurements and Analysis of Certain Search Algorithms*. PhD thesis, Carnegie-Mellon University, Dept. of Computer Science, May 1979.
- [195] A.V. Gelder. A satisfiability tester for non-clausal propositional calculus. *Information and Computation*, 79(1):1–21, Oct. 1988.
- [196] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, Nov. 1984.
- [197] S. Geman and C.-R. Hwang. *Diffusions of Global optimization*. Brown University, 1984.
- [198] M.R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Los Altos, California, 1987.

- [199] I.P. Gent and T. Walsh. The hardest random SAT problems. In *Proceedings of KI'94*, 1994.
- [200] P.C. Gilmore. A proof method for quantification theory. *IBM J. Res. Develop.*, 4:28–35, 1960.
- [201] A. Ginzberg. *Algebraic Theory of Automata*. Academic Press, New York, 1968.
- [202] F. Glover. Tabu search — Part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.
- [203] A. Goerdt. 1992. A threshold for unsatisfiability. *Proceedings of 17th annual Symposium on Mathematical Foundations of Computer Science*, Prague, Czechoslovakia.
- [204] A. Goldberg. Average case complexity of the satisfiability problem. In *Proc. Fourth Workshop on Automated Deduction*, pages 1–6, 1979.
- [205] A. Goldberg. On the complexity of the satisfiability problem. Technical Report Courant Computer Science No. 16, New York University, 1979.
- [206] A. Goldberg, P.W. Purdom, and C.A. Brown. Average time analysis of simplified Davis-Putnam procedures. *Information Processing Letters*, 15(2):72–75, 6 Sept. 1982 (Some printer errors in this paper were corrected in *Information Processing Letters*, 16:213, 1983).
- [207] A. Grasselli and F. Luccio. A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks. *IEEE Trans. on Computers*, EC-14:350–359, June 1965.
- [208] A. Grasselli and U. Montanari. On the Minimization of READ-ONLY Memories in Microprogrammed Digital Computers. *IEEE Trans. on Computers*, C-19:1111–1114, Nov. 1970.
- [209] J. Gu, P.W. Purdom, J. Franco, and B.W. Wah. Algorithms for satisfiability (SAT) problem: A survey. *DIMACS Volume Series on Discrete Mathematics and Theoretical Computer Science*, Vol. 35: *The Satisfiability (SAT) Problem*, pages 19–151, American Mathematical Society, 1997.
- [210] J. Gu, P.W. Purdom, J. Franco, and B. Wah. Algorithms for the Satisfiability Problem. Cambridge University Press, 1999.
- [211] J. Gu. Parallel algorithms and architectures for very fast search. Ph.D thesis. Technical Report UUCS-TR-88-005, July 1988.

- [212] J. Gu. How to solve Very Large-Scale Satisfiability problems. Technical Report UUCS-TR-88-032, 1988, and Technical Report UCECE-TR-90-002, 1990.
- [213] J. Gu. An $\alpha\beta$ -relaxation for global optimization. Technical Report UCECE-TR-91-003, Apr. 1991.
- [214] J. Gu. Efficient local search for very large-scale satisfiability problem. *SIGART Bulletin*, 3(1):8–12, Jan. 1992, ACM Press.
- [215] J. Gu. *On Optimizing a Search Problem*. In N. G. Bourbakis, editor, *Artificial Intelligence Methods and Applications*, Vol. 1, chapter 2, pages 63–105. World Scientific Publishers, New Jersey, Jan. 1992.
- [216] J. Gu. The *UniSAT* problem models (appendix). *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):865, Aug. 1992.
- [217] J. Gu. Local search for satisfiability (SAT) problem. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(4):1108–1129, Jul. 1993, and 24(4):709, Apr. 1994.
- [218] J. Gu. Global optimization for satisfiability (SAT) problem. *IEEE Trans. on Knowledge and Data Engineering*, 6(3):361–381, Jun. 1994, and 7(1):192, Feb. 1995.
- [219] J. Gu. *Optimization Algorithms for the Satisfiability (SAT) Problem*. In *Advances in Optimization and Approximation*. pages 72–154. Kluwer Academic Publishers, 1994.
- [220] J. Gu. Parallel algorithms for satisfiability (SAT) problem. *DIMACS Volume Series on Discrete Mathematics and Theoretical Computer Science*, Vol. 22, pages 105–161, American Mathematical Society, 1995.
- [221] J. Gu and R. Puri. Asynchronous circuit synthesis by Boolean satisfiability. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 14(8):961–973, Aug. 1995.
- [222] J. Gu. Satisfiability Problems in VLSI Engineering. Presented at *Annual Conference of the Institute for Operations Research and Management Science*, Oct. 1995, and *DIMACS Workshop on Satisfiability Problem*, Mar. 1996. To appear in *Discrete Applied Mathematics*.
- [223] J. Gu, Q.P. Gu, and D.-Z. Du. Convergence properties of optimization algorithms for the satisfiability (SAT) problem. *IEEE Trans. on Computers*, 45(2): 209–219, Feb. 1996.

- [224] J. Gu and Lizhoudu. An efficient implementation of the SAT1.5 algorithm. Technical Report, USTC, Sept. 1995.
- [225] J. Gu. The *Multi-SAT* algorithm. *Siena SAT Workshop and Workshop on Semidefinite and Interior-Point Methods*, May 1996. To appear in *Discrete Applied Mathematics*.
- [226] J. Gu and Q.P. Gu. Average time complexities of several local search algorithms for the satisfiability (SAT) problem. Technical Report UCECE-TR-91-004, 1991. In *Lecture Notes in Computer Science*, Vol. 834, pp. 146-154, 1994 and to appear in *IEEE Trans. on Knowledge and Data Engineering*.
- [227] J. Gu. Optimization by multispace search. Technical Report UCECE-TR-90-001, Jan. 1990.
- [228] J. Gu. *Multispace Search: A New Optimization Approach (Summary)*. In *Lecture Notes in Computer Science*, Vol. 834, pages 252–260. 1994.
- [229] J. Gu. Multispace search for satisfiability and NP-hard problems. *DIMACS Volume Series on Discrete Mathematics and Theoretical Computer Science*, Vol. 35, pages 407–517, American Mathematical Society, 1997.
- [230] J. Gu. Randomized and Deterministic Local Search for SAT and Scheduling Problems. *DIMACS Volume Series on Discrete Mathematics and Theoretical Computer Science*, Vol. 43, pages 61–108, American Mathematical Society, 1998.
- [231] J. Gu and X. Huang. Local search with search space smoothing: A case study of the traveling salesman problem (TSP). Technical Report UCECE-TR-91-006, Aug. 1991. In *IEEE Trans. on Systems, Man, and Cybernetics*, 24(5):728–735, May 1994.
- [232] J. Gu, W. Wang, and T.C. Henderson. A parallel architecture for discrete relaxation algorithm. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-9(6):816–831, Nov. 1987.
- [233] J. Gu and W. Wang. A novel discrete relaxation architecture. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):857–865. Aug. 1992.
- [234] J. Gu. *Constraint-Based Search*. Cambridge University Press, New York, to appear.
- [235] J. Gu. *Optimization by multispace search*. Kluwer Academic Publishers, to appear.

- [236] Y. Gurevich. Average case completeness. *J. of Computer and Systems Sciences*, 42(3):346–398, 1991.
- [237] A. Guzman. *Computer Recognition of Three-Dimensional Objects in a Visual Scene*. PhD thesis, MIT, 1968.
- [238] S. Ha and E.A. Lee. Compile-time scheduling and assignment of data-flow program graphs with data-dependent iteration. *IEEE Trans. on Computers*, 40(11):1225–1238, Nov. 1991.
- [239] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
- [240] G.T. Hamachi and J.K. Ousterhout. A switchbox router with obstacle avoidance. In *Proc. of the 21st ACM/IEEE Design Automation Conference*, pages 173–179, Jun. 1984.
- [241] M. Hanan, P.K. Wolff, and B.J. Agule. Some experimental results on placement techniques. *J. of Design Automation and Fault-Tolerant Computing*, 2:145–168, May 1978.
- [242] P. L. Hammer and S. Rudeanu. Boolean Methods in Operations Research and Related Areas. Springer-Verlag, New York, 1968.
- [243] P. Hansen and B. Jaumard. Uniquely solvable quadratic Boolean equations. *Discrete Applied Mathematics* 12:147–154, 1985.
- [244] P. Hansen, B. Jaumard, and V. Mathon. Constrained Nonlinear 0-1 Programming. *ORSA Journal on Computing* 5:97–119, 1993.
- [245] P. Hansen, B. Jaumard, and G. Plateau. An extension of nested satisfiability. Les Cahiers du GERAD, G-93-27, 1993.
- [246] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.
- [247] E. R. Hansen. *Global optimization using interval analysis*. M. Dekker, New York, 1992.
- [248] R. M. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
- [249] R. M. Haralick and L. G. Shapiro. The consistent labeling problem: Part 1. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-1(2):173–184, Apr. 1979.
- [250] F. Harary. *Graph Theory*. Addison-Wesley, Reading, 1969.

- [251] W. S. Havens. A theory of schema labeling. *Computational Intelligence*, 1(3 & 4):127–139, 1985.
- [252] T. Hedges, W. Dawson, and Y.E. Cho. Bitmap graph build algorithm for compaction. In *Digest Int'l Conf. on Computer-Aided Design*, pages 340–342, Sep. 1985.
- [253] T. C. Henderson and L. S. Davis. Hierarchical models and analysis of shape. *Pattern Recognition*, 14(1-6):197–204, 1981.
- [254] T. C. Henderson and A. Samal. Multi-constraint shape analysis. *Image and Vision Computing*, 4(2):84–96, May 1986.
- [255] P. V. Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, Cambridge, 1989.
- [256] P. Heusch. The Complexity of the Falsifiability Problem for Pure Implicational Formulas. *Proc. 20th Int'l Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, J. Wiedermann, P. Hajek (Eds.), Prague, Czech Republic. *Lecture Notes in Computer Science (LNCS 969)*, Springer-Verlag, Berlin, pages 221–226, 1995.
- [257] G.E. Hinton and J.A. Anderson, editors. *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1981.
- [258] G.E. Hinton and T.J. Sejnowski. *Learning and Relearning in Boltzmann Machine*. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, volume 1, chapter 7, pages 282–317. The MIT Press, Cambridge, 1986.
- [259] E. J. Hoffman, J. C. Loessi, and R. C. Moore. Constructions for the solution of the m queens problem. *Mathematics Magazine*, pages 66–72, 1969.
- [260] J. H. Holland. *Adaption in Natural and Adaptive Systems*. University of Michigan Press, Ann Arbor, 1975.
- [261] S.J. Hong and S. Muroga. Absolute minimization of completely specified switching functions. *IEEE Trans. on Computers*, 40(1):53–65, Jan. 1991.
- [262] P. Hood and Grover. Designing real time systems in Ada. Technical Report 1123-1, SofTech, Inc., Waltham, Jan. 1986.

- [263] J. Hooker and V. Vinay. An empirical study of branching rules for satisfiability. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [264] J.N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.
- [265] J.N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.
- [266] J.N. Hooker. Resolution vs. cutting plane solution of inference problems: Some computational experience. *Operations Research Letter*, 7(1):1–7, 1988.
- [267] J.N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. Technical Report 77-88-89, GSIA, Carnegie Mellon University, Aug. 1989.
- [268] A.L. Hopkins and et. al. FTMP - a highly reliable fault-tolerant multiprocessor for aircraft. In *Proceedings of the IEEE*, pages 1221–1239, Oct. 1978.
- [269] B.K.P. Horn. *Obtaining Shape from Shading Information*, in The Psychology of Computer Vision, P.H. Winston, editor, pages 115–155. McGraw-Hill, New York, 1975.
- [270] B.K.P. Horn. Understanding image intensity. *Artificial Intelligence*, 8:301–231, 1977.
- [271] B.K.P. Horn and M. J. Brooks, editors. *Shape from Shading*. The MIT Press, Cambridge, 1989.
- [272] B.K.P. Horn and M.J. Brooks. The variational approach to shape from shading. *Computer Vision, Graphics & Image Processing*, 33(2):174–208, 1986.
- [273] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, 1978.
- [274] R. Horst and H. Tuy. *Global Optimization: Deterministic Approaches*. Springer Verlag, Berlin, 1990.
- [275] J. Hsiang. Refutational theorem proving using term-rewriting systems. *Artificial Intelligence*, pages 255–300, 1985.
- [276] T.H. Hu, C.Y. Tang, and R.C.T. Lee. An average case analysis of a resolution principle algorithm in mechanical theorem proving. *Annals of Mathematics and Artificial Intelligence*, 6:235–252, 1992.

- [277] X. Huang, J. Gu, and Y. Wu. A constrained approach to multi font character recognition. *IEEE Transactions on Pattern Analysis And Machine Intelligence*, 15(8):838–843, Aug. 1993.
- [278] D.A. Huffman. *Impossible Objects as Nonsense Sentences*. In B. Meltzer and D. Michie, Eds., *Machine Intelligence*, pages 295–323. Edinburgh University Press, Edinburgh, Scotland, 1971.
- [279] R. A. Hummel and S. W. Zucker. On the foundations of relaxation labeling processes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-5(3):267–287, May 1983.
- [280] K. Ikeuchi. Model-based interpretation of range imagery. In *Proc. of Image Understanding Workshop*, pages 321–339. DARPA, Feb. 1987.
- [281] K. Ikeuchi and B.K.P. Horn. Numerical shape from shading and occluding boundaries. *Artificial Intelligence*, 17(1-3):141–184, 1981.
- [282] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random number generation from one way function. In *Proceedings of the Third ACM Symposium on Theory of Computing*, pages 12–14, 1989.
- [283] B. Indurkhya, H. S. Stone, and L. Xi-Cheng. Optimal partitioning of randomly generated distributed programs. *IEEE Trans. on Software Engineering*, SE-12:483–495, Mar 1986.
- [284] L. Ingber. *Adaptive Simulated Annealing (ASA)*. Lester Ingber Research, 1995.
- [285] T. Ishida. Parallel rule firing in production systems. *IEEE Trans. on Knowledge and Data Engineering*, 3(1):11–17, Mar. 1991.
- [286] A. Itai and J. Makowsky. On the complexity of Herbrand’s theorem. Working paper 243, Department of Computer Science, Israel Institute of Technology, 1982.
- [287] N. Itazaki and K. Kinoshita. Test pattern generation for circuits with tri-state modules by z-algorithm. *IEEE Trans. on CAD*, 8(12):1327–1334, Dec. 1989.
- [288] K. Iwama. CNF satisfiability test by counting and polynomial average time. *SIAM J. on Computing*, pages 385–391, 1989.
- [289] F. Jahanian and A.K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Trans. on Software Engineering*, SE-12(9):890–904, Sept. 1986.

- [290] T. Jayasri and D. Basu. An Approach to Organizing Microinstructions Which Minimizes the Width of Control Store Words. *IEEE Trans. on Computers*, C-25:514–521, May 1976.
- [291] R.E. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and AI*, 1:167–187, 1990.
- [292] R.G. Jeroslow. Computation-oriented reductions of predicate to propositional logic. *Decision Support Systems*, 4:183–197, 1988.
- [293] D.S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. of Computer and Systems Sciences*, 9:256–278, 1974.
- [294] D.S. Johnson. More approaches to the traveling salesman guide. *Nature*, 330:525, 1987.
- [295] D.S. Johnson. *Local Optimization and the Traveling Salesman Problem*. In M.S. Paterson, editor, *Lecture Notes in Computer Science*, Vol. 443: *Automata, Languages and Programming*, pages 446–461. Springer-Verlag, Berlin, 1990.
- [296] D.S. Johnson. Private Communications, 1993–1996.
- [297] D.S. Johnson and M.A. Trick, editors. *Clique, Graph Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. DIMACS Series Vol. 26. American Mathematical Society, 1996.
- [298] J.L. Johnson. A neural network approach to the 3-satisfiability problem. *J. of Parallel and Distributed Computing*, 6:435–449, 1989.
- [299] M.D. Johnston. Scheduling with neural networks — the case of the Hubble Space Telescope. In *NASA Memo*, 1989.
- [300] R. R. Johnson. Critical issues in computer architecture design. Private Communication, 1987-1994.
- [301] R.W. Johnson and A.M. McLoughlin. Computer-Aided Discovery of a Fast Matrix-Multiplication Algorithm. *NRL Memorandum Report 3994* May 1979.
- [302] W. Lewis Johnson. Letter from the editor. *SIGART Bulletin*, 2(2):1, April 1991, ACM Press.
- [303] W. Lewis Johnson. Letter from the editor. *SIGART Bulletin*, 2(5):1, Oct. 1991, ACM Press.

- [304] J. R. Josephson, B. Chandrasekaran, J. W. Smith Jr., and M. C. Tanner. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-17(3):445–454, May/June 1987.
- [305] P.C. Jackson Jr. Heuristic search algorithms for the satisfiability problem. Submitted to the third IEEE TAI Conference, Jul. 1991.
- [306] A. E. W. Jones and G. W. Forbes. An adaptive simulated annealing algorithm for global optimization over continuous variables. *Journal of Optimization Theory and Applications*, 6:1–37, 1995.
- [307] Y.C. Ju, B. Rao, and R.A. Saleh. Consistency checking and optimization of macro-models. *IEEE Trans. on CAD*, 10(8):957–967, Aug. 1991.
- [308] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.
- [309] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.
- [310] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Mathematical Sciences Research Center, AT&T Bell Laboratories*, Oct. 1989.
- [311] A. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail Bounds for occupancy and the satisfiability threshold conjecture. *Random Structures and Algorithms* 7, pp. 59–80, 1995.
- [312] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, (4):373–395, 1984.
- [313] G. Kedem and H. Watanabe. Graph optimization techniques for IC layout and compaction. *IEEE Trans. on CAD*, CAD-3:12–20, 1984.
- [314] J. Kella. State Minimization of Incompletely Specified Sequential Machines. *IEEE Trans. on Computers*, C-19:342–348, April 1970.
- [315] S. Kirkpatrick, C.D. Gelat, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

- [316] S. Kirkpatrick, G. Gyorgyi, N. Tishby, and L. Troyansky. The statistical mechanics of k -satisfaction. In *Proceedings of Neural Information Processing Systems*, Nov. 1993.
- [317] L.M. Kirousis and C.H. Papadimitriou. The complexity of recognizing polyhedral scenes. *J. of Computer and System Sciences*, 37:14–38, 1988.
- [318] L. M. Kirousis, E. Kranakis, and D. Krizanc. A better upper bound for the unsatisfiability threshold. In *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 60, 1996.
- [319] J. Kittler and J. Illingworth. Relaxation labeling algorithms — A review. *Image and Vision Computing*, pages 206–216, 1985.
- [320] D. E. Knuth. Estimating the efficiency of backtracking programs. *Mathematics of Computation*, 29(129):121–136, Jan. 1975.
- [321] D. Knuth. Nested satisfiability. *Acta Informatica* 28:1-6, 1990.
- [322] K.L. Kodandapani and E.J. McGrath. A wirelist compare program for verifying VLSI layouts. *IEEE Design and Test of Computers*, 3(3):46–51, 1986.
- [323] R. Kowalski. A proof procedure using connection graphs. *J. ACM*, 22(4):572–595, Oct. 1975.
- [324] M.R. Kramer and J. van Leeuwen. *The Complexity of Wire Routing and Finding Minimum Area Layouts for Arbitrary VLSI Circuits*, volume 2, chapter VLSI Theory, pages 129–146. Jai Press Inc., Greenwich, CT, 1984.
- [325] C.M. Krishna, K.G. Shin, and I.S. Bhandari. Processor tradeoffs in distributed real-time systems. *IEEE Trans. on Computers*, C-36(9):1030–1040, Sept. 1987.
- [326] D. C-L. Ku. DRA1 chip implementation report. Project Report, Dept. of Computer Science, Univ. of Utah, Mar. 1986.
- [327] Oliver Kullmann. A systematic approach to 3-SAT-decision, yielding 3-SAT-decision in less than 1.5045^n steps. *Theoretical Computer Science*, to appear.
- [328] V. Kumar. Algorithms for constraint satisfaction problems: A survey. Technical Report TR-91-28, Dept. of Computer Science, Univ. of Minnesota, 1991.

- [329] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *The AI Magazine*, 13(1):32–44, 1992.
- [330] V. Kumar and Y.-J Lin. A data-dependency based intelligent backtracking scheme for Prolog. *J. of Logic Programming*, 5(2), June 1988.
- [331] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7:49–58, 1970.
- [332] J. D. Laderman. A noncommutative algorithm for multiplying 3×3 matrices using 23 multiplications. *Bull. Amer. Math. Soc.*, 82(1):126–128, 1976.
- [333] E.D. Lagnese and D.E. Thomas. Architectural partitioning for system level synthesis of integrated circuits. *IEEE Trans. on CAD*, 10(7):847–860, July 1991.
- [334] G. Lakhani and R. Varadarajan. A wire-length minimization algorithm for circuit layout compaction. In *Proc. of ISCAS'87*, pages 276–279, May 1987.
- [335] B.S. Landman and R.L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. on Computers*, C-20:1469–1479, Dec. 1971.
- [336] T. Larabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on Computer-Aided Design*, 11(1):4–15, Jan. 1992.
- [337] C. Lassez. Constraint logic programming. *BYTE Magazine*, pages 171–176, Aug. 1987.
- [338] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelii. Algorithms for Synthesis of Hazard-free Asynchronous Circuits. In *Proc. of 28th DAC*, pages 302–308, 1991.
- [339] L. Lavagno, C. Moon, R. Brayton, and A. Sangiovanni-Vincentelli. Solving the State Assignment Problem for Signal Transition Graphs. In *Proc. of 29th DAC*, pages 568–572, 1992.
- [340] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem*. John Wiley & Sons, New York, 1985.
- [341] E.L. Lawler and D.E. Wood. Brance-and-bound methods: a survey. *Operations Research*, 14(4):699–719, Jul.–Aug. 1966.

- [342] C. Y. Lee. Representation of Switching Circuits by Binary Decision Programs. *Bell Systems Technical Journal*, 38:985–999, July 1959.
- [343] C. Lee. An algorithm for path connections and its applications. *IEEE Trans. on Electronic Computers*, VEC-10:346–365, Sept. 1961.
- [344] E.A. Lee. Consistency in dataflow graphs. *IEEE Trans. on Parallel and Distributed Systems*, 2(2):223–235, Apr. 1991.
- [345] R.C.T. Lee. Private Communications, 1992-1993.
- [346] J.P. Lehoczky and L. Sha. Performance of real-time bus scheduling algorithms. *ACM Performance Evaluation Review*, 14(1), May 1986. Special Issue.
- [347] H. R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the Association for Computing Machinery* 25, pages 134–135, 1978.
- [348] G. J. Li and B. W. Wah. How to Cope with Anomalies in Parallel Approximate Branch-and-Bound Algorithms. in *Proc. National Conf. on Artificial Intelligence*, AAAI, pages 212-215, Aug. 1984.
- [349] G.-J. Li. *Parallel Processing of Combinatorial Search Problems*. PhD thesis, School of Electrical Engineering, Purdue University, West Lafayette, Dec. 1985.
- [350] G. J. Li and B. W. Wah. How good are parallel and ordered depth-first searches? In *Proc. Int'l Conf. on Parallel Processing*, pages 992–999, University Park, PA, August 1986. Pennsylvania State Univ. Press.
- [351] G. J. Li and B. W. Wah. Computational efficiency of combinatorial OR-tree searches. *Trans. on Software Engineering*, 16(1):13–31, January 1990.
- [352] G.-J. Li and B. W. Wah. Parallel iterative refining A*: An efficient search scheme for solving combinatorial optimization problems. In *Proc. Int'l Conf. on Parallel Processing*, pages 608–615, University Park, PA, August 1991. Pennsylvania State Univ. Press.
- [353] J. Li and M. Chen. Compiling communication-efficient programs for massively parallel machines. *IEEE Trans. on Parallel and Distributed Systems*, 2(3):361–376, July 1991.
- [354] Y.-Z. Liao and C.K. Wong. An algorithm to compact a VLSI symbolic layout with mixed constraints. *IEEE Trans. on CAD*, CAD-2(2):62–69, 1983.

- [355] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing* 11:329–343, 1982.
- [356] K.J. Lieberherr and E. Specker. Complexity of partial satisfaction. *J. of ACM*, 28:411–421, 1981.
- [357] F.C.H. Lin and R.M. Keller. The gradient model load balancing method. *IEEE Trans. on Software Engineering*, SE-13(1):32–38, Jan. 1987.
- [358] K. J. Lin and C. S. Lin. Automatic Synthesis of Asynchronous Circuits. In *Proc. of 28th DAC*, pages 296–301, 1991.
- [359] S. Lin. Computer solutions of the traveling salesman problem. *Bell Sys. Tech. Journal*, 44(10):2245–2269, Dec. 1965.
- [360] W. M. Lin and V. K. P. Kumar. Parallel architectures for discrete relaxation algorithm. AAAI Workshop on Parallel Algorithms for AI, Detroit, 20 Aug. 1989.
- [361] Y.-J. Lin. *A Parallel Implementation of Logic Programs*. PhD thesis, The Univ. of Texas at Austin, Dept. of Computer Science, May 1988.
- [362] R. P. Lippmann. An introduction to computing with neural net. *IEEE ASSP Magazine*, 4(2):4–22, April 1987.
- [363] Richard J. Lipton. DNA Solution of Hard Computational Problems. *Science*, 268:542–544.
- [364] C. D. Locke. *Best-Effort Decision Making for Real-Time Scheduling*. PhD thesis, Carnegie-Mellon University, May 1985.
- [365] D.W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, 1978.
- [366] F. Luccio. Extending the Definition of Prime Compatibility Classes of States in Incompletely Specified Sequential Machine Reduction. *IEEE Trans. on Computers*, C-18:537–540, Jun. 1969.
- [367] D.G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, 1984.
- [368] N.A. Lynch. Multi-level atomicity — a new correctness criterion for database concurrency control. *ACM Trans. on Database Systems*, 8(4), Dec. 1983.
- [369] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–119, 1977.

- [370] A. El Maftouhi, and W. F. de la Vega. On Random 3-SAT. Manuscript, Laboratoire de Recherche en Informatique, Universite de paris-Sud, France. 1994.
- [371] F. Major, M. Turcotte, D. Gautheret, G. Lapalme, E. Fillion, and R. Cedergren. The combination of symbolic and numerical computation for three-dimensional modeling of RNA. *Sciences*, 253:1255–1260, 1991.
- [372] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment. In *Proc. of ACM/IEEE International Conference on CAD*, 1988.
- [373] J. Malik and D. Maydan. Recovering three-dimensional shape from a single image of curved objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-11(6):555–566, Jun. 1989.
- [374] J.A. Makowsky and A. Sharell. On average case complexity of SAT symmetric distribution. *J. of Logic and Computation*, 5(1):71–92, Feb. 1995.
- [375] D. Marple and A.E. Gamal. Area-delay optimization of programmable logic arrays. Technical report, Stanford University, Sept. 1986.
- [376] T.A. Marsland and M. Campbell. Parallel search of strongly ordered game trees. *ACM Computing Surveys*, 14(4):533–551, Dec. 1982.
- [377] T.A. Marsland and J. Schaeffer. *Computers, Chess, and Cognition*. Springer-Verlag, New York, 1990.
- [378] D.W. Matula, W.G. Marble, and J.D. Isaacson. *Graph coloring algorithms*. In Graph Theory and Computing. R.C. Read, editor, pages 109–122. Academic Press, New York, 1972.
- [379] B. Mazure, L. Sais, and E. Gregoire. Tabu search for SAT. In *Proceedings of CP'95 Workshop on Solving Really Hard Problems*, pages 127–130, 1995.
- [380] D. McAllester. Truth maintenance. In *Proceedings of AAAI'90*, pages 1109–1116, 1990.
- [381] J. T. McCall, J. G. Tront, F. G. Gray, R. M. Haralick, and W. M. McCormack. Parallel computer architectures and problem solving strategies for the consistent labeling problem. *IEEE Trans. On Computers*, C-34(11):973–980, Nov. 1985.

- [382] J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.
- [383] C.R. McLean and C.R. Dyer. An analog relaxation processor. In *Proceedings of the 5th International Conference on Pattern Recognition*, pages 58–60, 1980.
- [384] K. Mehlhorn. *Data Structures and Algorithms: Graph Algorithms and NP-Completeness*. Springer-Verlag, Berlin, 1984.
- [385] H.-M. Méjean, H. Morel, and G. Reynaud. A variational method for analyzing unit clause search. *SIAM J. on Computing*,
- [386] Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs*. Springer-Verlag, 1994.
- [387] M. Minoux, “The unique Horn-satisfiability problem and quadratic Boolean equations,” *Annals of Mathematics and Artificial Intelligence* special issue on connections between combinatorics and logic, J. Franco, M. Dunn, W. Wheeler, (eds.) 1992.
- [388] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of AAAI'90*, pages 17–24, Aug. 1990.
- [389] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.
- [390] D.P. Miranker. *TREAT: A New and Efficient Match Algorithm for AI Production Systems*. Pitman, London, 1990.
- [391] D.P. Miranker and B.J. Lofaso. The organization and performance of a treat-based production system compiler. *IEEE Trans. on Knowledge and Data Engineering*, 3(1):3–10, Mar. 1991.
- [392] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of AAAI'92*, pages 459–465, Jul. 1992.
- [393] M. Mitzenmacher. Tight thresholds for the pure literal rule. SRC Technical note 1997-011, Digital Research Corporation, Systems Research Center, Palo Alto, California, 1997 (obtained from <http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1997-011.html>)..

- [394] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
- [395] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:117–133, 1983.
- [396] B. Monien, E. Speckenmeyer, and O. Vornberger. Superlinear Speedup for Parallel Backtracking.
- [397] S. Mori, C.Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, Jul. 1992.
- [398] P. Morris. The breakout method for escaping from local minima. In *Proc. of the 11th National Conf. on Artificial Intelligence*, pages 40–45, Washington, DC, 1993.
- [399] B.A. Murtagh and M.A. Saunders. MINOS 5.0 user’s guide. Technical Report SOL 83-20, Dept. of Operations Research, Stanford University, Stanford, CA, 1983.
- [400] B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.
- [401] B.A. Nadel and J. Lin. Automobile transmission design as a constraint satisfaction problem: A collaborative research project with ford motor Co. Technical report, Wayne State University, 1990.
- [402] D. Navinchandra. *Exploration and Innovation in Design*. Springer Verlag, Sadeh, 1990.
- [403] D. Navinchandra and D.H. Marks. Layout planning as a consistent labeling optimization problem. In *Proceedings of 4th International Symposium on Robotics and AI in Construction*, 1987.
- [404] A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
- [405] L.M. Ni, C. Xu, and T.B. Gendreau. A distributed drafting algorithm for load balancing. *IEEE Trans. on Software Engineering*, SE-11(10):1153–1161, Oct. 1985.
- [406] A. Nijenhuis and H. S. Wilf. *Combinatorial Algorithms*. Academic Press, New York, 1975.
- [407] N.J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, Palo Alto, California, 1980.

- [408] M. Nishizawa. Partitioning of logic units. *Fujitsu Scientific and Technical Journal*, 7(2):1–13, Jun. 1971.
- [409] D. Pager. On the efficient algorithm for graph isomorphism. *J. of ACM*, 17(4):708–714, Jan. 1970.
- [410] C.H. Papadimitriou and K. Steiglitz. On the complexity of local search for the traveling salesman problem. *SIAM J. on Computing*, 6(1):76–83, 1977.
- [411] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, 1982.
- [412] C.H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual Symposium of the Foundations of Computer Science*, pages 163–169, 1991.
- [413] E. Koutsoupias, and C. H. Papadimitriou. 1992. On the greedy algorithm for satisfiability. *Information Processing Letters* **43**, pp. 53–55.
- [414] C.H. Papadimitriou, Private Communications, 1996.
- [415] P.M. Pardalos and J.B. Rosen. *Constrained Global Optimization: Algorithms and Applications*. Springer Verlag, New York, 1987.
- [416] P. M. Pardalos. *Complexity in numerical optimization*. World Scientific, Singapore and River Edge, N.J., 1993.
- [417] A.M. Patel, N.L. Soong, and R.K. Korn. Hierarchical VLSI routing — an approximate routing procedure. *IEEE Trans. on CAD*, CAD-4(2):121–126, Apr. 1985.
- [418] W. Patterson. *Mathematical Cryptology*. Rowman & Littlefield, New Jersey, 1987.
- [419] P.G. Paulin and J.P. Knight. Force-directed scheduling for the behavioral synthesis of ASIC's. *IEEE Trans. on CAD*, 8(6):661–679, June 1989.
- [420] J. Pearl. *Heuristics*. Addison-Wesley, Reading, 1984.
- [421] D. Pehoushek. SAT problem instances and algorithms. Private Communications, Stanford University, 1992.
- [422] M.A. Penna. A shape form shading analysis for a single perspective image of a polyhedron. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-11(6):545–554, Jun. 1989.

- [423] L.M. Pereira and A. Porto. *Selective Backtracking*, pages 107–114. Academic Press, 1982.
- [424] C. Petrie. Revised dependency-directed backtracking for default reasoning. In *Proceedings of AAAI'87*, pages 167–172, 1987.
- [425] J. Peysakh, *A fast algorithm to convert Boolean expressions into CNF*, IBM Computer Science RC 12913, No. 57971, T.J. Watson, New York, 1987.
- [426] T. Pitassi and A. Urquhart. The complexity of the hajós calculus. *SIAM J. on Disc. Math.*, 8(3):464–483, 1995.
- [427] J. Plotkin, J. Rosenthal, and J. Franco. Correction to probabilistic analysis of the Davis-Putnam Procedure for solving the Satisfiability problem. *Discrete Applied Mathematics*, 17:295–299, 1987.
- [428] D.P. La Potin and S.W. Director. Mason: a global floorplanning approach for VLSI design. *IEEE Trans. on CAD*, CAD-5(4):477–489, Oct. 1986.
- [429] D. K. Pradhan, editor. *Fault-Tolerant Computing: Theory and Practice*. Prentice-Hall, Englewood Cliffs, 1986.
- [430] D. Prawitz. An improved proof procedure. *Theoria*, 26(2):102–139, 1960.
- [431] D. Pretolani. A linear time algorithm for unique Horn satisfiability. *Information Processing Letters* 48:61–66, 1993.
- [432] D. Pretolani. Efficiency and stability of hypergraph SAT algorithms. *DIMACS Series Volume: Clique, Graph Coloring, and Satisfiability — Second DIMACS Implementation Challenge*. Editors: D.S. Johnson and M.A. Trick, American Mathematical Society, 1996.
- [433] P. Purdom. Survey of average time SAT performance. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [434] P.W. Purdom. Tree size by partial backtracking. *SIAM J. on Computing*, 7(4):481–491, Nov. 1978.
- [435] P.W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.
- [436] P.W. Purdom. Solving satisfiability with less searching. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6:510–513, 1984.

- [437] P.W. Purdom. A survey of average time analyses of satisfiability algorithms. *J. of Information Processing*, 13(4):449–455, 1990.
- [438] P.W. Purdom. Average time for the full pure literal rule. *Information Sciences*, 1994 78:269–291.
- [439] P.W. Purdom and C.A. Brown. An analysis of backtracking with search rearrangement. *SIAM J. on Computing*, 12(4):717–733, Nov. 1983.
- [440] P.W. Purdom and C.A. Brown. The pure literal rule and polynomial average time. *SIAM J. on Computing*, 14:943–953, 1985.
- [441] P.W. Purdom and C.A. Brown. Polynomial-average-time satisfiability problems. *Information Science*, 41:23–42, 1987.
- [442] P.W. Purdom, C.A. Brown, and E.L. Robertson. Backtracking with multi-level dynamic search rearrangement. *Acta Informatica*, 15:99–113, 1981.
- [443] P.W. Purdom and G.N. Haven. Backtracking and probing. Technical Report No. 387, Dept. of Computer Science, Indiana University, Aug. 1993.
- [444] P.W. Purdom and G.N. Haven. Probe Order Backtracking. Accepted in *SIAM J. on Computing*, to appear in 1997.
- [445] R. Puri and J. Gu. An efficient algorithm to search for minimal closed covers in sequential machines. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 12(6):737–745, Jun. 1993.
- [446] R. Puri and J. Gu. An efficient algorithm for computer microword length minimization. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 12(10):1449–1457, Oct. 1993.
- [447] R. Puri and J. Gu. Area efficient synthesis of asynchronous interface circuits. In *Proc. of IEEE International Conference on Computer Design*, 1994.
- [448] R. Puri and J. Gu. A divide-and-conquer approach for asynchronous circuit design. In *Proceedings of the 7th IEEE International Symposium on High-Level Synthesis*, pages 118–125, May 1994.
- [449] R. Puri and J. Gu. A BDD SAT solver for satisfiability testing. *Annals of Mathematics and Artificial Intelligence*, Vol. 17, pp. 315–337, 1996.

- [450] C. D. V. P. Rao and N. N. Biswas. On the Minimization of Word-width in the Control Memory of a Microprogrammed Digital Computer. *IEEE Trans. on Computers*, C-32(9):863–868, Sept. 1983.
- [451] C. V. S. Rao and N. N. Biswas. Minimization of Incompletely Specified Sequential Machines. *IEEE Trans. on Computers*, C-24:1089–1100, Nov. 1975.
- [452] R.C. Read and D.G. Corneil. The graph isomorphism disease. *J. of Graph Theory*, 1:339–363, 1977.
- [453] G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. In *Proc. of ICALP'86*, pages 314–323. Springer LNCS 226, 1986.
- [454] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro. A new symbolic channel router: YACR2. *IEEE Trans. on CAD*, CAD-4(3):208–219, July 1985.
- [455] M. Reichling. A simplified solution of the n queens' problem. *Information Processing Letters*, Vol. 25:253–255, June 1987.
- [456] M. Resende and T. Feo. A GRASP for MAX-SAT. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [457] R.L. Rivest. *Cryptography*. In Handbook of Theoretical Computer Science. J.V. Leeuwen, editor, chapter 13, pages 719–756. The MIT Press, Cambridge, 1990.
- [458] R.L. Rivest and C.M. Fiduccia. A 'greedy' channel router. In *Proc. of the 19th ACM/IEEE Design Automation Conference*, pages 418–424, Jun. 1982.
- [459] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, pages 23–41, 1965.
- [460] A. Rosenfeld. Computer vision: Basic principles. *Proceedings of the IEEE*, 76(8):863–868, Aug. 1988.
- [461] A. Rosenfeld, R. A. Hummel, and S. W. Zucker. Scene labeling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-6(6):420–433, June 1976.
- [462] W. Rosiers and M. Bruynooghe. Empirical study of some constraint satisfaction problems. In *Proceedings of AIMSA'86*, North Holland, Sept. 1986.

- [463] A.E. Ruehli, P.K. Wolff, and G. Goertzel. Analytical power/timing optimization technique for digital system. In *Proc. of the 14th ACM/IEEE Design Automation Conference*, pages 142–146, Jun. 1977.
- [464] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, 1995.
- [465] R.L. Russo. On the tradeoff between logic performance and circuit-to-pin ratio for LSI. *IEEE Trans. on Computers*, C-21:147–153, 1972.
- [466] R.L. Russo, P.H. Oden, and P.K. Wolff. A heuristic procedure for the partitioning and mapping of computer logic graphs. *IEEE Trans. on Computers*, C-20:1455–1462, Dec. 1971.
- [467] Y.G. Saab and V. B. Rao. Combinatorial optimization by stochastic evolution. *IEEE Transactions on CAD*, CAD-10(4):525–535, Apr. 1991.
- [468] A. Saldanha, T. Villa, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. A Framework for Satisfying Input and Output Encoding Constraints. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 170–175, 1991.
- [469] A. Samal. *Parallel Split-Level Relaxation*. PhD thesis, Dept. of Computer Science, Univ. of Utah, Aug. 1988.
- [470] A. Samal and T. C. Henderson. Performance of arc consistency algorithms on the Cray. Technical Report UUCS-TR-87-017, Dept. of Computer Science, Univ. of Utah, July 23 1987.
- [471] A. Samal and T.C. Henderson. Parallel consistent labeling algorithms. *International Journal of Parallel Programming*, 1988.
- [472] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 216–226, 1978.
- [473] W.L. Schiele. Improved compaction by minimized length of wires. In *Proc. of the 20th ACM/IEEE Design Automation Conference*, pages 121–127, 1983.
- [474] I. Schiermeyer. Solving 3-Satisfiability in less than $O(1.579^n)$ steps. *Lecture Notes in Computer Science* 702, pages 379–394, 1993.
- [475] I. Schiermeyer. Pure literal lookahead: an $O(1.497^n)$ 3-Satisfiability algorithm. In *Proc. of the Workshop on Satisfiability*, Università delgi Studi, Siena, Italy, pages 63–72, 1996.

- [476] J. S. Schlipf, F. Annexstein, J. Franco, and R. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters* 54, pages 133–137, 1995.
- [477] B.M. Schwartzschild. Statistical mechanics algorithm for Monte Carlo optimization. *Physics Today*, 35:17–19, 1982.
- [478] P. Schwarz and A. Spector. Synchronizing shared abstract types. *ACM Trans. on Computer Systems*, Aug. 1984.
- [479] M. G. Scutella. A note on Dowling and Gallier’s top-down algorithm for propositional Horn satisfiability. *Journal of Logic Programming* 8, pages 265–273, 1990.
- [480] C. Sechen and A.L. Sangiovanni-Vincentelli. The TIMBERWOLF placement and routing package. In *Proc. of the 1984 Custom Integrated Circuit Conf.*, May 1984.
- [481] B. Selman. Private Communications, Aug. 1992.
- [482] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of AAAI’92*, pages 440–446, Jul. 1992.
- [483] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proc. of the 13th Int'l Joint Conf. on Artificial Intelligence*, pages 290–295, 1993.
- [484] B. Selman, H.A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series Volume: Clique, Graph Coloring, and Satisfiability — Second DIMACS Implementation Challenge*. American Mathematical Society, pp. 290–295, 1996.
- [485] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proc. of the 12th National Conf. on Artificial Intelligence*, pages 337–343, Seattle, July 1994.
- [486] B. Selman, Private Communication, 1995.
- [487] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. SIS : A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Dept. of EECS, Univ. of California, Berkeley, May, 1992.
- [488] M.J. Shensa. A computational structure for the propositional calculus. In *Proceedings of IJCAI*, pages 384–388, 1989.

- [489] R.C.-J. Shi, A. Vannelli, and J. Vlach. An improvement on Karmarkar's algorithm for integer programming. COAL Bulletin of the Mathematical Programming Society, 21:23–28, 1992.
- [490] H. Shin, A.L. Sangiovanni-Vincentelli, and C.H. Sequin. Two-dimensional compaction by 'zero refining'. In *Proc. of the 23rd ACM/IEEE Design Automation Conference*, pages 115–122, Jun. 1986.
- [491] K.G. Shin and M.-S. Chen. On the number of acceptable task assignments in distributed computing systems. *IEEE Trans. on Computers*, 39(1):99–110, Jan. 1990.
- [492] P. Siegel. *Representation et Utilization de la Connaissances en Calcul Propositionnel*. PhD thesis, University Aix-Marseille II, 1987.
- [493] J.C. Simon. Off-line cursive word recognition. *Proceedings of the IEEE*, 80(7):1150–1161, Jul. 1992.
- [494] R. Sosić and J. Gu. Quick n -queen search on VAX and Bobcat machines. CS547 AI Course Project, Winter Quarter, Feb. 1988.
- [495] R. Sosić and J. Gu. How to search for million queens. Technical Report UUCS-TR-88-008, Dept. of Computer Science, Univ. of Utah, Feb. 1988.
- [496] R. Sosić and J. Gu. A polynomial time algorithm for the n -queens problem. *SIGART Bulletin*, 1(3):7–11, Aug. 1990, ACM Press.
- [497] R. Sosić and J. Gu. Fast search algorithms for the n -queens problem. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-21(6):1572–1576, Nov./Dec. 1991.
- [498] R. Sosić and J. Gu. 3,000,000 queens in less than one minute. *SIGART Bulletin*, 2(2):22–24, Apr. 1991, ACM Press.
- [499] R. Sosić and J. Gu. A parallel local search algorithm for satisfiability (SAT) problem. *Submitted for publication*, 1993.
- [500] R. Sosić and J. Gu. A parallel local search algorithm for the n -queen problem. *Submitted for publication*, 1993.
- [501] R. Sosić and J. Gu. Efficient local search with conflict minimization. *IEEE Trans. on Knowledge and Data Engineering*, 6(5):661–668, Oct. 1994.

- [502] R. Sosič, J. Gu, and R. Johnson. The Unison algorithm: Fast evaluation of Boolean expressions. Accepted for publication in *Communications of the ACM* in 1992. *ACM Transactions on Design Automation of Electronic Systems*, (1)4:456–477, Oct. 1996.
- [503] R. Sosič, J. Gu, and R. Johnson. A universal Boolean evaluator. *IEEE Trans. on Computers*, accepted for publication in 1992.
- [504] J. Soukup. Circuit layout. In *Proceedings of the IEEE*, pages 1281–1304, Oct. 1981.
- [505] E. Speckenmeyer. Personal Communication. 1996.
- [506] R. Stallman and G.J. Sussman. Forward reasoning and dependency directed backtracking. *Artificial Intelligence*, 9(2):135–196, 1977.
- [507] J. Stankovic, K. Ramamritham, and S. Cheng. Evaluation of a bidding algorithm for hard real-time distributed systems. *IEEE Trans. on Computers*, C-34(12):1130–1143, Dec. 1985.
- [508] J.A. Stankovic. Real-time computing systems: The next generation. Manuscript. Feb. 18, 1988.
- [509] L. Sterling and E. Shapiro. *The Art of Prolog, Advanced Programming Techniques*. The MIT Press, Cambridge, Massachusetts, 1986.
- [510] H.S. Stone and P. Sipala. The average complexity of depth-first search with backtracking and cutoff. *IBM J. Res. Develop.*, 30(3):242–258, May 1986.
- [511] H.S. Stone and J.M. Stone. Efficient search techniques – an empirical study of the n-queens problem. *IBM J. Res. Develop.*, 31(4):464–474, July 1987.
- [512] C.Y. Suen, M. Berthod, and S. Mori. Automatic recognition of hand printed characters – the state of the art. *Proceedings of the IEEE*, 68(4):469–487, Apr. 1980.
- [513] S. Sutanthavibul, E. Shragowitz, and J.B. Rosen. An analytical approach to floorplan design and optimization. *IEEE Trans. on CAD*, 10(6):761–769, June 1991.
- [514] R. P. Swaminathan and D. K. Wagner. The arborescence-realization problem. *Discrete Applied Mathematics* 59, pages 267–283, 1995.

- [515] M. Takashima, T. Mitsuhashi, T. Chiba, and K. Yoshida. Programs for verifying circuit connectivity of MOS/VLSI artwork. In *Proc. of the 19th ACM/IEEE Design Automation Conference*, pages 544–550, 1982.
- [516] A. Thayse and M. Davio, *Boolean differential calculus and its application to switching theory*, IEEE Trans. on Computers **22** (1973), 409–420.
- [517] H. Tokuda, J. Wendorf, and H. Wang. Implementation of a time driven scheduler for real-time operating systems. In *Proc. of the Real-Time Systems Symposium*, Dec. 1987.
- [518] A. Törn and A. Žilinskas. *Global Optimization*. Springer-Verlag, 1989.
- [519] P. P. Trabado, A. Lloris-Ruiz, and J. Ortega-Lopera. Solution of Switching Equations Based on a Tabular Algebra. *IEEE Trans. on Computers*, C-42:591–596, May 1993.
- [520] K. Truemper. Alpha-balanced graphs and matrices and GF(3)-representability of matroids. *Journal of Combinatorial Theory B* **32**, pages 112–139, 1982.
- [521] K. Truemper. Monotone Decomposition of Matrices. Technical Report UTDCS-1-94, University of Texas at Dallas. 1994.
- [522] K. Truemper. Polynomial algorithms for problems over d-systems. Presented in the 3rd International Symposium on AI & Mathematics, Jan. 1994.
- [523] K. Truemper. In preparation, expected 1997. Effective Logic Computation.
- [524] G.S. Tseitin. *On the Complexity of Derivations in Propositional Calculus*. In Structures in Constructive Mathematics and Mathematical Logic, Part II, A.O. Slisenko, ed., pages 115–125. 1968.
- [525] K. J. Turner. *Computer Perception of Curved Objects Using a Television Camera*. PhD thesis, Univ. Edinburgh, 1974.
- [526] J.D. Tygar and R. Ellickson. Efficient netlist comparison using hierarchy and randomization. In *Proc. of the 22nd ACM/IEEE Design Automation Conference*, pages 702–708, 1985.
- [527] J. E. Tyler. Parallel computer architectures and problem solving strategies for the consistent labeling problem. Master's thesis, Dept. of Elec. Engg., Virginia Polytech. Inst. State Univ., Blacksburg, Virginia, Oct. 1983.

- [528] J.D. Ullman. *Principles of Database Systems*. Computer Science Press, Rockville, 1982.
- [529] J.R. Ullman. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, Jan. 1976.
- [530] J.R. Ullman. A binary n -gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *The Computer Journal*, 20(2):141–147, Feb. 1977.
- [531] S.D. Urban and L.M.L. Delcambre. Constraint analysis: A design process for specifying operations on objects. *IEEE Trans. on Knowledge and Data Engineering*, 2(4):391–400, Dec. 1990.
- [532] A. Urquhart. Hard examples for resolution. *J. of ACM*, 34:209–219, 1987.
- [533] A. Urquhart. The complexity of gentzen systems for propositional logic. *Theoretical Computer Science*, 66:87–97, 1989.
- [534] A. Urquhart. The relative complexity of resolution and cut-free gentzen systems. *Annals of Mathematics and Artificial Intelligence*, 6:157–168, 1992.
- [535] A. Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, 1995.
- [536] M. Valtorta. Response to “Explicit Solutions to the N-Queens Problem for all N.” *SIGART Bulletin*, 2(4):4, Aug. 1991, ACM Press.
- [537] M. van der Woude and X. Timermans. Compaction of hierarchical cells with minimum and maximum compaction constraints. In *Proc. Int'l Symposium on Circuits and Systems*, pages 1018–1021, 1983.
- [538] P. Vanbekbergen, G. Goossens, F. Catthoor, and H. De Man. Optimized Synthesis of Asynchronous Control Circuits from Graph-Theoretic Specifications. *IEEE Trans. on CAD*, 11(11):1426–1438, Nov. 1992.
- [539] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A Generalized State Assignment Theory for Transformations on Signal Transition Graphs. In *Proc. of ICCAD*, pages 112–117, 1992.
- [540] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A Generalized State Assignment Theory for Transformations on Signal Transition Graphs. *J. of VLSI Signal processing*, 1993. In Press.

- [541] B. W. Wah, G. J. Li, and C. F. Yu. Multiprocessing of combinatorial search problems. *IEEE Computer*, 18(6):93–108, June 1985. Also in *Tutorial: Computers for Artificial Intelligence Applications*, ed. B. W. Wah, IEEE Computer Society, 1986, pp. 173-188.
- [542] B. Wah and G.J. Li. *Computers for Artificial Intelligence Applications*. IEEE Computer Society Press, Washington D. C., 1986.
- [543] B.W. Wah, editor. *New Computers for Artificial Intelligence Processing*. *IEEE Computer*, volume 20, number 1. IEEE Computer Society Press, 1987.
- [544] B.W. Wah, M.B. Lowrie, and G.-J. Li. Computers for symbolic processing. *Proceedings of the IEEE*, 77(4):509–540, Apr. 1989.
- [545] B. W. Wah, G. J. Li, and C. F. Yu. Multiprocessing of combinatorial search problems. In L. Kanal, V. Kumar, and P. S. Gopalakrishnan, editors, *Parallel Algorithms for Machine Intelligence and Pattern Recognition*, pages 102–145. Springer-Verlag, New York, NY, 1990.
- [546] B. W. Wah and L.-C. Chu. Combinatorial search algorithms with meta-control: Modeling and implementations. *Int'l J. of Artificial Intelligence Tools*, 1(3):369–397, September 1992.
- [547] B. W. Wah and Y. Shang. A comparison of a class of IDA* search algorithms. *Int'l J. of Artificial Intelligence Tools*, 3(4):493–523, October 1995.
- [548] B. W. Wah and Y. Shang. A discrete lagrangian-based global-search method for solving satisfiability problems. In Ding-Zhu Du, Jun Gu, and Panos Pardalos, editors, *Proc. of the DIMACS Workshop on Satisfiability Problem: Theory and Applications*. American Mathematical Society, March 1996.
- [549] B. W. Wah and C. F. Yu. Stochastic modeling of branch-and-bound algorithms with best-first search. *Trans. on Software Engineering*, SE-11(9):922–934, September 1985.
- [550] B. Wah and Y.-J. Chang. Trace-Based Methods for Solving Nonlinear Global Optimization and Satisfiability Problems. *J. of Global Optimization*. Submitted in July 1995 and accepted to appear in 1996.
- [551] D. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI271, MIT, Nov. 1972.
- [552] D. Waltz. *Understanding Line Drawings of Scenes with Shadows*. In P. H. Winston, *The Psychology of Computer Vision*, chapter 2, pages 19–92. McGraw-Hill Book Company, New York, 1975.

- [553] W. Wang, J. Gu, and T. C. Henderson. A pipelined architecture for parallel image relaxation operations. *IEEE Trans. on Circuits and Systems*, CAS-34(11):1375–1384, Nov. 1987.
- [554] W. Wang, J. Gu, and K.F. Smith. A regular layout for concurrent discrete relaxation computation. In *Proceedings of 1987 IEEE International CAS Symposium*, pages 927–933, May 1987.
- [555] W. Wang and C.K. Rushforth. An adaptive local search algorithm for channel assignment problem. *IEEE Trans. on Vehicular Technology*, Vol. 45, No. 3, pp. 459–466, Aug. 1996.
- [556] W. Wang and C.K. Rushforth. Structured partitioning for channel assignment problem. *IEEE Trans. on Vehicular Technology*, 1996.
- [557] M. B. Wells. *Elements of Combinatorial Computing*. Pergamon Press, Oxford, 1971.
- [558] H.P. Williams. Linear and integer programming applied to the propositional calculus. *Systems Research and Information Sciences*, 2:81–100, 1987.
- [559] P.H. Winston. *Artificial Intelligence*. Addison-Wesley, Reading, 1984.
- [560] E.E. Witte, R.D. Chamberlain, and M.A. Franklin. Parallel simulated annealing using speculative computation. *IEEE Trans. on Parallel and Distributed Systems*, 2(4):483–494, Oct. 1991.
- [561] L. Wos, G. A. Robinson, and D. Carson. Efficiency and completeness of the set of support strategy. *Journal of the ACM*, 12:536–541, 1965.
- [562] L.C. Wu and C.Y. Tang. Solving the satisfiability problem by using randomized approach. *Information Processing Letters*, 41:187–190, 1992.
- [563] T. Y. Young and K. S. Fu, editors. *Handbook of Pattern Recognition and Image Processing*. Academic Press, Orlando, 1986.
- [564] C. F. Yu. *Efficient Combinatorial Search Algorithm*. PhD thesis, School of Electrical Engineering, Purdue University, West Lafayette, IN, Dec. 1986.
- [565] C. F. Yu and B. W. Wah. Stochastic modeling of branch-and-bound algorithms with best-first search. *IEEE Trans. on Software Engineering*, 11(9):922–934, Sept 1985.

- [566] C.F. Yu and B.W. Wah. Efficient branch-and-bound algorithms on a two-level memory system. *IEEE Trans. on Software Engineering*, 14(9):1342–1356, Sept 1988.
- [567] R. Zabih and D. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of AAAI'88*, pages 155–160, 1988.
- [568] V.N. Zemlyachenko, N.M. Korneeko, and R.I. Tyshkevich. Graph isomorphism problem. *J. of Soviet Mathematics*, 29:1426–1481, 1985.
- [569] S. Zhang and A. G. Constantinides. Lagrange programming neural networks. *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 39(7):441–452, 1992.
- [570] S. Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Trans. on Software Engineering*, SE-14(9):1327–1341, Sept. 1988.
- [571] S. W. Zucker, R. A. Hummel, and A. Rosenfeld. An application of relaxation labeling to line and curve enhancement. *IEEE Trans. on Computers*, C-26:394–403, 922–929, 1977.

The Steiner Ratio of L_p -planes

Jens Albrecht

Institute of Mathematics and Computer Science

University of Greifswald, Germany

E-mail: albrecht@mail.uni-greifswald.de

Dietmar Cieslik

Institute of Mathematics and Computer Science

University of Greifswald, Germany

E-mail: cieslik@mail.uni-greifswald.de

Contents

1	Introduction	573
2	Well-known Bounds and Exact Values	576
3	Better upper bounds	579
4	Sets with four elements	580
5	The Steiner ratio of dual planes	584
6	Concluding remarks	584
References		

1 Introduction

Starting with the famous book "What is Mathematics" by Courant and Robbins the following problem has been popularized under the name of

Steiner: For a given finite set of points in a metric space find a network which connects all points of the set with minimal length. Such a network must be a tree, which is called a Steiner Minimal Tree (SMT). It may contain vertices other than the points which are to be connected. Such points are called Steiner points.¹ A classical survey of this problem in the Euclidean plane was given by Gilbert and Pollak [23]. An updated one can be found in [27].

Obviously, these problems depend essentially on the way how the distances in the plane are determined. In the present paper we consider planes with p -norm, defined in the following way: For the points $X = (x_1, x_2)$ and $Y = (y_1, y_2)$ of the affine plane A_2 we define the distance by

$$\rho(X, Y) = \rho_p(X, Y) = (|x_1 - y_1|^p + |x_2 - y_2|^p)^{1/p}$$

where $1 \leq p < \infty$ is a real number. If p runs to infinity we get the so-called Maximum distance

$$\rho_\infty(X, Y) = \max\{|x_1 - y_1|, |x_2 - y_2|\}$$

In each case, ρ_p , $1 \leq p \leq \infty$, defines a metric and we obtain a two-dimensional Banach space, written by \mathcal{L}_p^2 , whereby the norm $\|.\|_p$ is derived from the distance by

$$\|X\|_p = \rho_p(X, O),$$

where O is the origin of A_2 . Then

$$B(p) = \{X : \|X\|_p \leq 1\}$$

defines the unit balls of the plane. Two planes \mathcal{L}_p^2 and \mathcal{L}_q^2 are called dual, and the values p and q are called conjugated if $\frac{1}{p} + \frac{1}{q} = 1$, that means, $q = p/(p-1)$. In this sense, the planes \mathcal{L}_1^2 and \mathcal{L}_∞^2 are dual, and the Euclidean plane \mathcal{L}_2^2 is self-dual.

A (finite) graph $G = (V, E)$ with the set V of vertices and the set E of edges is embedded in this space in the sense that

- V is a finite set of points in the space;

¹The history of Steiner's Problem started with P.Fermat [17] early in the 17th century and C.F.Gauß [22] in 1836. At first perhaps with the famous book *What is Mathematics* by R.Courant and H.Robbins in 1941, this problem became popularized under the name of Steiner.

- Each edge $\underline{XY} \in E$ is a segment $\{tX + (1-t)Y : 0 \leq t \leq 1\}$, $X, Y \in V$; and
- The length of G is defined by

$$L(G) = L_p(G) = \sum_{\underline{XY} \in E} \rho_p(X, Y).$$

Now, **Steiner's Problem of Minimal Trees** is the following:
Given a finite set N of points in the space \mathcal{L}_p^2 .
Find a connected graph $G = (V, E)$ embedded in the space such that $N \subseteq V$ and $L_p(G)$ is minimal as possible.

A solution of Steiner's Problem is called a Steiner Minimal Tree (SMT) for N in \mathcal{L}_p^2 . The vertices in the set $V \setminus N$ are called Steiner points. We may assume that for any SMT $T = (V, E)$ for N the following holds: The degree of each Steiner point is at least three and

$$|V \setminus N| \leq |N| - 2. \quad (1)$$

Moreover, Du and Liu [29] show that for $1 < p < \infty$ the degree of a Steiner point is exactly three, and in (1) equality holds if and only if all given points are of degree one. In \mathcal{L}_1^2 and in \mathcal{L}_∞^2 we find Steiner points of degree four.²

If we don't allow Steiner points, that is if we connect certain pairs of given points only, then we refer to a Minimum Spanning Tree (MST). Starting with Boruvka in 1926 and Kruskal in 1956 Minimum Spanning Trees have a well-documented history [24] and effective constructions [3].

A minimum spanning tree in a graph $G = (N, E)$ with a positive length-function $f : E \rightarrow \mathbb{R}$, can be found with the help of Kruskal's [28] well-known method:

1. Start with the forest $T = (N, \emptyset)$;
2. Sequentially choose the shortest edge that does not form a circle with already chosen edges;
3. Stop when all vertices are connected, that is when $|N| - 1$ edges have been chosen.

²For a complete discussion of the combinatorial structure of SMT's compare [7].

Then an MST for a finite set N of points in \mathcal{L}_p^2 can be found obtaining the graph $G = (N, \binom{N}{2})$ with the length-function $f(\underline{XY}) = \rho_p(X, Y)$. Hence, we can find an MST for a finite set of points in a metric space in fast time.

There also exist algorithms to construct SMT's in \mathcal{L}_p^2 , see [9], but they need exponential time. This is not strange, because Steiner's Problem in the Euclidean plane \mathcal{L}_2^2 and in the plane with rectilinear norm \mathcal{L}_1^2 is \mathcal{NP} -hard, [19], [20], [21].

The relative defect, which describes the length of an SMT divided by the length of an MST, is given in the **Steiner Ratio**:

$$m(2, p) = \inf \left\{ \frac{L(\text{SMT for } N)}{L(\text{MST for } N)} : N \subset \mathcal{L}_p^2 \text{ a finite set} \right\}.$$

$m(2, p)$ is a measure of how good an MST as an approximation of Steiner's Problem in the space \mathcal{L}_p^2 is. It is not hard to see that we have $1 \geq m(2, p) \geq 1/2$, see [23]. In other terms, an MST is an approximation of an SMT with a length at most twice the length of an SMT.

2 Well-known Bounds and Exact Values

As an introductory example consider three points which form the nodes of an equilateral triangle of unit side length in the Euclidean plane. An MST for these points has length 2. An SMT uses one Steiner point and has length $3 \cdot \sqrt{1/3} = \sqrt{3}$. So we have an upper bound for the Steiner ratio of the Euclidean plane, namely $m(2, 2) \leq \sqrt{3}/2 = 0.86602\dots$

A long-standing conjecture, given by Gilbert and Pollak in 1968, said that equality holds. Many persons have tried to show this: Pollak [30] and Du, Yao, Hwang [16] have shown that the conjecture is valid for sets N consisting of $n = 4$ points; Du, Hwang, Yao [15] stated this result to the case $n = 5$, and Rubinstein, Thomas [31] have done the same for the case $n = 6$.

On the other hand, many attempts have been made to estimate the Steiner ratio for the Euclidean plane from below:

$$\begin{aligned} \geq 1/\sqrt{3} &= 0.57735\dots && \text{Graham, Hwang [25]} \\ \geq \sqrt{2\sqrt{3} + 2 - (7 + 2\sqrt{3})} &= 0.74309\dots && \text{Chung, Hwang [5]} \\ \geq 4/5 &= 0.8 && \text{Du, Hwang [12]} \\ &\geq 0.82416\dots && \text{Chung, Graham [4]} \end{aligned}$$

Finally, Du and Hwang created a lot of new methods and succeeded in proving the Gilbert-Pollak conjecture completely:

Theorem 2.1 (*Du, Hwang [13], [14]*) *The Steiner Ratio of the Euclidean plane equals*

$$m(2, 2) = \frac{\sqrt{3}}{2} = 0.86602 \dots \quad (2)$$

Another example: The unit ball of \mathcal{L}_1^2 , is the convex hull of $N = \{\pm(1, 0), \pm(0, 1)\}$. The distance of any two different points in N equals 2. Hence, an MST for N has the length 6. Conversely, an SMT for N with the Steiner point $(0, 0)$ has the length 4. This implies $m(2, 1) \leq 2/3$. Moreover,

Theorem 2.2 (*Hwang [26]*) *The Steiner ratio of the planes which unit ball is a parallelogram equals*

$$m(2, 1) = m(2, \infty) = \frac{2}{3} = 0.6666 \dots \quad (3)$$

Du and Liu determined an upper bound for the Steiner ratio of \mathcal{L}_p -planes, using direct calculations of the ratio between the length of SMT's and the length of MST's for sets with three elements:

Theorem 2.3 (*Du, Liu [29]*) *The following is true for the Steiner ratio of the \mathcal{L}_p -planes:*

$$m(2, p) \leq \frac{(2^p - 1)^{1/p} + (2^q - 1)^{1/q}}{4}, \quad (4)$$

where q is the conjugated number to p . Consequently,

$$m(2, p) \leq \frac{\sqrt{3}}{2} = 0.866025 \dots \quad (5)$$

for each number p . Moreover, $m(2, p) = \sqrt{3}/2$ if and only if $p = 2$.

Another upper bound is given by

Theorem 2.4 (*Cieslik [6]*) *The following inequalities are true for the Steiner ratio of the \mathcal{L}_p -planes:*

$$m(2, p) \leq \frac{4}{3} \cdot 2^{-1/p} \quad (6)$$

if $p \leq 2$. And

$$m(2, p) \leq \frac{2}{3} \cdot 2^{1/p} \quad (7)$$

if $p \geq 2$.³

Now, we have two upper bounds for the Steiner ratio of \mathcal{L}_p^2 , and we are interested in lower bounds. Here it is senseless to investigate specific sets of given points. With methods, that are of another kind than those that we discuss, we get

Theorem 2.5 (*Cieslik [6]*) *The following is true for the Steiner ratio of the \mathcal{L}_p -planes:*

$$m(2, p) \geq \begin{cases} \frac{2^{1/p}}{\frac{3}{2}} & \text{if } 1 \leq p \leq \frac{\ln 16}{\ln 13.5} \\ \frac{\sqrt{6}}{2} \cdot 2^{-1/p} & \text{if } \frac{\ln 16}{\ln 13.5} \leq p \leq 2. \end{cases} \quad (8)$$

We can find bounds for $m(2, p)$, $p \geq 2$, if we replace p by $p/(p - 1)$ on the right side.

This fact implies: $m(2, p) \geq \sqrt[4]{1/6} = 0.63894\dots$ for each number p . But a general lower bound for all \mathcal{L}_p -planes can be formulated more sharply:

Theorem 2.6 (*Gao, Du, Graham [18]*) *For the Steiner ratio of \mathcal{L}_p -planes the following is true:*

$$m(2, p) \geq \frac{2}{3} = 0.66\dots \quad (9)$$

Equality holds if and only if $p = 1$ or $p = \infty$.

All these bounds are not bad. For instance we have found the following estimates for the Steiner ratio of \mathcal{L}_4^2 :

$$0.72823\dots = \sqrt{\frac{3}{8}} \cdot \sqrt{\sqrt{2}} \leq m(2, 4) \leq \frac{2}{3} \cdot \sqrt{\sqrt{2}} = 0.79280\dots \quad (10)$$

We will find several better bounds in many planes.

³That means, the last inequality follows from (6), if we replace p by the conjugated number $p/(p - 1)$ on the right side.

3 Better upper bounds

The proof of 2.3 used a specific triangle. Now, we will use a triangle which has a side parallel to the line $\{(x, x) : x \in \mathbb{R}\}$. Let $1 < p < \infty$ and $A = (0, 1)$, $B = (1, 0)$ and $C = (x_p, x_p)$. We want that the triangle spanned by A , B and C is equilateral and, additionally, x_p lies between 1 and 2. Hence, x_p is a zero of the function f whereby

$$f(x) = x^p + (x - 1)^p - 2.$$

Of course, f is a strictly monotone increasing and continuous function. Hence, $f(1) = -1$ and $f(2) = 2^p - 1 > 0$ imply the existence and uniqueness of x_p . Then

$$L(\text{MST for } \{A, B, C\}) = 2 \cdot 2^{1/p}.$$

We choose $S = (z_p, z_p)$ as a Steiner point with $\frac{1}{2} \leq z_p \leq 1$, whereby z_p minimizes the function g ⁴:

$$\begin{aligned} g(z) &= \rho(A, S) + \rho(B, S) + \rho(C, S) \\ &= 2(z^p + (1 - z)^p)^{1/p} + (x_p - z) \cdot 2^{1/p}. \end{aligned}$$

The derivation of g is

$$g'(z) = 2(z^p + (1 - z)^p)^{(1/p)-1} [z^{p-1} - (1 - z)^{p-1}] - 2^{1/p}$$

and

$$\begin{aligned} g''(z) &= 2\left(\frac{1}{p} - 1\right)(z^p + (1 - z)^p)^{(1/p)-2} p [z^{p-1} - (1 - z)^{p-1}]^2 \\ &\quad + 2(z^p + (1 - z)^p)^{(1/p)-1}(p - 1) [z^{p-2} + (1 - z)^{p-2}] \\ &= 2(p - 1)(z^p + (1 - z)^p)^{(1/p)-2} \cdot [(z^p + (1 - z)^p)(z^{p-2} + (1 - z)^{p-2}) \\ &\quad - (z^{p-1} - (1 - z)^{p-1})^2] \\ &= \dots [z^p(1 - z)^{p-2} + (1 - z)^p z^{p-2} + 2z^{p-1}(1 - z)^{p-1}] \\ &> 0. \end{aligned}$$

⁴ g is the so-called Fermat function, which describes the behaviour of the problem to find a point such that the sum of its distances to a finite number of given points is minimal as possible.

Since it holds $g'(\frac{1}{2}) = -2^{1/p}$ and $g'(1) = 2 - 2^{1/p} > 0$, we have the existence and the uniqueness of z_p . The quantity $g(z_p)/2^{1+1/p}$ is an upper bound for the Steiner ratio. Consequently, we have proved

Theorem 3.1 *Let $1 < p < \infty$ and let x_p, z_p be values defined above. Then*

$$m(2, p) \leq \left(\frac{z_p^p + (1 - z_p)^p}{2} \right)^{1/p} + \frac{1}{2}(x_p - z_p). \quad (11)$$

This result gives the following estimates for $m(2, p)$ for specific values for p :

p	q	(4)	(11) with p	(11) with q
1.1	11	0.782399...	0.775933...	0.775933...
1.2	6	0.809264...	0.797975...	0.797975...
1.3	4.3...	0.829043...	0.816708...	0.816708...
1.4	3.5	0.842759...	0.832320...	0.832320...
1.5	3	0.852049...	0.844625...	0.844625...
1.6	2.6...	0.858207...	0.853640...	0.853640...
1.7	2.428571...	0.862145...	0.859755...	0.859755...
1.8	2.25	0.864491...	0.863518...	0.863518...
1.9	2.1...	0.865681...	0.865460...	0.865460...
2.0	2	0.866025...	0.866025...	0.866025...

4 Sets with four elements

Consider the following restriction of the Steiner ratio:

$$m_n(2, p) = \inf \left\{ \frac{L(\text{SMT for } N)}{L(\text{MST for } N)} : N \subset \mathcal{L}_p^2 \text{ with } |N| \leq n \right\}. \quad (12)$$

Of course, then we have

$$m(2, p) = \inf \{m_n(2, p) : n > 2\}.$$

It is not hard to see that

Lemma 4.1

$$m_3(2, p) \geq \frac{3}{4}. \quad (13)$$

Proof. Let $N = \{A, B, C\}$ be a three-point set, say with $\rho(A, C)$ greater than both $\rho(A, B)$ and $\rho(B, C)$. Then

$$L_M := L(\text{MST for } N) = \rho(A, B) + \rho(B, C).$$

If the SMT has a length L_S less than L_M , it must consist of three edges from A, B and C to a common Steiner point S . Then

$$\begin{aligned} 4 \cdot L_S &= 4 \cdot (\rho(A, S) + \rho(B, S) + \rho(C, S)) \\ &= 2 \cdot (\rho(A, S) + \rho(S, B)) + 2 \cdot (\rho(B, S) + \rho(S, C)) \\ &\quad + 2 \cdot (\rho(C, S) + \rho(S, A)) \\ &\geq 2 \cdot (\rho(A, B) + \rho(B, C) + \rho(C, A)) \\ &\geq 2L_M + 2\rho(B, C) \\ &\geq 2L_M + \rho(A, B) + \rho(A, C) \\ &\geq 3L_M. \end{aligned}$$

□

On the other hand, we have seen that the Steiner ratio is less than $3/4 = 0.75$ for several planes.⁵ Hence, our upper bounds are too weak and we have to investigate sets with four elements to get sharper estimates.

Remember 2.4. The original proof uses a metric in the Banach-Mazur compactum, compare [6]. On the other hand, Albrecht [1] found this result considering the extreme points of the sets $B(1)$ and $B(\infty)$ in \mathcal{L}_p^2 . This idea suggests to consider the four given points $A = (x_p, 0)$, $B = (0, 1)$, $C = (-x_p, 0)$ and $D = (0, -1)$. Let $S_1 = (a_p, b_p)$ and $S_2 = -S_1$ be Steiner points. The tree T contains the edges $\underline{S_1A}$, $\underline{S_1B}$, $\underline{S_1S_2}$, $\underline{S_2C}$ and $\underline{S_2D}$.⁶ The parameters we determine in the following sense:

- The value x_p satisfies the condition that the triangles $\triangle ABD$ and $\triangle BCD$ are equilateral, that means

$$(1 + x_p^p)^{1/p} = 2$$

⁵For instance, for planes \mathcal{L}_p^2 where p is not far from 1 or from ∞ .

⁶Since each Steiner point has degree at least three.

and

$$x_p = (2^p - 1)^{1/p}.$$

Consequently, an MST for the points A, B, C and D has the length 6.

- Let the values a_p and b_p be nonnegative and minimizing the function

$$\begin{aligned} L(T) &= f(a, b) \\ &= 2(a^p + (1 - b)^p)^{1/p} + 2(b^p + (x_p - a)^p)^{1/p} + 2(a^p + b^p)^{1/p}. \end{aligned}$$

Clearly, such values exist. The determination of a_p and b_p needs solving a system of two nonlinear equations:

$$\begin{aligned} \frac{\partial f}{\partial a}(a_p, b_p) &= 0 \\ \frac{\partial f}{\partial b}(a_p, b_p) &= 0. \end{aligned}$$

The value $\frac{1}{6} \cdot f(a_p, b_p)$ is a new upper bound for $m(2, p)$.

We list several values for the new bound and for the bound (6) which show that the new bound is sharper than (6) if the values of the quantity p are small:

p	(6)	$\frac{1}{6} \cdot f(a_p, b_p)$
1.1	0.710027...	0.709895...
1.2	0.748308...	0.747433...
1.3	0.782306...	0.779794...
1.4	0.812675...	0.807191...
1.5	0.839947...	0.829471...
1.6	0.864559...	0.846816...
1.7	0.886874...	0.859909...
1.8	0.907193...	0.869614...

Similarly, let $A = (1, 1)$, $B = (-x_p, x_p)$, $C = (-1, -1)$ and $D = (x_p, -x_p)$ be given points, and let $S_1 = (a_p, b_p)$ and $S_2 = (-a_p, -b_p)$ be Steiner points, whereby

- We chose $x_p > 1$ such that the triangles $\triangle ABC$ and $\triangle ACD$ are equilateral, that means, the x_p is a zero of

$$f(x) = (x - 1)^p + (x + 1)^p - 2^{p+1}.$$

Hence, an MST for the points A, B, C and D has the length $6 \cdot 2^{1/p}$.

- The tree T contains the edges $\underline{S_1A}$, $\underline{S_1D}$, $\underline{S_1S_2}$, $\underline{S_2B}$ and $\underline{S_2C}$. The point $(a_p, b_p) \in [0, 1]^2$ minimizes the function

$$\begin{aligned} L(T) &= g(a, b) \\ &= 2((1-a)^p + (1-b)^p)^{1/p} \\ &\quad + 2((x-a)^p + (x-b)^p)^{1/p} + 2(a^p + b^p)^{1/p}. \end{aligned}$$

The determination of a_p and b_p needs solving a system of two nonlinear equations:

$$\begin{aligned} \frac{\partial g}{\partial a}(a_p, b_p) &= 0 \\ \frac{\partial g}{\partial b}(a_p, b_p) &= 0. \end{aligned}$$

Then the new bound is $\frac{1}{6} \cdot 2^{-1/p} \cdot g(a_p, b_p)$.

We list several values for the new bound and the bound (7) which show that the new bound is sharper than (7) if the values of the quantity p are big:

p	(7)	$\frac{1}{6} \cdot 2^{-1/p} \cdot g(a_p, b_p)$
11	0.710027...	0.709895...
6	0.748308...	0.747433...
4.3...	0.782306...	0.779796...
3.5	0.812675...	0.807210...
3	0.839947...	0.829539...
2.6...	0.864559...	0.846957...
2.25	0.907193...	0.869830...

Consequently, we get

Theorem 4.2 *The Steiner ratio of \mathcal{L}_p^2 is essentially less than $\frac{3}{4}$ if $p \leq 1.2$ and if $p \geq 6$.*

Albrecht [1] remarked that both constructions don't give an SMT, that means the bounds are upper bounds and never exact values for the Steiner ratio $m(2, p)$.

5 The Steiner ratio of dual planes

Now, we give a "plausible" conjecture following from our investigations into this subject. Namely, that for the Steiner ratio of \mathcal{L}_p -planes the following is true:

$$m(2, p) = m(2, q) \quad (14)$$

if the numbers p and q are conjugated.

This is not a new conjecture, since we find it also with Liu and Du [29]. Which facts do support 14? At first, we have that many inequalities, namely (4), (11), (6) and (7), have the same value for the parameter p and its conjugated number.⁷

Secondly, it seems that the quantity $m(2, p)$ is a concave function in the value p .

Furthermore, for the parameter 1 and ∞ the conjecture is true, since

$$m(2, 1) = m(2, \infty) = \frac{2}{3} = 0.666 \dots$$

The fourth fact which supports the conjecture is more generally. The duality of \mathcal{L}_p^2 and $\mathcal{L}_{p/(p-1)}^2$ is a special form of the duality in finite-dimensional Banach spaces. Let B the unit ball of such space, then the dual unit ball DB is defined by

$$DB = \{X : X^T Y \leq 1 \text{ for all } Y \in B\}.$$

It was conjectured that the Steiner ratio of two-dimensional Banach-spaces is equal to the Steiner ratio of its dual space, and Wan et al. [32] show that this conjecture is true for all sets with at most five points. This implies

$$m_n(2, p) = m_n \left(2, \frac{p}{p-1} \right)$$

for all numbers $n \leq 5$ and parameters p .

We think that it would be a surprise if the conjecture 14 fails.

6 Concluding remarks

There are two directions of generalizations:

⁷This is not true for the inequalities given in the section before, but we said that these bounds are not the best possible ones.

We consider two-dimensional Banach spaces, so-called Banach-Minkowski planes. These are affine planes normed by a centrally symmetric and convex body, called the unit ball. More exactly, let B be such a body then B induces a norm $\| \cdot \| = \| \cdot \|_B$ by

$$\|X\|_B = \inf\{t > 0 : X \in tB\}$$

for any point in the plane. On the other hand, let $\| \cdot \|$ be a norm, then $B = \{X : \|X\| \leq 1\}$ is a unit ball in the above sense.

Steiner's Problem in Banach-Minkowski planes is the subject of investigations during the past thirty years⁸. Most of the research dealt with the Steiner ratio of Banach-Minkowski planes. Cieslik [6] and Du et al. [11] independently conjectured that for any Banach-Minkowski plane the Steiner ratio m fulfills

$$0.66\dots = 2/3 \leq m \leq \sqrt{3}/2 = 0.86602\dots$$

The best known bounds are

Theorem 6.1 *For the Steiner ratio m of Banach-Minkowski planes the following is true:*

(a) (Gao, Du, Graham [18])

$$m \geq 2/3.$$

If there is a natural number m such that the bound $2/3$ is adopted by a set of n points, then $n = 4$, and B is a parallelogram.

(b) (Du et.al. [11])

$$m \leq \frac{\sqrt{13} - 1}{3} = 0.8685\dots$$

On the other hand, we consider higher-dimensional spaces equipped with p -norm. More exactly: Let A_d be the d -dimensional affine space. For the points $X = (x_1, \dots, x_d)$ and $Y = (y_1, \dots, y_d)$ of the space we define the distance by

$$\rho_p(X, Y) = \left(\sum_{i=1}^d |x_i - y_i|^p \right)^{1/p}$$

⁸Starting with [10]; find a survey in [7]

where $1 \leq p < \infty$ is a real number. If p runs to infinity we get the so-called Maximum distance

$$\rho_\infty(X, Y) = \max\{|x_i - y_i| : 0 \leq i \leq d\}$$

In each case we obtain a finite-dimensional Banach space written by \mathcal{L}_p^d . Similarly as in the planar case, we define the Steiner ratio by

$$m(d, p) = \inf \left\{ \frac{L(\text{SMT for } N)}{L(\text{MST for } N)} : N \subset \mathcal{L}_p^d \text{ a finite set} \right\}.$$

It is an interesting problem to determine these quantities exactly. At the moment we are only know some bounds. Using investigations about equilateral sets we have

Theorem 6.2 (Albrecht, Cieslik [1],[2])

Let $1 < p < \infty$ and $d \geq 3$. Then

$$m(d, p) \leq \frac{d+1}{d} \cdot \left(\frac{1}{2}\right)^{1/p} \cdot \min \left\{ \frac{d^{1/p}}{2}, 1 \right\}.$$

On the other hand, using isometric embeddings of Euclidean spaces in spaces \mathcal{L}_p^d , we find several specific bounds

Theorem 6.3 (Cieslik [8])

$$\begin{aligned} m(d, 4) &\leq m(3, 2) \leq 0.81119\dots \text{ for } d > 5; \\ m(d, 4) &\leq m(4, 2) \leq 0.76871\dots \text{ for } d > 10; \\ m(d, 4) &\leq m(7, 2) \leq 0.72247\dots \text{ for } d > 28; \\ m(d, 4) &\leq m(23, 2) \leq 0.69839\dots \text{ for } d > 275. \end{aligned}$$

References

- [1] J. Albrecht. *Das Steinerverhältnis endlich dimensionaler L_p -Räumen*. Master's thesis, Ernst-Moritz-Arndt Universität Greifswald, 1997.
- [2] J. Albrecht and D. Cieslik. The Steiner ratio of finite dimensional L_p -spaces. to appear in *Advances in Steiner Trees*, 1998.
- [3] D. Cheriton and R.E. Tarjan. Finding Minimum Spanning Trees. *SIAM J. Comp.* Vol.5 (1976) pp. 724-742.

- [4] F.R.K. Chung and R.L. Graham. A new bound for Euclidean Steiner Minimal Trees. *Ann. N.Y. Acad. Sci.* Vol.440 (1985) pp. 328-346.
- [5] F.R.K. Chung and F.K. Hwang. A lower bound for the Steiner Tree Problem. *SIAM J. Appl. Math.* Vol.34 (1978) pp. 27-36.
- [6] D. Cieslik. The Steiner-ratio in Banach-Minkowski planes. In R. Bodendieck, editor, *Contemporary Methods in Graph Theory*, (Bibliographisches Institut, Mannheim, 1990) pp. 231-247.
- [7] D. Cieslik. *Steiner Minimal Trees*. (Kluwer Academic Publishers, 1998).
- [8] D. Cieslik. The Steiner ratio of \mathcal{L}_{2k}^d . to appear in *Applied Discrete Mathematics*.
- [9] D. Cieslik and J. Linhart. Steiner Minimal Trees in L_p^2 . *Discrete Mathematics* Vol. 155 (1996) pp. 39-48.
- [10] E.J. Cockayne. On the Steiner Problem. *Canad. Math. Bull.* Vol. 10 (1967) pp. 431-450.
- [11] D.Z. Du, B. Gao, R.L. Graham, Z.C. Liu, and P.J. Wan. Minimum Steiner Trees in Normed Planes. *Discrete and Computational Geometry* Vol. 9 (1993) pp. 351-370.
- [12] D.Z. Du and F.K. Hwang. A new bound for the Steiner Ratio. *Trans. Am. Math. Soc.* Vol. 278 (1983) 137-148.
- [13] D.Z. Du and F.K. Hwang. An Approach for Proving Lower Bounds: Solution of Gilbert-Pollak's conjecture on Steiner ratio. *Proc. of the 31st Ann. Symp. on Foundations of Computer Science*, St. Louis, 1990.
- [14] D.Z. Du and F.K. Hwang. Reducing the Steiner Problem in a normed space. *SIAM J. Computing* Vol. 21 (1992) 1001-1007.
- [15] D.Z. Du, F.K. Hwang, and E.N. Yao. The Steiner ratio conjecture is true for five points. *J. Combin. Theory, Ser. A* Vol. 38 (1985) pp. 230-240.
- [16] D.Z. Du, E.Y. Yao, and F.K. Hwang. A Short Proof of a Result of Pollak on Steiner Minimal Trees. *J. Combin. Theory, Ser. A* Vol. 32 (1982) pp. 396-400.

- [17] P. Fermat. *Abhandlungen über Maxima und Minima*. Oswalds Klassiker der exakten Wissenschaften, Number 238, 1934.
- [18] B. Gao, D.Z. Du and R.L. Graham. A Tight Lower Bound for the Steiner Ratio in Minkowski Planes. *Discrete Mathematics* Vol. 142 (1993) 49-63.
- [19] M.R. Garey, R.L. Graham, and D.S. Johnson. The complexity of computing Steiner Minimal Trees. *SIAM J. Appl. Math.* Vol. 32 (1977) pp. 835-859.
- [20] M.R. Garey and D.S. Johnson. The rectilinear Steiner Minimal Trees is \mathcal{NP} -complete. *SIAM J. Appl. Math.* Vol. 32 (1977) pp. 826-834.
- [21] M.R. Garey and D.S. Johnson. *Computers and Intractability*. (San Francisco, 1979).
- [22] C.F. Gauß. Briefwechsel Gauß-Schuhmacher. *Werke Bd. X,1*, (Göttingen, 1917) pp. 459-468.
- [23] E.N. Gilbert and H.O. Pollak. Steiner Minimal Trees. *SIAM J. Appl. Math.* Vol. 16 (1968) pp. 1-29.
- [24] R.L. Graham and P. Hell. On the History of the Minimum Spanning Tree Problem. *Ann. Hist. Comp.* Vol. 7 (1985) pp. 43-57.
- [25] R.L. Graham and F.K. Hwang. A remark on Steiner Minimal Trees. *Bull. of the Inst. of Math. Ac. Sinica* Vol. 4 (1967) pp. 177-182.
- [26] F.K. Hwang. On Steiner Minimal Trees with rectilinear distance. *SIAM J. Appl. Math.* Vol. 30 (1976) pp. 104-114.
- [27] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. (North-Holland, 1992).
- [28] J.B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. of the Am. Math. Soc.* Vol. 7 (1956) pp. 48-50.
- [29] Z.C. Liu and D.Z. Du. On Steiner Minimal Trees with L_p Distance. *Algorithmica* Vol. 7 (1992) pp. 179–192.
- [30] H.O. Pollak. Some remarks on the Steiner Problem. *J. Combin. Theory, Ser. A* Vol. 24 (1978) pp. 278-295.

- [31] J.H. Rubinstein and D.A. Thomas. The Steiner Ratio conjecture for six points. *J. Combin. Theory, Ser. A* Vol. 58 (1991) pp. 54–77.
- [32] P.J. Wan, D.Z. Du, and R.L. Graham. The Steiner ratio of the Dual Normed Plane. *Discrete Mathematics* Vol. 171 (1997) pp. 261–275.

A Cogitative Algorithm for Solving the Equal Circles Packing Problem

Huang, Wenqi ¹

Department of Computer Science

Huazhong Univ. of Sci. and Tech. Wuhan 430074, P.R. of China

Wu, Yu-Liang ² (corresponding author)

Department of Computer Science and Engineering

Chinese Univ. of Hong Kong, Hong Kong

C. K. Wong

Department of Computer Science and Engineering

Chinese Univ. of Hong Kong, Hong Kong

Contents

1	Introduction	592
2	The Problem and the Corner-Occupying Conjecture	593
3	Basic Strategies of the Algorithm	594
3.1	Gravity Strategy	594
3.2	Edging Strategy	595
3.3	Depth Strategy	595
3.4	Queuing Strategy	595
3.5	Basic Algorithm	596

¹This work was partially supported by China National Focus Programme of Fundamental Research Development, 863 National High Technology Programme, Natural Science Foundation, Science Foundation for Doctoral Training Unit and Foundation for Computer Science Open Laboratory of Software Institute of Academica Sinica.

²This work was partially supported by Hong Kong Earmarked Research Grant 2050133 and CUHK Direct Grant 2050199.

4 Improvement on the Completeness of the Algorithms	597
4.1 The Action Value Strategy	597
4.2 Description of the New Algorithm	597
5 Experimental Results	598
5.1 A Criterion for Algorithm Completeness	598
5.2 Experimental Parameters	598
5.3 Result Analysis	598
6 Conclusions	599
References	

1 Introduction

In the past decades, quite a large number of NP-hard problems have been formulated. These problems can be found in a large number of fields including military, political, engineering, and even business administration. The classical equal circles packing problem is one of them. Unfortunately, though much research has been done in the last two decades on these problems, the results have shown that it is not likely to have any algorithm that is fast and can solve these problems exactly [1]. Consequently, devising approximation heuristics has been an important topic in solving these problems. In [2], an enumeration-based approximation methodology, called the shifting strategy, was proposed. The strategy is of great theoretical significance since it can be regarded as the best approximation algorithm possible for this kind of problems, but it is far from being practical due to its very high polynomial complexity. Therefore, it was suggested toward the end of their paper that other kinds of heuristics with lower time complexity should be sought for practical usage.

To find a heuristic algorithm which is not absolutely complete (i.e., it may not find a solution for certain problem instances, even though such a solution exists) but fast, people have tried to turn to nature for inspiration. Various strategies inspired from observations on some physical processes in nature have been proposed in heuristic designs. It has been shown that in practice, these heuristics are quite effective in solving many NP-hard problems [3, 4, 5, 6, 7, 8, 9]. In this paper, we will propose a different kind of approach, called the cogitative approach, to solve the equal circles

packing problem. The key notion in this approach lies in emulating human experience or intuition gained in solving similar problems in everyday life with certain enhancements to fit in better for solving the targeted NP-hard problems. We demonstrate the effectiveness of this approach by showing our experimental results of two cogitative algorithms for this problem. The experimental results seem to justify the practical value of this approach well.

2 The Problem and the Corner-Occupying Conjecture

The equal circles packing problem is defined as follows: Given two positive real numbers R_o and r , and an integer M , we want to know whether these M round circles, each with the same radius of r , can all be packed within a bounding circle of radius R_o without any mutual overlapping. Moreover, we would also like to know the positions of all the circles if there exists a such solution. We say that a problem instance has a solution if there exists such a non-overlapping packing for this problem instance.

Without loss of generality, we can assume that $R_o > r, M > 1$, and the bounding circle is fixed on a plane with a Descartes co-ordinate system. Clearly, the position of a circle in the plane can be determined by the co-ordinates of the circle's center.

For thousands of years, it has been believed by many that, if there exists a solution of such a packing instance then a so-called “sequential corner-occupying manner of packing” is also existent. A corner-occupying packing manner, which is a notion derived from the ancient professional masons’ working experience, is defined as follows. When a circle is placed within the bounding circle, it can be placed anywhere tangent to the bounding circle if there is no circle yet in the bounding circle; otherwise, it should be placed to a position either tangent to two certain circles already being placed or tangent to the bounding circle and some placed circle. This packing manner has been believed to be “reasonable” based on human practical experience and it is also believed that the solution, if there is one, can also be constructed by packing the circles in this way. We call such a belief as the *corner-occupying conjecture*. Following such a conjecture, without loss of generality, we can start placing the first circle in the lowest position directly, where it is tangent to the lowest point of the bounding circle. When placing the $(v+1)$ -th circle ($v = 1, 2, \dots, M-1$), we only need to find a circle position which is tangent to certain two circles (including the

bounding circle). Hence, when placing the $(v+1)$ -th circle, the candidate positions to be chosen will not exceed $2 \binom{v+1}{2}$. Thus the searching space for a solution has been reduced from the intrinsically exponential to a low polynomial in this heuristic.

We now formulate our concepts and the algorithm in more details below.

Definition 2.1 Configuration. A configuration is an ordered pair $\langle P, Q \rangle$ which represents the current state of the packing process, where P represents the set of the circles already packed into the bounding circle (we assume all the circles packed must be inside the bounding circle), and Q , the set of circles not yet packed into the bounding circle. Initially, P is empty, and the configuration at this stage is called the initial configuration; Q is empty, when a solution is obtained. It is assumed that the positions of all the circles in P are known and (implicitly) kept in the configuration.

Definition 2.2 Corner Position. Given a configuration $\langle P, Q \rangle$ ($P \neq \emptyset$ and $Q \neq \emptyset$), a corner position is defined to be a candidate packing position (within the bounding circle) for a circle in Q that can make the circle tangent to (not necessarily without over-lapping) (1) any two circles in P , or (2) any circle in P and the bounding circle. Obviously, in any configuration, any two circles in P can only construct at most two corner positions; therefore, given that there are k circles in P , the number of corner positions will not exceed $2 \binom{k+1}{2}$.

Definition 2.3. Action and Rational Action. Given a configuration $\langle P, Q \rangle$ ($P \neq \emptyset$ and $Q \neq \emptyset$), an action is defined to be the moving of one circle from Q to a corner position of the current configuration. A rational action is an action that causes no overlapping between any two circles (including the bounding circle).

3 Basic Strategies of the Algorithm

We will now show the steps of constructing our algorithm based on the ideas obtained from some past human social experience (or intuition).

3.1 Gravity Strategy

Definition 3.1 Given two points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ on the plane, we say the priority of P_1 is greater than that of P_2 , if any of the following two

conditions holds:

- (i) $y_1 < y_2$;
- (ii) $y_1 = y_2$ and $x_1 < x_2$.

A gravity strategy will implicitly assign a higher priority to corner positions near the starting point of the packing process.

3.2 Edging Strategy

Definition 3.2. The degree of edging is defined to be the distance between the centers of a circle and the bounding circle.

An edging strategy can be used to pack a circle near the boundary of the bounding circle.

3.3 Depth Strategy

Intuitively speaking, given a configuration $\langle P, Q \rangle$ ($P \neq \emptyset$ and $Q \neq \emptyset$) and two circles m_1 and m_2 in P that forms some corner positions (note that m_1 and m_2 need not be tangent to each other), the depth of the corner positions can be regarded as the distance between its contributors m_1 and m_2 . When we place a circle q on a corner position constructed by circles m_1 and m_2 , circle q would look like being “inserted into” the “gap” formed by m_1 and m_2 “deeper”, if the distance between m_1 and m_2 is larger. Therefore, we can measure the “depth” of such a “gap” by the distance between m_1 and m_2 .

Definition 3.3. In configuration $\langle P, Q \rangle$ ($P \neq \emptyset$ and $Q \neq \emptyset$), we define the depth of a corner position as (1) the distance between the two circles, if the corner position is formed by two circles in P , or (2) the distance between a circle and the boundary of the bounding circle, if the corner position is formed by a circle in P and the bounding circle.

The depth strategy can assign a higher priority to a deeper “gap” and enforce the circles to be packed as “deepest” as possible.

3.4 Queuing Strategy

Definition 3.4. Given that q is the circle packed during a rational action, the distance, d_c , between the center of the circle q and that of the circle of last packing action is called the degree of packing discontinuity of this rational action.

The queuing strategy can be used to construct a packing sequence with circles packed consecutively in order being placed close to each other geographically, which can help construct an our so-called corner-occupying packing sequence.

3.5 Basic Algorithm

Based on the above strategies, we proposed the following two basic algorithms: algorithm I $\langle \text{Dep}, D, Y, X \rangle$ and algorithm II $\langle D_c, D, Y, X \rangle$.

In a configuration $\langle P, Q \rangle$, we can obtain two kinds of quaternary ordered groups (for the two algorithms), $\langle \text{dep}, d, y, x \rangle$ and $\langle d_c, d, y, x \rangle$, for each candidate rational action, where dep represents the depth of the related corner position, d and d_c represent the degree of the edging and the degree of packing discontinuity of the packed circle, and x and y are the coordinates of the corner position of the action.

Definition 3.5. (for Algorithm I) Given a configuration $\langle P, Q \rangle$ ($P \neq \emptyset$ and $Q \neq \emptyset$), two candidate rational actions and their corresponding quaternary ordered groups, $\alpha_1 = \langle \text{dep}^{(1)}, d^{(1)}, y^{(1)}, x^{(1)} \rangle$ and $\alpha_2 = \langle \text{dep}^{(2)}, d^{(2)}, y^{(2)}, x^{(2)} \rangle$, we say that α_1 is prior to α_2 , if any of the following conditions is satisfied:

- (i) $\text{dep}^{(1)} > \text{dep}^{(2)}$;
- (ii) $\text{dep}^{(1)} = \text{dep}^{(2)}$ and $d^{(1)} > d^{(2)}$;
- (iii) $\text{dep}^{(1)} = \text{dep}^{(2)}$ and $d^{(1)} = d^{(2)}$ and point $(x^{(1)}, y^{(1)})$ is prior to point $(x^{(2)}, y^{(2)})$.

Definition 3.6. (for Algorithm II) Given a configuration $\langle P, Q \rangle$ ($P \neq \emptyset$ and $Q \neq \emptyset$), two candidate rational actions and their corresponding quaternary ordered groups, $\beta_1 = \langle d_c^{(1)}, d^{(1)}, y^{(1)}, x^{(1)} \rangle$ and $\beta_2 = \langle d_c^{(2)}, d^{(2)}, y^{(2)}, x^{(2)} \rangle$, we say that β_1 is prior to β_2 , if any of the following conditions is satisfied:

- (i) $d_c^{(1)} < d_c^{(2)}$;
- (ii) $d_c^{(1)} = d_c^{(2)}$ and $d^{(1)} > d^{(2)}$;
- (iii) $d_c^{(1)} = d_c^{(2)}$ and $d^{(1)} = d^{(2)}$ and point $(x^{(1)}, y^{(1)})$ prior to point $(x^{(2)}, y^{(2)})$.

Now we describe algorithm I $\langle \text{Dep}, D, Y, X \rangle$ as follows:

Step 1. In the initial configuration $\langle P_0, Q_0 \rangle$ ($P = \emptyset$), we can simply place a circle at the bottom of the bounding circle.

Step K+1. In configuration $\langle P_k, Q_k \rangle$ ($k=1, 2, \dots$), where there are k circles in P_k , we will take one of the following.

- (i) If $k = M$, then the solution is found. Report the solution.
- (ii) If $k < M$, and there does not exist any candidate rational action, then just terminate and give a failure report.
- (iii) If $k < M$, and there exist some candidate rational actions to be selected, then simply perform the one whose quaternary ordered group $\langle \text{dep}, d, y, x \rangle$ has the highest degree of priority. A new configuration $\langle P_{k+1}, Q_{k+1} \rangle$ is then obtained. The step is repeated until a halting condition is met.

The description of algorithm II $\langle D_c, D, Y, X \rangle$ is similar.

4 Improvement on the Completeness of the Algorithms

4.1 The Action Value Strategy

In the above, we have constructed several strategies employing the cogitative approach and designed two basic algorithms. The two algorithms though run quite fast because of their simplicity, they are not exempt from the common drawback of all other kinds of heuristics: the lacking of completeness. That is, sometimes the two algorithms may not be able to find a solution for certain problem instances, even though such a solution exists. In order to increase the completeness of the algorithms, the following strategy can be adopted.

Definition 4.1. For a given algorithm (I or II) and a certain configuration, we define the *packing value* of a rational action according to such algorithm as the number of circles that can be further packed (under the same algorithm) given that this rational action is taken.

We use such a packing value to help in estimating the “quality” of selecting a particular action.

4.2 Description of the New Algorithm

From the above strategy, we can obtain two *valued algorithms* based on the two basic algorithms, denoted by algorithm I $\langle V, \text{Dep}, D, Y, X \rangle$ and algorithm II $\langle V, D_c, D, Y, X \rangle$, respectively.

The algorithm $\langle V, \text{Dep}, D, Y, X \rangle$ is described as follows:

Step 1. In the initial configuration $\langle P_0, Q_0 \rangle$ ($P = \emptyset$), we can simply place a circle at the bottom of the bounding circle.

Step K+1. In configuration $\langle P_k, Q_k \rangle$ ($k=1, 2, \dots$), where there are k circles in P_k , we will take one of the following.

- (i) If $k = M$, then the solution is found. Report the solution.
- (ii) If $k < M$, and there does not exist any candidate rational action, then just terminate and give a failure report.
- (iii) If $k < M$, and there are some candidate rational actions to be selected, then we will select among them the one with the highest packing value according to the basic algorithm I $\langle \text{dep}, d, y, x \rangle$. A new configuration $\langle P_{k+1}, Q_{k+1} \rangle$ is then obtained. The step is repeated until a halting condition is met.

The description of algorithm II $\langle V, D_c, D, Y, X \rangle$ is similar.

5 Experimental Results

5.1 A Criterion for Algorithm Completeness

In this study, we adopt the following criterion in comparing the degrees of the completeness of two different algorithms. Given M circles of the same size, we shall find the minimum radius R_o of the bounding circle that each algorithm can achieve to yield a successful packing. The algorithm that yields the smaller R_o has a higher degree of completeness.

5.2 Experimental Parameters

On a PC 586/90 computer, we have run 41 examples for 4 algorithms: the two basic algorithms and the two action value algorithms. The results are shown in Table 1. In this experiment, r is set to be 20 in all runs. R_o represents the minimum bounding circle radius (up to the accuracy of 0.01) needed for each related algorithm and given M value. T refers to the CPU time spent in the experiment.

5.3 Result Analysis

Our experimental results have shown that each of the four proposed algorithms has a full completeness on the cases with M values of up to 7. I.e., either algorithm can find an optimum solution on all these cases. We show the completed results of algorithm I $\langle \text{Dep}, D, Y, X \rangle$ and algorithm I $\langle V, D, Y, X \rangle$ on $M = 6$ in Fig. 1(e) and that of algorithm II $\langle D_c, D, Y, X \rangle$ and algorithm II $\langle V, D_c, D, Y, X \rangle$ in Fig. 1(d). On M values of 3, 4,

5, and 7, all of the 4 algorithms completed with identical solutions, which are shown in Fig 1(a, b, c, f).

On the 41 examples shown in Table 1, although we have no way of knowing whether the results shown are optimum or not, we do believe that most of them should be very close to the optimum solutions already. On Figs. 2 to 9, we show some small completed examples; and on Fig. 10, we show a larger example where 96 circles were packed in less than 5 mins. From a coarse eye-examining, all of the results seem to be quite ideal, if not optimum. To justify the completeness of the algorithms, we have made some manual efforts by asking some graduate students to pack coins manually and compared their results with the algorithms'. We have not been able to find an example whose solution manually done is better than that produced by the programs (neglecting the potential measuring and round-off errors). In this study, we have also implemented many other variations on the algorithms, but none of them have shown to be better than the 4 presented ones, in terms of either speed or result quality.

Although the completeness of the two basic algorithms is a bit lower than that of their two action value versions, they can run very fast. E.g., most results can be completed within one second even on an M value of up to 200. On the other hand, the two action value algorithms run slower but are better in completeness. In general, on the two action value algorithms, the algorithm II $\langle V, D_c, D, Y, X \rangle$ is slightly better than the algorithm I $\langle V, Dep, D, Y, X \rangle$ in most cases. As a whole, we believe these four algorithms should complement each other quite well and should be very powerful in serving most problem instances of the investigated problem in terms of practical applications.

6 Conclusions

In this paper, we have introduced a different kind of approach, namely, the cogitative approach, in heuristic designs for solving NP-hard problems. Compared to the conventional nature-simulating based approaches, the proposed one emphasizes observations on the way human solve similar hard problems and incorporating the intuitions gained during many years of experience.

Two very fast and effective cogitative algorithms for solving the hard equal circles packing problem have been proposed and experimentally justified. The experimental results are very encouraging.

The problem of packing equal circles is at least as difficult as many other known NP-hard problems. But we have shown that the proposed cogitative algorithms attack the problem quite effectively and very good results can be obtained easily, which seems to be a promising indication on further potential in developing heuristics along this direction. We believe similar techniques should be applicable in developing efficient approximation algorithms for solving unequal circles packing and unequal polygons packing problems, which are of greater practical value. We are currently working along this line of investigation.

M	<Dep, D, Y, X>		<D _c , D, Y, X>		<V, Dep, D, Y, X>		<V, D _c , D, Y, X>	
	R _o (cm)	T(sec)	R _o (cm)	T(sec)	R _o (cm)	T(sec)	R _o (cm)	T(sec)
10	76.27	0.00	76.27	0.00	76.27	0.05	76.27	0.05
11	79.14	0.00	78.48	0.00	78.48	0.05	78.48	0.05
12	81.30	0.00	81.30	0.05	80.79	0.05	80.79	0.11
13	84.73	0.05	84.73	0.00	84.73	0.11	84.73	0.16
14	86.57	0.00	86.57	0.00	86.57	0.22	86.57	0.16
15	90.55	0.05	90.55	0.00	90.55	0.22	90.55	0.27
16	92.31	0.05	92.31	0.05	92.31	0.27	92.31	0.33
17	96.82	0.00	96.74	0.05	95.85	0.22	96.36	0.33
18	97.76	0.00	97.28	0.00	97.28	0.49	97.28	0.60
19	100.28	0.00	97.28	0.00	97.28	0.60	97.28	0.66
20	102.45	0.00	102.45	0.00	102.45	0.66	102.45	0.71
21	105.21	0.00	105.21	0.00	105.05	0.71	105.05	0.88
22	108.99	0.00	108.80	0.00	108.99	1.04	108.80	1.04
23	111.01	0.00	111.01	0.00	111.01	1.32	111.01	1.32
24	113.84	0.00	113.45	0.00	113.37	1.37	113.05	1.04
25	116.20	0.05	115.22	0.05	115.22	1.70	115.22	1.76
26	117.10	0.00	117.09	0.00	117.05	2.03	117.09	2.03
27	119.75	0.00	118.86	0.00	118.82	2.25	118.18	1.87
28	121.94	0.00	120.93	0.05	120.69	2.47	120.69	2.47
29	123.28	0.05	123.27	0.05	123.26	3.02	122.99	2.58
30	125.46	0.00	124.41	0.05	124.03	3.24	124.03	3.19
31	128.09	0.05	127.13	0.05	127.23	3.90	127.10	3.68
32	129.28	0.05	129.14	0.00	129.11	3.19	129.10	4.34
33	132.54	0.00	130.27	0.00	129.79	4.73	129.78	4.67
34	134.14	0.00	133.26	0.00	133.14	5.22	133.23	4.23
35	135.11	0.00	135.14	0.05	134.14	5.82	134.03	5.93
36	136.39	0.05	135.18	0.05	135.19	6.76	135.18	6.92
37	138.27	0.05	135.18	0.00	135.49	6.26	135.18	7.20
38	140.74	0.00	139.40	0.05	139.40	8.30	139.27	7.91
39	141.58	0.05	141.52	0.00	141.37	9.07	141.37	9.07
40	144.12	0.00	142.90	0.05	142.78	10.00	142.78	9.62
41	146.57	0.00	145.58	0.05	145.52	11.37	145.55	11.26
42	147.85	0.05	147.43	0.05	147.31	12.25	147.43	12.31
43	149.98	0.05	149.06	0.00	148.92	13.52	148.97	13.30
44	151.74	0.00	151.39	0.05	150.92	14.67	150.52	14.23
45	153.41	0.05	152.92	0.05	152.42	13.85	152.30	12.80
46	154.60	0.05	153.86	0.05	153.86	17.47	153.86	17.36
47	155.61	0.05	155.30	0.05	155.11	18.79	154.92	18.19
48	157.26	0.05	157.24	0.05	156.75	19.89	156.06	19.12
49	159.28	0.05	158.45	0.05	158.24	21.15	158.20	21.65
50	160.31	0.05	159.75	0.05	159.76	23.57	159.32	22.64

Table 1

References

- [1] M.R. Garey and D.S. Johnson, Computer and Intractability: A Guide to the Theory of NP-Completeness, *Freeman*, San Francisco, 1978.
- [2] Dorit S. Hochbaum and Wolfgang Maass, Approximation Schemes for Covering and Packing Problems in Image Processing and VLSI, *Journal of the Association for Computing Machinery*, 32(1), pp. 130-136, 1985.
- [3] Huang Wenqi and Zhan Shouhao, A Quasi-physical Method for Solving Packing Problems, *Acta Mathematicae Applicatae Sinica*, Vol. 2, No. 2, pp. 176-180, 1979.
- [4] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, Optimization by Simulated Annealing, *Science*, Vol. 220, pp. 671-680, 1983.
- [5] Huang Wenqi, A Quasi-physical Method for Solving the Covering Problem – an Approach to Tackling NP-hard Problems, *Chinese Journal of Computers*, Vol. 12, No. 8, pp. 610-616, 1989.
- [6] Li Wei and Huang Wenqi, A Mathematic-physical Approach to the Satisfiability Problem, *Science in China (Series A)*, Vol. 38, No. 1, pp. 116-128, 1995.
- [7] Huang Wenqi, Zhu Hong, Xu Xiangyang, and Song Yimin, A Heuristic Algorithm for Solving Square Packing Problem, *Chinese Journal of Computers*, Vol. 16, No. 11, pp. 829-836, 1993.
- [8] Huang Wenqi, Li Qinghua, and Yu Xiangdong, A Quasi-physical Method for Solving the Three Dimensional Packing Problem, *Acta Mathematicae Applicatae Sinica*, Vol. 9, No. 4, pp. 443-453, 1986.
- [9] R.J. Fowler, M. S. Paterson, and S. L. Tanimoto, Optimal Packing and Covering in the Plane are NP-complete, *Inform. Process. Lett.*, Vol. 12, pp. 133-137, 1981.

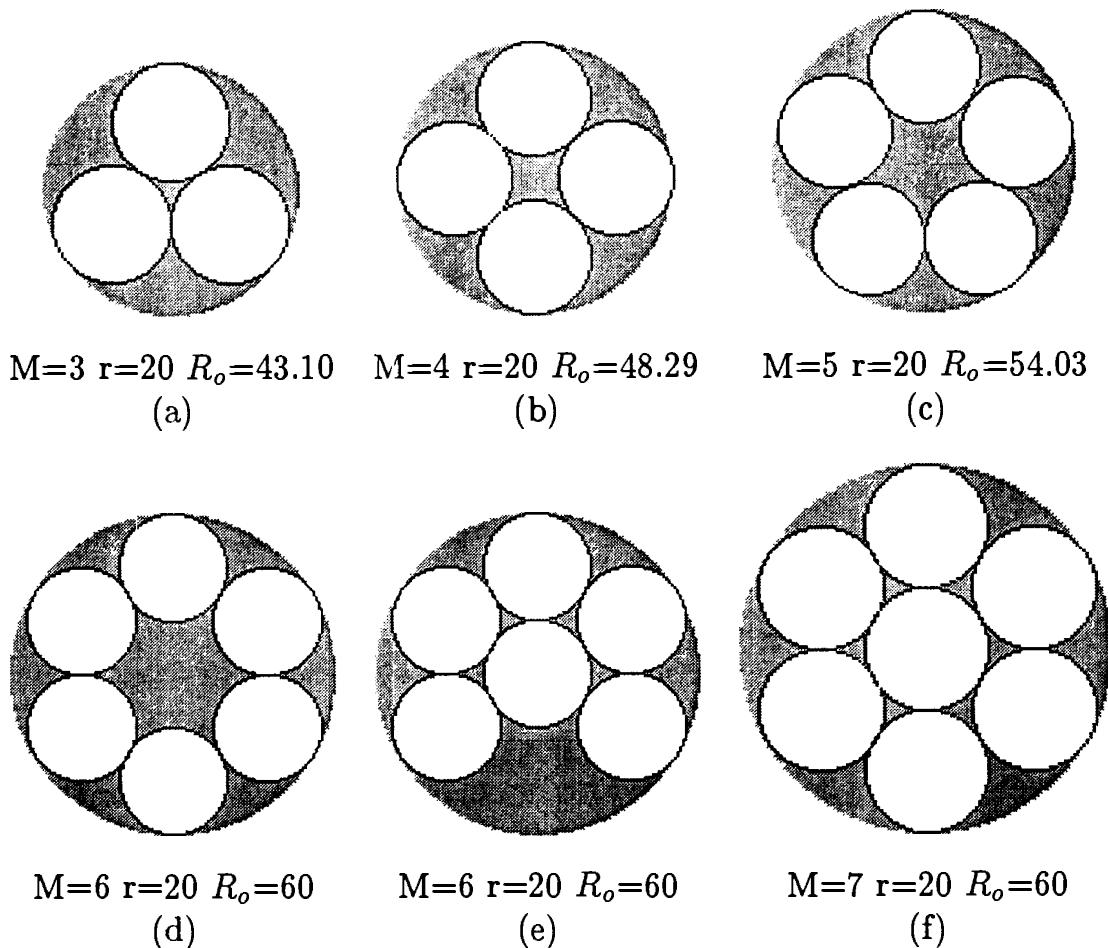
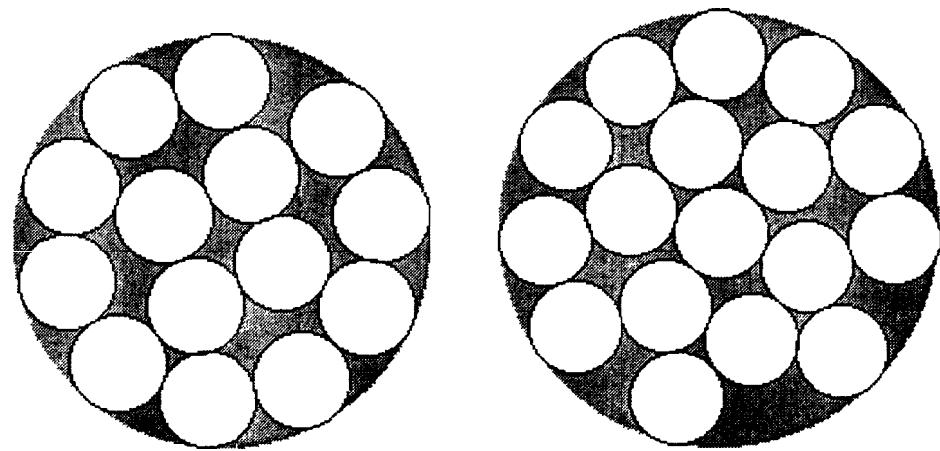


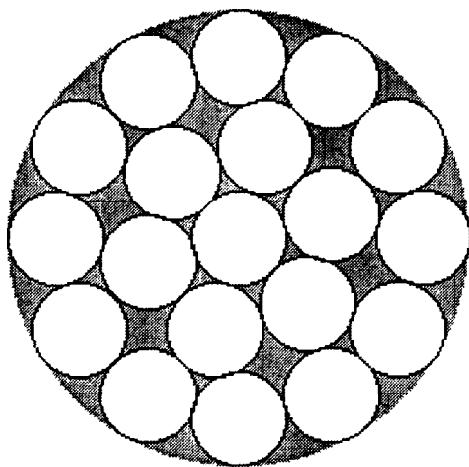
Fig. 1



$M=14$ $r=20$ $R_o=86.57$ $t=0.16$ $M=17$ $r=20$ $R_o=90.85$ $t=0.22$
 $\langle V, D_c, D, Y, X \rangle$ $\langle V, Dep, D, Y, X \rangle$

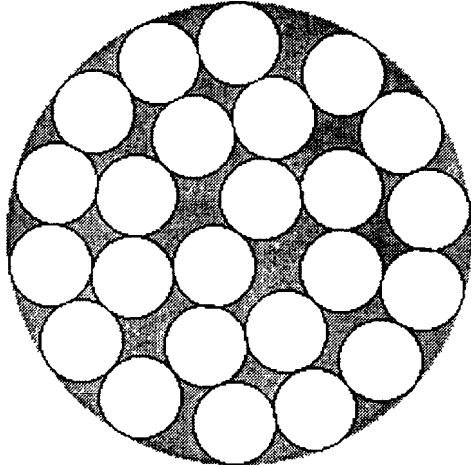
Fig. 2

Fig. 3



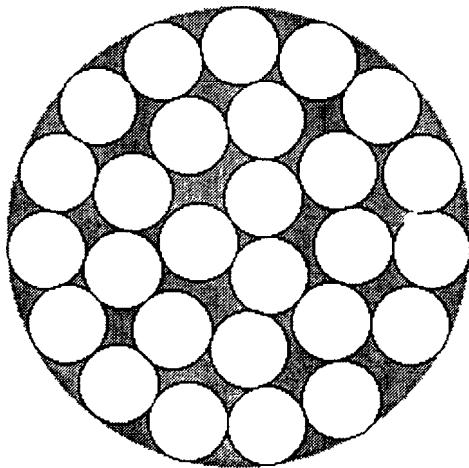
M=19 r=20 $R_o=97.28$ t=0.66
<V,D_c,D,Y,X>

Fig. 4



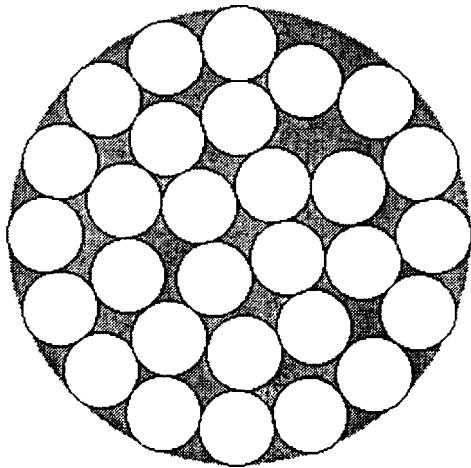
M=24 r=20 $R_o=113.05$ t=1.04
<V,D_c,D,Y,X>

Fig. 5



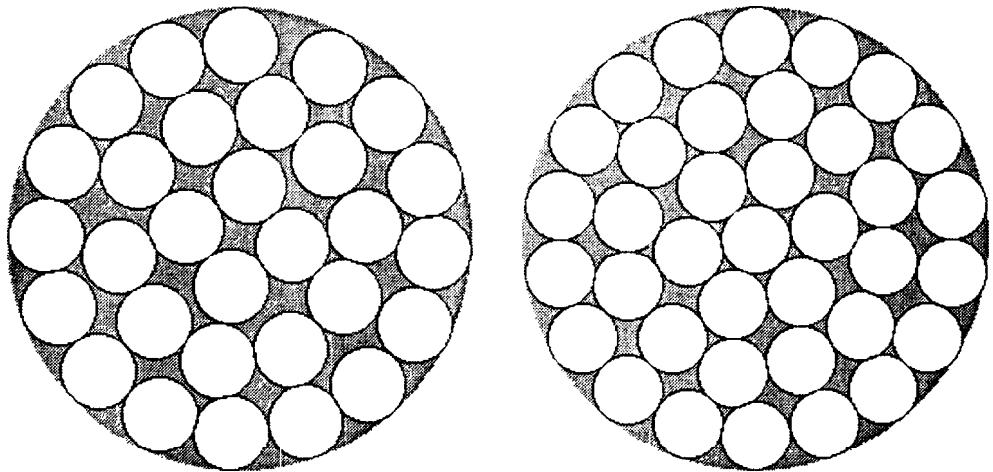
M=27 r=20 $R_o=118.18$ t=1.87
<V,D_c,D,Y,X>

Fig. 6



M=29 r=20 $R_o=122.99$ t=2.58
<V,D_c,D,Y,X>

Fig. 7

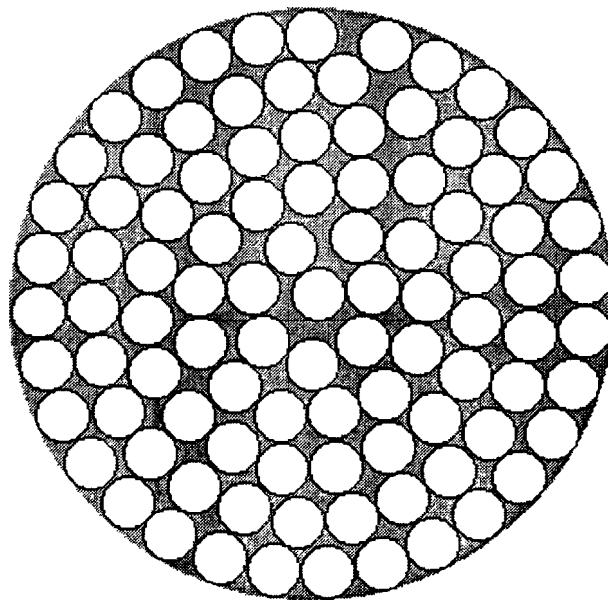


M=30 r=20 $R_o=124.03$ t=3.19 M=37 r=20 $R_o=135.18$ t=7.20
<V,D_c,D,Y,X>

Fig. 8

<V,D_c,D,Y,X>

Fig. 9



M=100 (4 left unpacked) r=20 $R_o=220.00$ t=296.26

Fig. 10

Author Index

- Aardal, K.I., 341, 369
Aarts, E., 29, 31, 32, 47-48, 358, 370
Aarts, E.H.L., 31, 65
Abbattista, F., 40, 48
Abe, S., 282, 286
Abel, L., 524, 529
Abello, J., 41, 48
Abramson, B., 384, 423, 529
Achatz, H., 104, 134
Ackley, D.H., 274-275, 286
Adams, M.B., 275, 290
Adleman, L.M., 40, 48
Adleman, Leonard M., 529
Adler, M., 198-199
Agerwala, T., 524, 529
Agostino, S. de, 250
Agrawal, V., 391, 524, 534
Agrawal, V.D., 235, 251
Agule, B.J., 524, 547
Aho, A.V., 392, 524, 527, 529
Ahrens, W., 384, 423, 529
Ahuja, R.K., 87, 92, 95, 98, 135, 145
Akers, S.B., 479, 524, 529
Akgül, M., 88, 98, 100-101, 103-104, 135
Akkoyunlu, E.A., 19-20, 48
Albers, S., 191-192, 199
Albrecht, J., 581, 583, 586
Albrecht, Jens, 573
Alizadeh, F., 41, 48
Aloimonos, J., 523, 529
Alon, N., 18, 30, 48
Alt, H., 78, 115, 135
Ambler, A.P., 13, 46, 48
Amin, A.T., 16, 48
Andersen, S.K., 246, 256
Anderson, J.A., 523, 529, 548
Anderson, J.R., 262, 274-276, 291
Anderson, R.J., 175, 199
Andre, P., 391, 415, 540
Annexstein, F., 472, 565
Ansari, N., 274, 285-286
Argos, P., 41, 73
Armstrong, R.D., 114, 135
Arora, S., 13-14, 48-49, 530
Arrow, K.J., 530
Aschemann, G., 531
Ashar, P., 390-391, 397, 530
Aspvall, B., 392, 475, 530
Assmann, S.F., 195-196, 199
Auguston, J.G., 19-20, 49
Ausiello, G., 12, 49
Auton, L., 397, 525, 537
Auton, L.D., 414, 537
Avis, D., 112, 135
Aviyoshi, H., 20-21, 73
Avondo-Boden, G., 41, 49
Aykanat, C., 284, 286
Baas, S.M., 27, 49
Babai, L., 30, 48
Babel, L., 23-24, 27, 49
Bäck, T., 34, 49
Bafna, V., 225-228, 250
Bagchi, A., 524, 530
Baird, H., 524, 530
Baker, B.S., 159, 171-172, 199
Balas, E., 14-15, 21-22, 24-27, 34, 40, 49-50, 88, 105-106, 128-129, 131, 135-136, 147

- Balinski, M.L., 77, 99-102, 114, 136
 Ballard, D.H., 32, 45, 50, 523, 530
 Baltzer, 88, 92, 140
 Bammel, S.E., 132, 136
 Bandelt, H.-J., 124, 136
 Banerjee, P., 530
 Banerji, D.K., 524, 537
 Bar-Yehuda, R., 213, 221-225, 240,
 246, 250
 Barahona, F., 41, 50
 Barbour, A.E., 254, 530
 Barr, R.S., 96-97, 105, 108, 136-
 137
 Barrow, H.G., 13, 46, 48, 50
 Bartal, Y., 191, 199
 Basu, D., 524, 551
 Battiti, R., 37, 50
 Baybars, I., 300, 334, 370
 Beall, C.L., 299-300, 344, 367, 377
 Beame, P., 504, 530
 Beck, L.L., 344, 375
 Becker, A., 225-226, 250
 Bednarek, A.R., 19, 51
 Beineke, L.W., 51
 Békési, J., 175, 199
 Bellare, M., 14, 51
 Bellifemmine, F., 40, 48
 Ben-Davis, S., 525, 530
 Ben-Tal, A., 531
 Bender, E.A., 343, 370
 Bengtsson, M., 284, 286
 Bennetts, R.G., 524, 531
 Bentham, H. van, 368-370
 Bentham, H.P., 368, 370
 Berge, C., 14, 51
 Beringer, A., 531
 Berman, K., 478, 531
 Berman, P., 13, 44-45, 51, 225-
 228, 250
 Bernhardsson, B., 384, 423, 531
 Berry, L.A., 300, 370
 Berstein, P.A., 523, 531
 Berthod, M., 524, 567
 Bertoni, A., 33, 51
 Bertsekas, D.P., 88, 93-95, 105-107,
 137, 299, 370
 Bhandari, I.S., 524, 553
 Bhattacharya, B.K., 15, 51-52
 Bibel, Wolfgang, 409, 531
 Bilbro, G., 262, 274-276, 286
 Birkhoff, G., 77, 137
 Biswas, N.N., 524, 531, 563
 Bitner, J.R., 390, 410-411, 413, 531
 Blair, C.E., 391-393, 445, 531
 Blanks, J.P., 524, 531
 Blazewicz, J., 182, 199, 524, 531
 Blough, D.M., 44, 52
 Blum, N., 78, 115, 135
 Böhm, M., 391, 394-396, 412, 531
 Bokhari, S.H., 524, 532
 Bolles, R.C., 47, 52
 Bollobás, B., 8, 18, 30, 52, 323,
 327, 370
 Bomze, I.M., 1, 9, 11, 24-25, 27,
 35, 39, 52-53
 Bomze, Immanuel M., 1
 Bondy, J.A., 231, 250
 Bonias, I., 321-323, 370
 Bonner, R.E., 19, 53
 Bonomi, F., 524, 532
 Bonvanie, M.C., 27, 49
 Boppana, R., 14, 31, 53
 Boros, E., 414, 474, 493, 532
 Boufkhad, Y., 391, 415, 491, 540
 Bouju, A., 361, 371
 Boulala, M., 15, 53
 Bourjolly, J.-M., 24, 53
 Bourret, P., 284, 287

- Bovet, D.P., 12, 53, 250
Bovier, A., 266, 286
Box, F., 301, 336, 346, 371
Boyce, J.F., 361, 371
Brady, H.N., 524, 532
Brady, M., 527, 532
Bramel, J., 156, 200
Brandstädt, 216, 250
Brayton, R., 480, 519-522, 554, 557
Brayton, R.K., 414, 524, 564-565
Brelaz, D., 24, 53, 342, 346, 371
Breuer, M.A., 244, 251, 524, 532
Brewer, F., 524, 532
Brockington, M., 28, 53
Broder, A.Z., 500-501, 525, 532
Brogan, W.L., 87, 137
Bron, C., 19-21, 53
Brooks, M.J., 523, 532, 549
Brouwer, A.E., 42, 54
Brown, C.A., 390, 392, 400-402,
 410, 413, 473, 532-533, 544,
 562
Brown, C.M., 13, 46, 48, 523, 530
Brown, D.J., 159, 167-170, 199-
 200, 206
Brown, M., 45, 50
Bruderlin, B., 524, 533
Bruynooghe, M., 415, 523, 533, 563
Bryant, R.E., 390-391, 396-397, 479-
 480, 480, 533
Buchler, N.E.G., 283, 286
Budinich, M., 1, 16, 18, 39, 52, 54
Budinich, Marco, 1
Budinich, P., 16, 54
Bugrara, K., 390, 397, 410, 533
Bugrara, K.M., 401-402, 533-534
Buhmann, J.M., 284, 288
Bui, T.N., 35, 54
Bultan, T., 284, 286
Bundy, A., 523, 534
Büning, H.K., 399, 411, 414, 508,
 534
Büning, Kleine H., 408, 476, 533
Burkard, R.E., 85, 87, 114, 116-
 118, 120-122, 129-132, 138,
 141, 185-186, 200
Burkard, Rainer E., 75
Burlet, M., 15, 54
Burns, J.E., 15, 54
Burns, J.L., 524, 534
Buro, M., 399, 411, 414, 508, 534
Burstall, R.M., 13, 46, 48, 50
Burstein, M., 524, 534
Bushnell, M., 391, 524, 534
Butler, G., 413, 534
Cabon, B., 284, 287
Cai, M., 233-234, 250
Cain, R.A., 47, 52
Camerini, P., 133, 138
Cameron, S.H., 343, 371
Campadelli, P., 33, 51
Campbell, M., 523, 557
Carlier, J., 391, 400, 415, 451, 540
Carmassi, F., 354, 371
Carpaneto, G., 88, 92, 114, 122,
 139
Carraghan, R., 26, 41, 54
Carraresi, P., 86, 92, 116, 138-139
Carson, D., 408, 571
Carter, B., 34-35, 54, 69
Castañon, D.A., 88, 105-107, 137,
 139
Castelino, D., 362, 371
Castelino, D.J., 361, 364, 371
Catthoor, F., 569
Cechlárová, K., 80, 139
Cederbaum, I., 255

- Cedergren, R., 557
 Cela, Eranda, 75
 Ceria, S., 27, 34, 49
 Černy, V., 417, 534
 Chakradhar, S., 251, 391, 524, 534
 Chakravarty, I., 523, 534
 Chamberlain, R.D., 571
 Chan, L.M.A., 156, 200
 Chan, P.K., 524, 534
 Chandra, A.K., 170, 183, 197, 200,
 523, 535
 Chandrasekaran, B., 523, 552
 Chandrasekaran, R., 471, 535
 Chandru, V., 471, 535
 Chang, G.J., 15, 54
 Chang, M.-S., 15, 54
 Chang, R.C., 14-15, 54-55
 Chang, S.-J., 15, 74
 Chang, Y., 390, 535
 Chang, Y.-J., 386, 388, 455, 535,
 570
 Chao, M.T., 400-401, 535
 Charney, H.R., 524, 535
 Charon, I., 248, 251
 Chattopadhyay, A., 524, 537
 Chellappa, R., 523, 542
 Chen, B., 191-192, 200
 Chen, L., 15, 33, 72
 Chen, M., 524, 555
 Chen, M.-S., 524, 566
 Chen, R., 247, 251
 Chen, W.T., 394-395, 535, 541
 Chen, Y.-H., 15, 54
 Cheng, K.T., 235, 251
 Cheng, S., 524, 577
 Cheriton, D., 575, 586
 Cherkassky, B.V., 92, 95, 139
 Chiba, N., 14-15, 20-21, 41, 55
 Chiba, T., 524, 568
 Chien, R.T., 44, 70
 Chin, R.T., 523, 535
 Cho, Y.E., 524, 548
 Choi, M.Y., 267, 292
 Chor, B., 525, 530
 Choukhmane, E., 14-15, 55
 Chowdhury, D., 267, 287
 Chrobak, M., 15, 41, 55
 Chu, L.-C., 570
 Chu, Tam-Anh, 535
 Chu, Y., 524, 536
 Chudak, F.A., 226, 228, 251
 Chung, F.R.K., 576, 587
 Chvátal, V., 14-15, 41, 49, 51, 55,
 251, 314, 371, 390, 407,
 415, 491, 499, 504, 535-
 536
 Cichocki, A., 275-278, 281, 287,
 536
 Cieslik, D., 575, 577-578, 581, 585-
 587
 Cieslik, Dietmar, 573
 Cleary, J., 523, 536
 Clocksin, W.F., 536
 Clowes, M.B., 391, 536
 Cockayne, E.J., 585, 587
 Coddington, P., 285, 287
 Coffman, E.G., 159, 171, 199
 Coffman, Jr, E.G., 154, 156, 176,
 183, 187-188, 192, 194, 196,
 200-201
 Coffman, Jr., Edward G., 151
 Cohen, B., 430-431, 441, 457, 460,
 491, 510, 512, 516-517, 565
 Cohen, J., 523, 536
 Cohoon, J.P., 524, 536
 Colmerauer, A., 523, 536
 Conery, J.S., 395, 536
 Conforti, M., 471-472, 536

- Constantinides, A.G., 572
Cook, S.A., 382, 392, 399, 525, 536
Cooper, P.R., 395, 536
Coorg, S.R., 217, 252
Coppersmith, D., 79, 139
Cormen, T.H., 537
Corneil, D.G., 19, 67, 216, 252,
 523, 537, 563
Corneujols, G., 27, 34, 49
Cornuéjols, G., 471-472, 536
Corradi, K., 55
Couch, L.W., 305, 371
Courant, R., 269, 287
Cozzens, M.B., 315-318, 320, 324-
 325, 337, 347, 349-351, 353,
 371-372
Crama, Y., 124, 131, 136, 139, 474,
 477-478, 493, 532, 537
Crawford, J., 397, 525, 537
Crawford, J.M., 414, 437, 537
Crescenzi, P., 12-14, 49, 53, 55
Croce, F.Della, 23-24, 55
Crocker, A., 523-524, 539
Crompton, W., 364, 372
Cronin, D.H., 300, 370
Csirik, J., 154, 163-165, 172, 182,
 185, 195, 201
Culberson, J.C., 28, 53
Cun, B.Le, 41, 56
Cung, V.-D., 41, 56
Cunningham, W.H., 96-98, 102, 139-
 140
Cuyvers, L., 2, 20, 66
Cvetanovic, Z., 524, 537
Cvetković, D.M., 16, 56
- Dadson, C.E., 301, 372
Dahlhaus, E., 41, 56
Dai, W., 524, 537
Dalal, M., 476, 537
Dalbis, D., 40, 48
Danninger, G., 24, 52
Dantzig, G.B., 96, 140
Das, S.R., 524, 537
Dasgupta, S., 32, 71-72
Davio, M., 568
Davis, L.S., 523, 537-538, 548
Davis, M., 390-391, 396-397, 406,
 506, 538
Dawson, W., 524, 548
De Man, H., 483, 520, 524, 569
De Simone, C., 15, 56
Deineko, V.G., 85, 140
Dechter, A., 523-524, 538
Dechter, R., 245, 252, 414-415, 508,
 523-524, 538
Degot, V., 41, 56
Delambre, L.M.L., 523, 569
Demers, A., 153, 159, 161, 171-
 172, 204
Deng, X., 233-234, 250
Denker, J.S., 523, 538
Deo, N., 41, 56
Derigs, U., 85, 114, 122, 138, 140
Dershowitz, N., 390, 539
Desai, N., 26, 69
Desler, J.F., 21, 56
Deutsch, D.N., 524, 539
Deutsch, J.P.A., 523, 539
Devadas, S., 390-391, 397, 524, 530,
 539
Devroye, L., 112, 135
Dhall, S.K., 524, 539
Dhar, V., 523-524, 539
Dijkstra, E.W., 140
Dimitropoulis, C.H.D., 361, 371
Dincbas, M., 523, 539
Dirac, G.A., 15, 56

- Director, S.W., 524, 561
 Doenhardt, J., 524, 539
 Doignon, J.P., 314, 372
 Donald, J., 252
 Donath, W.E., 110, 140, 524, 539
 Doob, M., 16, 56
 Doreian, P., 19, 56
 Dowaji, S., 41, 56
 Dowling, W.F., 469, 540
 Downey, R.G., 252
 Dowsland, K.A., 262, 287
 Doyle, J., 415, 540
 Dror, M., 198, 201
 Du, B., 524, 540
 Du, D.-Z., 13, 56, 64, 389, 451,
 506, 525-526, 540, 545, 578,
 575-577, 584-585, 587-589
 Du, D.Z., 578, 575-577, 584-585,
 587-589
 Dubois, O., 391, 400, 415, 451, 491,
 540
 Dudding, R.C., 86, 141
 Durbin, R., 279, 287
 Durkin, J., 301, 372
 Dyer, C.R., 523, 535, 558
- Eastman, C., 523, 540
 Ebenegger, C., 22, 57
 Ecker, K., 182, 199
 Eckstein, J., 94-95, 137
 Edmonds, J., 113, 140
 Eggleton, R.B., 314, 321, 372
 Eiger, G., 531
 Eiter, T., 525, 540
 Ekin, O., 101, 135
 Elfadel, I.M., 272, 287
 Ellickson, R., 524, 568
 Elliot, G., 400, 414-415, 547
 Elmohamed, S., 285, 287
- Elwin, J., 252
 Eppley, P.H., 35, 54
 Epstein, R., 41, 50
 Erdős, P., 18, 30, 52, 108, 140
 Erdős, R., 252
 Erdős, P., 323, 326-327, 372
 Eswaran, K.P., 523, 540
 Etherington, D.W., 476, 537
 Euler, R., 128, 132, 140-141
 Even, G., 228, 231-233, 240, 243-
 244, 247, 252
 Even, S., 392, 469, 540
 Ewashko, T.A., 86, 141
- Faigle, U., 191, 201
 Falkowski, B.-J., 423, 540
 Fang, L., 275, 281-282, 284-285,
 287
 Fang, M.Y., 394-395, 541
 Faugeras, O., 523, 541
 Faugeras, O.D., 523, 541
 Fausett, L., 275, 287
 Faybusovich, L., 272, 287
 Fedjki, C., 391-392, 401, 549
 Feige, U., 13-14, 48, 57
 Feldman, J., 41, 73
 Feldman, J.A., 523, 541
 Fellows, M.R., 252
 Felsner, S., 15, 57
 Feo, T., 563
 Feo, T.A., 30, 57, 71
 Ferland, J.A., 35, 57
 Fermat, P., 574, 588
 Fernandez de la Vega, W., 175-
 177, 182, 196, 201
 Festa, P., 238, 252
 Festa, Paola, 209
 Fiat, A., 191, 199
 Fiduccia, C.M., 524, 563

- Fillion, E., 557
Filonenko, V.L., 85, 140
Findler, J.D., 523, 541
Finklestein, L.A., 413, 532-533
Fiorini, C., 13-14, 55
Fischer, K.H., 267, 288
Fisher, D.C., 158, 201
Fisher, M.L., 41, 57
Fleurent, C., 35, 57
Floudas, C.A., 285, 288, 523, 541
Fonlupt, J., 15, 54, 216, 252
Forbes, G.W., 552
Ford, L.R., 82, 141
Fortin, D., 124, 141
Foster, J.A., 35, 57
Foulds, L.R., 309, 372
Fowler, R.J., 592, 602
Fox, G., 285, 287
Fox, M.S., 523, 541
Franco, J., 14-15, 55, 390, 400-401,
411, 413, 429, 472, 478,
489, 490, 492, 525, 531,
535, 541-542, 544, 561, 565
Franco, John, 379
Frank, A., 15, 57
Franklin, M.A., 571
Frankot, R.T., 523, 542
Fratta, L., 133, 138
Frayman, F., 523, 542
Frazier, R.A., 301, 372
Fredman, M.L., 92, 141
Freeman, J.W., 415, 542
Frenk, J.B.G., 110, 141, 170, 202
Freuder, E.C., 400, 413, 415, 542
Friden, C., 23, 36, 57-58
Friesen, D.K., 173-174, 184, 186,
192, 201-202
Frieze, A., 18, 30, 58
Frieze, A.M., 131-132, 141, 491,
499-501, 525, 532, 542
Fröhlich, K., 129-130, 138
Fu, K.S., 523, 571
Fu, X., 504, 542
Fujisawa, K., 7, 17, 58
Fujisawa, T., 15, 64, 66
Fujito, T., 225-228, 250
Fukao, T., 33, 74
Fulkerson, D.R., 113, 140, 314, 372
Fulkerson, R., 82, 113, 141
Funabiki, N., 33, 58
Funahashi, K.-I., 275, 289
Funke, M., 237-238, 248, 253
Fuqua, T.W., 524, 542
Füredi, Z., 331, 372
Gabow, H.N., 80, 114, 141
Gajski, D., 524, 532
Galambos, G., 154, 159, 164-165,
170, 175, 180, 182, 191,
199, 201-202
Galambos, Gabor, 151
Galil, Z., 392, 407, 542
Gallager, R.G., 299, 370
Gallier, J.H., 469, 540
Gallo, G., 86, 116, 138-139, 414,
476, 525, 542-543
Gamal, A.E., 524, 557
Gambosi, G., 180, 202
Gamst, A., 301, 346, 373
Ganesan, R., 33, 63
Gao, B., 13, 56, 578, 585, 587-588
Garcia-Molina, H., 523, 543
Gardner, P.C., 32, 50
Garey, M., 12, 46, 58, 154, 156,
176, 183, 187-188, 192, 194,
196, 200
Garey, M.R., 152-154, 159, 161,

- 171-173, 197, 202-203, 211,
253, 342, 373, 382, 422,
524, 543, 576, 588, 592,
602
- Garfinkel, R., 114, 142
- Garg, N., 253
- Gasarch, W.I., 14, 54
- Gaschnig, J., 400, 415, 523, 543
- Gauß, 574, 588
- Gautheret, D., 557
- Gavril, F., 15, 58, 216, 253, 257
- Gay, Y., 347, 372, 377
- Geiger, D., 213, 221-226, 240, 246,
250, 262, 288
- Gelat, C.D., 417, 483, 552
- Gelatt, C.D. Jr., 358, 374
- Gelatt, C.D., 31, 64, 261, 289, 592,
602
- Gelder, A.V., 391, 399, 15, 479,
525, 543
- Geman, D., 523, 543
- Gendreau, M., 23, 36-37, 58, 72
- Gendreau, T.B., 524, 559
- Genesereth, M.R., 390, 543
- Gent, I.P., 525, 544
- Gerards, A.M.H., 15, 59
- Gerhards, L., 20, 58
- Gershovitz, 531
- Ghosh, A., 390-391, 397, 530
- Giacomini, R., 39, 52
- Gibbons, L.E., 8-10, 35, 38, 59
- Gibbons, P.B., 198-199
- Gilbert, E.N., 574, 576, 588
- Gill, P., 24, 53
- Gilmore, P.C., 178, 182, 203, 392,
544
- Ginzberg, A., 523, 544
- Girosi, F., 262, 288
- Gislen, L., 285, 288
- Glauber, R.J., 267, 288
- Glicksberg, I., 113, 141
- Glover, F., 36, 59, 80, 92, 96-97,
108, 136, 142, 361-362, 373,
458, 460, 483, 544
- Glover, R., 92, 142
- Godbeer, G.H., 32, 59
- Goecke, O., 85, 140
- Goemans, M.X., 17, 59, 64., 110,
142, 226, 228, 242, 251,
253
- Goerdt, A., 491, 544
- Goertzel, G., 524, 564
- Goldberg, A., 392, 400-401, 412,
544
- Goldberg, A.V., 92, 95, 122, 139,
142
- Goldberg, D.E., 34, 59, 362, 373
- Goldberg, M.K., 40-41, 59
- Goldfarb, D., 100, 142
- Goldreich, O., 14, 51, 525, 530
- Goldsmith, J., 542
- Goldstein, R.A., 283, 286
- Goldwasser, S., 13-14, 57
- Golomb, S.W., 156, 203
- Golumbic, M.C., 15, 26, 59-60
- Gomory, R.E., 102, 136, 178, 182,
203
- Gonzalez, J., 114, 136
- Goodman, N., 523, 531
- Goossens, G., 483, 520, 524, 569
- Gotoh, S., 257
- Graham, R.L., 152-153, 159, 161,
171-172, 179, 190, 192, 197,
202-203, 575-576, 578, 584-
585, 587-589
- Grasselli, A., 524, 544
- Gray, F.G., 393-395, 557
- Gray, J.N., 523, 540

- Gregoire, E., 441, 511-512, 557
Griggs, J.R., 331, 372
Grigoras, G., 257
Grommes, R., 132, 141
Gross, O., 113, 141-142
Gross, O.A., 314, 372
Grossi, G., 33, 51
Grossman, T., 33, 60
Grötschel, M., 5-6, 15, 60, 237, 242,
 253, 318, 373
Grove, E.F., 181, 203
Grover, 524, 548
Gu, J., 383, 386-389, 390-397, 401,
 403-405, 411, 414-421, 423-
 424, 426-439, 441-445, 447-
 449, 451, 478-479, 481, 483,
 485-487, 489, 506, 511-512,
 519-520, 523-527, 540, 544-
 546, 550, 562, 566-567, 571
Gu, Jun, 379
Gu, Q.P., 389, 391-392, 416, 424,
 428-436, 441, 451, 506, 525,
 545-546
Guenoche, A., 248, 251
Guo, X., 247, 251
Gupta, R., 244, 251
Gupta, U.I., 15, 60
Gurevich, Y., 525, 547
Guzman, A., 523, 547
Gwan, G., 128, 147
Gyorgyi, G., 525, 553

Ha, S., 524, 547
Hadzilacos, V., 523, 531
Hager, R., 252
Hahn, W., 118, 120, 138
Hajós, G., 43, 60
Haken, A., 407, 415, 547
Hakimi, S.L., 16, 21, 48, 56

Hale, W.K., 308, 315-316, 325, 337,
 347, 357, 373
Hall, L.A., 190, 203
Hall, Ph., 78, 142
Halldórsson, M.M., 14, 31, 53
Hamachi, G.T., 524, 547
Hammer, P.L., 15, 22, 57, 60, 387,
 414, 474, 477-478, 493, 532,
 537, 547
Han, M.-H., 47, 60
Hanan, M., 524, 547
Hansen, E.R., 547
Hansen, P., 36, 60, 129, 142, 424,
 458, 460, 477, 479, 537,
 547
Harada, T., 33, 74
Haralick, R.M., 393-395, 400, 414-
 415, 523-524, 547, 557
Harary, F., 19, 45, 60, 247, 253,
 309, 373, 523, 547
Hardy, G.H., 86, 143
Harris, A.J., 323, 327, 370
Hasselberg, J., 28, 45, 61
Hastad, J., 14, 61
Haven, G.N., 383, 390-391, 411,
 437, 562
Havens, W.S., 524, 548
Haykin, S., 32, 61, 263, 388
Hayward, R.B., 15, 61
Hearn, D.W., 8-10, 35, 38, 59
Hedges, T., 524, 548
Hell, P., 15, 51, 575, 588
Heller, I., 15, 61
Helmberg, C., 17, 61
Henderson, T.C., 391, 393-396, 396,
 414-415, 419, 423, 436, 486,
 523, 538, 527, 546, 548,
 559, 564, 571
Hentenryck, P.V., 523, 539, 548

- Herault, L., 275, 277, 284-285, 288
 Hertz, A., 23, 36, 57-58, 61, 373
 Hertz, J., 32, 61
 Hertz, J.A., 267, 288
 Heusch, P., 548
 Hickman, 105, 108, 137
 Hifi, M., 35, 61
 Hilbert, D., 269, 287
 Hinton, G.E., 274-275, 286, 292,
 523, 529, 548
 Hipolito, A., 341, 368-370
 Hirasawa, K., 282, 286
 Hirata, H., 274, 278, 280, 291
 Hirschler, D.S., 183, 197, 200
 Ho, C.W., 15, 61
 Ho, Y.C., 400, 541
 Hoang, C.T., 15, 61
 Hochbaum, D., 153, 176, 184, 193,
 203, 226, 228, 248, 251,
 253
 Hochbaum, Dorit S., 592, 602
 Hofbauer, J., 38, 61
 Hoffman, A.J., 85, 143, 524, 539
 Hoffman, E.J., 423, 548
 Hofmann, T., 284, 288
 Hofri, M., 154, 204
 Holland, H., 362, 374
 Holland, J.H., 34, 62, 548
 Hollerbach, J.M., 527, 532
 Homer, S., 31, 62
 Hong, S.J., 524, 548
 Hood, P., 524, 548
 Hooker, J., 391-393, 401, 414, 445,
 462, 549
 Hooker, J.N., 471, 535
 Hoos, H.H., 531
 Hopcroft, J.E., 78, 143, 392, 529
 Hopfield, J.J., 32, 62, 261, 265-
 267, 274, 278, 288-289
 Hopkins, A.L., 524, 549
 Hopkins, G., 231, 250
 Horaud, P., 47, 52
 Horaud, R., 46-47, 62
 Horn, B.K.P., 523, 532, 549, 550
 Horn, R.A., 16, 62
 Horowitz, E., 400, 549
 Horst, R., 285, 289, 549
 Hou, E.S.H., 274, 285-286
 Houck, D.J., 21, 62
 Houweninge, M. van, 110, 141
 Hsiang, J., 390, 539, 549
 Hsu, W.L., 15, 62
 Hu, T.C., 247, 253
 Hu, T.H., 392, 400, 523, 525, 549
 Hualde, J.M., 41, 56
 Huang, J., 15, 51
 Huang, Wenqi, 591-592, 602
 Huang, X., 483, 524, 546, 550
 Hudry, O., 248, 251
 Huffman, D.A., 391, 550
 Huffman, J., 15, 33, 72
 Hummel, R.A., 415, 523, 550, 563,
 572
 Hung, M.S., 97, 100, 143
 Hunt III, H.B., 12, 62
 Hurkins, C., 336, 377
 Hurley, S., 360-361, 364, 371-372,
 374
 Hurwicz, L., 530
 Hwang, C.-R., 543
 Hwang, F.K., 574, 576-577, 587
 Ide, M., 20-21, 73
 Igarashi, H., 278-279, 289
 Ikeuchi, K., 523, 550
 Ikura, Y., 15, 62
 Illingworth, J., 415, 553
 Impagliazzo, R., 523, 550

- Imreh, B., 163, 165, 201
Indurkhyā, B., 524, 550
Ingber, L., 550
Isaacson, J.D., 523, 557
Isaak, G., 248, 253
Ishida, T., 523, 550
Ishii, S., 274, 278, 281, 289, 292
Itai, A., 30, 48, 392, 469, 540, 550
Itazaki, N., 524, 550
Ivković, 181, 204

Iwama, K., 390, 400, 402, 413, 451,
 550
Jackson Jr., P.C., 552
Jagota, A., 8-9, 14, 32-33, 62-63,
 70-71
Jahanian, F., 524, 550
Jang, D., 47, 60
Jansen, B., 341, 366-370, 369, 377
Jaumard, B., 36, 60, 424, 458, 460,
 477-479, 537, 547
Jayasri, T., 524, 551
Jensen, T.R., 309, 317, 348, 374
Jeroslow, R.E., 391, 411, 551
Jeroslow, R.G., 391-393, 445, 551
Jerrum, M., 18, 31-32, 63
Johnson, C.R., 16, 62
Johnson, D., 12, 46, 58
Johnson, D.B., 244, 254
Johnson, D.S., 12, 20, 18, 28-29,
 31, 35, 39-40, 63, 88, 143,
 153-154, 156, 159, 161-1655555,
 171-174, 176-177, 179, 183,
 187-188, 192, 194, 196-197,
 200, 202-204, 211, 253, 342,
 373, 382, 412, 422-524, 526-
 527, 529, 543, 551, 576,
 588, 592, 602
Johnson, J.L., 391, 444, 551

Johnson, L.F., 20, 63
Johnson, R., 391, 394-396, 436, 486,
 527, 567
Johnson, R.R., 485, 551
Johnson, R.W., 551
Johnson, T.L., 527, 532
Johnson, W. Lewis, 423, 428, 551
Johnston, H.C., 20, 63
Johnston, M.D., 417, 421, 423, 428,
 551, 558
Johri, P.K., 301, 374
Jones, A.E.W., 552
Jones, M.H., 530
Jonker, R., 92, 105, 122, 143
Josephson, J.R., 523, 552
Josephson, N., 390, 539
Ju, Y.C., 524, 552

Kaderman, J.D., 554
Kajitani, Y., 257
Kaller, D., 15, 52
Kamath, A., 491, 552
Kamath, A.P., 88, 147, 391-396,
 401, 445, 461-462, 507, 509-
 510, 512, 552
Kanter, I., 267, 289
Kaplan, P.D., 40, 68
Kapoor, A., 471-472, 536
Kapsalis, A., 364, 376
Karger, D., 374
Karger, D.R., 191, 204
Karloff, H., 191, 199
Karmarkar, N., 18, 63, 177, 182,
 186, 196, 204, 365, 374,
 462, 465, 552
Karmarkar, N.K., 88, 147, 391-396,
 401, 445, 461-462, 507, 509-
 510, 512, 552
Karney, D., 96, 142

- Karp, R.M., 12, 41, 63, 78, 109, 111, 113, 127, 143, 177, 182, 186, 196, 204, 211, 254, 527, 529
- Karpinski, M., 14-15, 41, 56, 64
- Kasap, Suat, 259
- Kashiwabara, T., 15, 64, 66
- Katseff, H.P., 159, 199
- Katzela, I., 301, 374
- Kaufman, L., 129, 142
- Kautz, H., 430-431, 441, 457, 460, 491, 510, 512, 516-517, 565
- Kawakami, J., 282, 286
- Kedem, G., 524, 552
- Kella, J., 524, 552
- Keller, O.H., 43, 64
- Keller, R.M., 524, 556
- Kellerer, H., 190, 204
- Kennedy, R., 95, 122, 142
- Kennington, J., 105, 144
- Kenyon, C., 175, 205
- Kerbosch, J., 19-21, 53
- Kern, W., 191, 201
- Keutzer, K., 524, 539
- Kevorkian, A.K., 254
- Khanna, S., 198, 205
- Khuri, S., 34, 49
- Kikusts. P., 22, 64
- Kilian, J., 14, 57
- Kilpelainen, P., 525, 540
- Kim, D., 267, 292
- Kim, D.S., 64
- Kim, H., 246, 254
- Kim, P., 247, 256
- Kinnersley, N.G., 185, 205
- Kinoshita, K., 524, 550
- Kirkpatrick, D.G., 523, 537
- Kirkpatrick, S., 15, 64, 261, 289, 358, 374, 417, 483, 525, 552-553, 592, 602
- Kirousis, L.M., 491, 553
- Kishi, Y., 301, 374
- Kittler, J., 415, 553
- Kleer, J. de, 415, 523-524, 538
- Klein, D.J., 247, 253-254
- Klein, P.N., 15, 64
- Kleinberg, J., 17, 64
- Kleinschmidt, P., 100, 104, 134, 144
- Kleitman, D.J., 331, 372
- Klingman, D., 92, 96-97, 108, 136, 142
- Klinz, B., 85, 116, 121, 131, 138, 144
- Knälmann, A., 359, 375
- Knight, J.P., 524, 560
- Knuth, D., 553
- Knuth, D.E., 17, 64, 400, 553
- Kodandapani, K.L., 524, 553
- Kodilian, M., 110, 142
- Kogan, A., 414, 532
- Kohata, Y., 23, 73
- Kohavi, Z., 235, 255
- Kojima, M., 7, 17, 58
- König, D., 82, 144
- Kopf, R., 28-29, 64
- Korn, R.K., 524, 560
- Korneeko, N.M., 523, 572
- Korst, J., 31, 32, 47, 358, 370
- Kosaraju, S.R., 527, 529
- Kosowsky, J.J., 272, 283, 289, 293
- Koutsoupias, E., 391-392, 416, 560
- Kowalski, R., 523, 553
- Kozen, D., 46, 64
- Kozuka, S., 283, 289, 364, 374
- Kramer, M.R., 524, 553
- Kranakis, E., 491, 553
- Kratsch, D., 216, 250

- Krause, K.L., 189, 205
Krishna, C.M., 524, 553
Krivelevich, M., 18, 48
Krizanc, D., 491, 553
Krogh, A., 32, 61
Kruskal, J.B., 575, 588
Ku, D.C-L, 553
Kuh, E.S., 524, 537
Kuhn, H.W., 89, 144
Kullmann, Oliver, 399, 553
Kumar, V., 423, 428, 553-554
Kumar, V.K.P., 556
Kunzmann, A., 244, 254
Kurita, N., 275, 289
Kurokawa, T., 283, 289, 364, 374
Kurtz, T.G., 496, 554
Kurzberg, J.M., 112, 144

Laarhoven, P.J.M. van, 31, 65
Lagarias, J.C., 43, 64
Lagerholm, M., 266, 274, 285, 289
Lagnese, E.D., 524, 554
Laguna, M., 36, 59
Lai, C.W., 112, 135
Laird, P., 417, 423, 428, 558
Lakhani, G., 524, 554
Lam, C.W.H., 413, 534
Landman, B.S., 524, 554
Landweer, P.R., 135
Lanfear, T.A., 302, 325, 343, 354,
 374
Langston, M.A., 173-174, 184, 185-
 186, 192, 194, 202, 205
Lapalme, G., 557
Laporte, G., 24, 53
Larrabee, T. 390, 414, 512, 520,
 524, 555
Lassez, C., 523, 555
Lauritzen, S.L., 246, 254

Lavagno, L., 414, 519-522, 554, 565
Lawler, E.L., 15, 65, 92, 144, 190,
 205, 523, 554
Lazarus, A.J., 109, 144
Le Verge, H., 132, 141
Lee, C.C., 159, 164-165, 167-169,
 205-206
Lee, C.W., 100, 144
Lee, C.Y., 479, 555
Lee, D., 244, 254
Lee, D.T., 15, 60, 159, 164-165,
 167-169, 205-206
Lee, E.A., 524, 547, 555
Lee, H.S., 15, 55
Lee, K., 15, 33, 65, 72
Lee, K.-C., 274, 284, 289-290
Lee, K.C., 33, 58
Lee, R.C.T., 15, 61, 392, 400, 523,
 525, 549, 555
Lee, Y., 88, 144
Lee., C., 524, 555
Leeuwen, J. van, 524, 553
Lehoczky, J.P., 524, 555
Leifman, L.J., 20, 65
Leighton, T., 233, 242, 247, 254
Leiserson, C.E., 537
Lempel, A., 255
Lengauer, T., 524, 539
Lenstra, H.W., 182, 205
Lenstra, J.K., 29, 48, 190, 205, 336,
 377, 523, 554
Lettmann, T., 408, 533
Leuker, G.S., 15, 71
Leung, D.S.P., 306, 345, 374
Leung, J.Y.-T., 194, 200-201
Leung, J.Y.T., 15, 60
Levesque, H., 386, 391-392, 397,
 401, 416, 428, 431, 436,
 440, 507, 509, 525, 558,

- 565
 Levin, L., 523, 550
 Levy, B.C., 275, 290
 Levy, H., 215, 237, 249, 255
 Lewis, H.R., 475, 555
 Li, G.-J., 395, 555
 Li, G.J., 395, 555, 570
 Li, J., 524, 555
 Li, Qinghua, 592, 602
 Li, T., 275, 281-282, 284-285, 287
 Li, Wei, 592, 602
 Li, X., 247, 255
 Liang, F.M., 170, 205
 Liang, Y.D., 216, 251, 255
 Liao, Y.Z., 524, 555
 Libchaber, A., 40, 68
 Lichtenstein, D., 556
 Lieberherr, K.J., 411, 556
 Liebling, T.M., 127, 147
 Liggins, M., 125-126, 147
 Lin, B., 483, 520, 524, 569
 Lin, C.S., 556
 Lin, F., 33, 65
 Lin, F.C.H., 524, 556
 Lin, J., 523, 559
 Lin, K.J., 556
 Lin, S., 416, 429, 556
 Lin, W.M., 556
 Lin, Y-J, 554
 Lin, Y.J., 395, 556
 Lindenberg, W., 20, 58
 Linhart, J., 576, 587
 Lippmann, R.P., 523, 556
 Lipscomb, J., 32, 59
 Lipton, R.J., 14, 65, 556
 Little, W.A., 290
 Littlewood, J.E., 86, 143
 Liu, C.L., 524, 539
 Liu, D.D., 319-320, 350, 375
 Liu, J., 231, 255
 Liu, L.L., 394-395, 535
 Liu, S., 40, 68
 Liu, Z.C., 575, 577, 584-585, 587-588
 Lizhoudu, 426, 431-432, 437-438, 483, 511, 546
 Lloris-Ruiz, A., 524, 568
 Lloyd, E., 181, 204, 214, 221, 257
 Lloyd, E.L., 244, 255
 Locke, C.D., 523, 556
 Loessi, J.C., 423, 548
 Lofaso, B.J., 523, 558
 Logemann, G., 390-391, 396-397, 406, 409, 506, 538
 Looi, C.-K., 32, 65, 275, 278, 284, 290
 Lorie, R.A., 523, 540
 Loukakis, E., 20, 22, 26, 65
 Lovász, L., 5-6, 10, 13-15, 17, 57, 60, 65, 237, 242, 253, 318, 373
 Loveland, D., 390-391, 396-397, 406, 506, 538
 Loveland, D.W., 390-391, 396-397, 406, 556
 Lowe, J.K., 391-393, 445, 531
 Lowe, L., 215, 237, 249, 255
 Lowrie, M.B., 395, 570
 Lozano-Perez, T., 527, 532
 Lu, Chin Lung, 216, 251
 Lu, X., 231, 258
 Luby, M., 32, 41, 59, 65, 523, 525, 530, 550
 Lucchesi, C.L., 247, 255
 Luccio, F., 524, 544, 556
 Luccio, F.L., 218-220, 245, 255
 Lueker, G.S., 154, 175-177, 182, 196, 201

- Luenberger, D.G., 386, 448-449, 451, 453, 467, 556
Lund, C., 14, 48, 51, 54, 255, 530
Lynch, N.A., 523, 556
- Ma, H.T., 524, 539
Maass, Wolfgang, 592, 602
MacWilliams, J., 42, 66
Machkworth, A.K., 390, 414, 523, 556
Machol, R.E., 86, 144
Maffioli, F., 133, 138
Maffray, F., 15, 61
Maftouhi, A. El, 491, 557
Magazine, M.J., 197, 206
Maghout, K., 19, 66
Magnanti, T.L., 87, 135
Magos, D., 133-135, 144
Major, F., 557
Makowsky, J., 469, 550
Makowsky, J.A., 525, 557
Malik, J., 523, 557
Malik, S., 480, 557
Manacher, G.K., 15, 41, 66
Mankus, T.A., 15, 41, 66
Mann, R., 262, 274-276, 286
Mannila, H., 525, 540
Mannino, C., 40, 66
Mao, W., 163, 205
Marathe, M.V., 12, 62, 216, 255
Marble, W.G., 523, 557
Marchant, J.A., 47, 70
Marchiori, E., 35, 66
Marcus, P.M., 19, 66
Marks, D.H., 523, 559
Marple, D., 524, 557
Marsh, A.B., 102, 140
Marsland, T.A., 523, 557
Martel, C.U., 174, 205
- Martello, S., 88, 117, 122, 139, 144-145
Martello, Silvano, 151
Martin, R.E., 301, 372
Martinez, D., 284, 287
Mason, M.T., 527, 532
Masuda, S., 15, 64, 66
Mathon, V., 477, 547
Matias, Y., 198-199
Matsuda, S., 278-279, 290
Matula, D.W., 18, 66, 344, 375, 523, 557
Mautor, T., 41, 56
Maydan, D., 523, 557
Mayr, E.W., 175, 199
Mazure, B., 441, 511-512, 557
McAllester, D., 523-524, 557, 572
McCall, J.T., 393-395, 557
McCormack, W.M., 393-395, 557
McCullough, W.S., 262, 290
McGeoch, C.C., 88, 143
McGeoch, C.H., 527, 529
McGrath, E.J., 524, 553
McGregor, J.J., 523, 558
McLean, C.R., 558
McLoughlin, A.M., 551
Meeusen, W., 2, 20, 66
Mehlhorn, K., 78, 92, 115, 135, 523, 558
Méjean, H.-M., 397, 401, 558
Melamed, I.I., 266, 275, 277-278, 281-282, 284-285, 290
Mellish, C.S., 536
Mercure, H., 24, 53
Merlin, P.M., 523, 535
Metze, G., 44, 70
Metzger, B.H., 315, 344, 375
Meyniel, H., 15, 67
Mézard, M., 110, 145

- Mezard, M., 266-267, 290
 Michalewicz, Z., 558
 Middlecamp, L.C., 302, 308, 375
 Miliotis, P., 133, 135, 144
 Miller, D., 88, 105-107, 136, 145
 Miller, T.K., 262, 274-277, 286, 292
 Minker, J., 19-20, 49
 Minoux, M., 478, 558
 Minton, S., 417, 423, 428, 558
 Minty, G.J., 15, 67
 Miranker, D.P., 523, 558
 Mitchell, D., 386, 391-392, 397, 401,
 416, 428, 431, 436, 440,
 507, 509, 525, 558, 565
 Mitsuhashi, T., 568, 524
 Mitsuma, S., 29, 73
 Mittal, S., 523, 542
 Mitzenmacher, M., 501, 558
 Mizuike, T., 301, 374
 Mohar, B., 18, 67
 Mohr, R., 559
 Mok, A.K., 524, 550
 Monien, B., 222, 255, 395, 399,
 415, 486, 559
 Monma, C.L., 88, 92, 140
 Montanari, U., 524, 544
 Moon, C., 414, 519-522, 554, 565
 Moon, J.W., 21, 67
 Moore, R.C., 423, 548
 Morel, H., 397, 401, 558
 Mori, S., 524, 559, 567
 Morris, P., 457, 559
 Moser, L., 21, 67
 Motwani, R., 14, 48, 374, 491, 530,
 552
 Motzkin, T.S., 8, 67
 Mueller, R., 15, 57
 Müller, B., 359, 375
 Muller, B., 266, 290
 Mulligan, G.D., 19, 67
 Mulmuley, K., 79, 145
 Murgai, R., 414, 565
 Murgolo, F.D., 179-180, 206
 Muroga, S., 524, 548
 Murphey, R., 127, 145
 Murphey, Robert A., 295
 Murphy, W.J., 14, 67
 Murtagh, B.A., 462, 559
 Murthy, A.S., 34, 67
 Nadel, B.A., 414, 523, 559
 Naghshineh, M., 297, 301, 374
 Nair, K.P.K., 114, 147
 Nakajima, K., 15, 64, 66
 Nakata, K., 7, 17, 58
 Naor, J., 15, 41, 55, 67, 213, 221-
 225, 240, 246, 250
 Naor, M., 15, 67
 Naor, S., 228, 231-233, 240, 243-
 244, 247, 252
 Navinchandra, D., 523, 559
 Nemhauser, G.L., 21, 25-27, 67-
 68, 342, 375
 Nemhauster, G.L., 15, 62
 Neng, B., 110-111, 145
 Nešetřil, J., 14, 49
 Newell, A., 523, 559
 Newton, A.R., 524, 534, 539
 Ni, L.M., 524, 559
 Niehaus, W., 40, 49
 Nieuwkoop, J.-J., 275, 277, 284-285, 288
 Nijenhuis, A., 523, 559
 Nilsson, N.J., 390, 392, 523, 543,
 559
 Nishizawa, M., 524, 560
 Nishizeki, T., 14-15, 20-21, 41, 55
 Norvig, P., 423, 427-428, 564

- Oden, P.H., 524, 564
Ogawa, H., 47, 68
Ohlsson, M., 282,-284 290
Oklobdzija, V.G., 524, 534
Olariu, S., 15, 68
Opsut, R.J., 329, 375
Orenstein, T., 235, 255
Orlin, J.B., 87-88, 92, 95, 97-98,
 135, 144-145
Ortega-Lopera, J., 524, 568
Osteen, R.E., 20, 68
Oosterhout, J.K., 524, 547
Ouyang, Q., 40, 68

Pachter, L., 247, 256
Padberg, M.W., 24, 68
Page, E.S., 114, 117, 145
Pager, D., 523, 560
Palem, K., 491, 552
Palmer, R.G., 32, 61
Pan, Y.F., 534
Panconesi, A., 13, 68
Pandu Rangan, C., 216, 255
Papadimitriou, C., 12, 14-15, 68,
 256
Papadimitriou, C.H., 12, 20, 63,
 261, 290, 391-392, 396, 416,
 424, 430, 523, 553, 560
Paparrizos, K., 101, 104, 134, 146
Pardalos, P.M., 1-2, 7-10, 23, 26,
 28, 33, 35, 37-39, 41, 48,
 54, 59, 61, 68-69, 110, 127,
 145-146, 238, 249, 252, 256,
 285, 288-289, 317, 375, 453,
 523, 526, 540-541, 560
Pardalos, Panos M., 1, 209, 295
Paris, W.D., 524, 536
Parisi, G., 110, 145, 266-267, 290
Park, K., 34-35, 54, 69

Parthasarathy, G., 34, 67
Pataki, G., 27, 34, 49
Patel, A.M., 524, 560
Paterson, M.S., 592, 602
Patterson, W., 523, 560
Paul, M., 78, 115, 135
Paulin, P.G., 524, 560
Paull, M., 400-401, 490, 541
Paull, M.C., 19, 69
Pearl, J., 245, 252, 414, 538, 560
Peay, Jr, E.R., 41, 69
Peemöller, J., 342, 375
Pehoushek, D., 560
Peinado, M., 31, 62
Pekny, J., 88, 105-107, 136, 145
Pelavin, R., 524, 534
Pelc, A., 44-45, 51
Peleg, D., 245, 256
Pelillo, M., 1, 8-9, 11, 27, 33, 38-
 39, 47, 52-53, 70
Pelillo, Marcello, 1
Penna, M.A., 523, 560
Pereira, L.M., 533, 561
Perl, J., 246, 254, 256
Perron, O., 43, 70
Peters, J., 105, 146
Peterson, C., 32, 70, 262, 266, 270,
 274-280, 282, 284-285, 288-
 291
Petreschi, R., 250
Petrie, C., 415, 561
Pevzner, P.A., 41, 73
Peysakh, J., 479, 525, 561
Pferschy, U., 115-116, 146, 190, 204
Philips, A.B., 417, 423, 428, 558
Phillips, A.T., 9, 23, 38-39, 69
Phillips, C., 107, 146
Phillips, S.J., 191, 204
Pi, H., 282,-284 290

- Picard, J.C., 5, 23, 58, 70
 Picco, P., 266, 286
 Pierskalla, W.P., 123, 128, 146
 Pinter, R.Y., 41, 73
 Pitassi, T., 504, 530, 561
 Pitsoulis, L.S., 127, 145
 Pitts, W., 262, 290
 Pla, F., 47, 70
 Plaisted, D., 390, 539
 Plass, M.F., 392, 530
 Plateau, G., 547
 Plato, D.L., 524, 535
 Plotkin, J., 400-401, 561
 Plotkin, S.A., 95, 142
 Poljak, S., 18, 67
 Pollak, H.O., 574, 576, 588
 Pólya, G., 86, 143
 Pomeranz, I., 235, 255
 Poore, A.B., 125-126, 146-147
 Popplestone, R.J., 13, 46, 48
 Porto, A., 561
 Posa, L., 252
 Postiglione, A., 180, 202
 Potin, D.P. La, 524, 561
 Pradhan, D.K., 524, 561
 Prawitz, D., 561
 Preparata, F.P., 44, 70
 Pretolani, D., 414, 476, 478, 525,
 543, 561
 Price, K., 523, 541
 Protasi, M., 12, 37, 49, 50
 Pulleyblank, W.R., 15, 60, 70, 117,
 144
 Punnen, A.P., 114, 147
 Purdom, P., 390, 397, 410, 525,
 533-534, 561
 Purdom, P.W., 383, 390-392, 400-
 403, 410-411, 413, 429, 437,
 473, 489, 523, 532-533, 544,
 561-562
 Purdom, Paul W., 379
 Puri, R., 397, 417, 432, 448, 481,
 483, 519-520, 524, 545, 562
 Pusztaszeri, J., 127, 147
 Putnam, H., 390-391, 396-397, 406,
 409, 538
 Qi, L., 128, 136, 147
 Qian, F., 274, 278, 280, 291
 Qian, T., 238, 249, 256
 Quellmalz, A., 359, 375
 Queyranne, M., 5, 70
 Quinn, M.J., 415, 542
 Rabani, Y., 191, 199
 Radhakrishnan, V., 12, 62
 Radig, B., 47, 70
 Raghavan, P., 14, 71
 Ralf, K., 301, 373
 Ramachandran, 241, 256
 Ramakrishnan, K.G., 18, 63, 88,
 110, 146, 147, 391-396, 401,
 445, 461-462, 507, 509-510,
 512, 552
 Ramamritham, K., 524, 567
 Ramanan, P., 167-169, 206
 Ramanujam, J., 32, 71, 275-277,
 284-285, 291
 Randić, M., 247, 254
 Rangan, C.P., 217, 252
 Ranjan, D., 13, 68
 Rao, B., 524, 552
 Rao, C.D.V.P., 524, 563
 Rao, M.R., 471-472, 536
 Rao, S., 233, 242, 247, 254
 Rao, V.B., 417, 564
 Rappe, J., 23, 69
 Rave, W., 301, 346, 373

- Ravi, R., 216, 255
Raychaudhuri, A., 314-315, 319, 325,
 349, 352, 375
Rayward-Smith, V.J., 364, 376
Read, R.C., 523, 563
Reddy, M.R., 87, 135
Reddy, S.M., 32, 71-72
Reed, B., 499, 536
Reed, G.M., 524, 563
Reedy, S., 244, 254
Reeves, C.R., 285, 291
Regan, K.W., 33, 63
Reichl, L.E., 266, 291
Reichling, M., 423, 563
Reinelt, G., 237-238, 248, 253
Reingold, E.M., 390, 410-411, 413,
 531
Reinhardt, J., 266, 290
Rendl, F., 17, 39, 53, 61, 80, 117,
 138, 147-148, 317, 375
Rensing, P.E., 127, 147
Rényi, A., 108, 140
Resende, M., 563
Resende, M.G.C., 18, 23, 30, 41,
 48, 57, 63, 69, 71, 238, 249,
 252, 256
Resende, Mauricio G.C., 209, 295
Ressell, S., 423, 427-428, 564
Reynaud, G., 397, 401, 558
Richards, D.S., 574, 588
Richey, M.B., 169, 206
Rijavec, N., 125-126, 147
Rinnooy Kan, A.H.G., 110, 113,
 141, 143, 190, 205, 523,
 554
Rivenburgh, R.D., 40, 59
Rivest, 523-524, 563
Rivest, R.L., 537
Roberts, F.S., 314-318, 323-324, 329,
 331-332, 337, 347-351, 353,
 357, 371-372, 375-376
Robertson III, A.J., 126, 147
Robertson, E.L., 400, 413, 562
Robinson, G.A., 408, 571
Robinson, J.A., 390-391, 396, 406-
 407, 563
Robson, J.M., 22, 71
Rodgers, G., 7, 33, 41, 69
Rodgers, G.P., 23, 26, 69
Rovainen, P., 284, 286
Rom, W.D., 100, 143
Roohy-Laleh, E., 98, 147
Roos, C., 341, 366-370, 377
Roscoe, A.W., 524, 563
Rose, D.J., 15, 71
Rosen, B., 256
Rosen, J.B., 524, 560, 567
Rosenfeld, A., 415, 523, 563, 572
Rosenfeld, E., 523, 529
Rosenthal, J., 400-401, 561
Rosiers, W., 415, 563
Ross, I.C., 19, 60
Rossi, C., 39, 52
Rote, G., 80, 148
Rotem, D., 15, 71
Roth, R.M., 213, 221-225, 240, 246,
 250
Rothstein, J., 132, 136
Roucaïrol, C., 41, 56
Rubin, A.L., 323, 326-327, 372
Rubinstein, J.H., 576, 589
Rudeanu, S., 387, 477, 547
Rudolf, R., 85, 116, 121, 124, 129,
 131, 138, 141, 144
Ruehli, A.E., 524, 564
Ruhe, G., 28-29, 64
Rumelhart, D.E., 274, 292
Rushforth, C.K., 431, 432, 483, 571

- Russakoff, A., 77, 136
 Russell, A., 14, 51
 Russo, R.L., 524, 554, 564
 Saab, Y.G., 416, 564
 Sachs, H., 16, 56
 Sadayappan, P., 32, 71, 275-277,
 284-285, 291
 Safra, S., 13-14, 49, 57
 Sagan, B.E., 15, 71
 Sahni, S., 193, 206, 400, 549
 Sais, L., 441, 511-512, 557
 Saito, N., 14-15, 41, 55
 Sakai Troxell, D., 377
 Sakai, D., 331-333, 376
 Saks, M., 474, 493, 532
 Salamon, P., 252
 Saldanha, A., 414, 524, 564-565
 Saleh, R.A., 524, 552
 Saltzman, M.J., 128-129, 131, 136
 Salvail, L., 23, 36-37, 58
 Salzer, H.E., 156, 206
 Samal, A., 394-395, 523, 548, 564
 Samuelsson, H., 25-26, 50
 Sanchis, L., 14, 33, 63, 71
 Sangiovanni-Vincentelli, A., 414, 480,
 519, 520-522, 524, 554, 557,
 563-566
 Santomauro, M., 524, 563
 Sargent, J.S., 530
 Sassano, A., 15, 40, 56, 66
 Sastry, V.U.K., 34, 67
 Sato, M.-A., 274, 278, 281, 289,
 292
 Saunders, Ma.A., 462, 559
 Savoj, H., 414, 565
 Schaefer, T.J., 392, 564
 Schaeffer, J., 523, 557
 Schaffer, A.A., 15, 67
 Schannath, H., 100, 144
 Scheidt, G. vom, 361, 371
 Schieber, B., 228, 231, 233, 240,
 243-244, 247, 252
 Schiele, W.L., 524, 564
 Schiermeyer, I., 415, 564
 Schlag, M.D.F., 524, 534
 Schlipf, J., 478, 531, 542
 Schlipf, J.S., 472, 565
 Schmitz, L., 423, 540
 Schnitger, G., 13, 51
 Schrader, R., 85, 140
 Schrijver, A., 5-6, 10, 15, 59-60,
 65, 242, 253, 318, 373
 Schultz, R., 222, 255
 Schwartzschild, B..M., 417, 565
 Schwarz, J.T., 79, 148
 Schwarz, P., 524, 565
 Schwetman, H.D., 189, 205
 Scutella, M.G., 469, 476, 542, 565
 Sechen, C., 524, 565
 Sejnowski, T.J., 274-275, 286, 523,
 548
 Selman, B., 386, 391-392, 397, 401,
 416, 428, 430-431, 436, 440-
 441, 457, 460, 491, 507,
 509-510, 512, 516-518, 525,
 558, 565
 Sentovich, E.M., 414, 565
 Sequin, C.H., 524, 566
 Servatius, B., 524, 530
 Sethi, R., 524, 529
 Seymour, P.D., 233, 242, 256
 Sha, L., 524, 555
 Shamir, A., 214, 221, 242, 256, 392,
 469, 540
 Shang, Y., 386, 388, 390-391, 455,
 460, 512, 517, 519, 570
 Shannon, C.E., 17, 71

- Shao, J., 132, 148
Shapiro, E., 567
Shapiro, L.G., 523-524, 547
Sharell, A., 525, 557
Shatcher, R.D., 246, 256
Shaw, A.C., 244, 257
Shearer, J.B., 42, 54
Shen, Y.Y., 189, 205
Shensa, M.J., 390, 565
Shi, R.C.-J., 392, 401, 467, 566
Shi, W., 524, 530
Shim, G.M., 267, 292
Shin, H., 524, 566
Shin, K.G., 524, 553, 566
Shin, S.-Y., 40, 74
Shirakawa, I., 20-21, 73
Shmoys, D., 184, 193, 203
Shmoys, D.B., 190, 205, 523, 554
Shor, N.Z., 6, 71
Shor, P.W., 43, 64
Shragowitz, E., 524, 567
Shrivastava, Y., 32, 71-72
Siddiqi, K., 47, 70
Siegel, P., 390, 566
Sigismundi, G., 26-27, 67
Sigmund, K., 38, 61
Silvestri, R., 13-14, 55
Simchi-Levi, D., 156, 200, 206
Simeone, B., 478, 537
Simon, H.A., 523, 559
Simon, J.C., 524, 566
Simonis, H., 523, 539
Simovici, D.A., 257
Singh, K.J., 414, 565
Sipala, P., 400, 567
Skiena, S.S., 359, 376
Skordas, T., 46-47, 62
Sloane, N.J.A., 42, 54, 66, 72
Smith, B., 88, 105, 139
Smith, D.H., 302, 303, 336, 343, 360, 374, 376
Smith, G.D., 364, 376
Smith, G.W., 214, 234, 236, 257
Smith, J.W., 523, 552
Smith, K.F., 396, 419, 571
Smith, S.H., 30, 57, 71
Smith, W.D., 42, 54
Snyder, W.E., 262, 274-276, 286
Söderberg, B., 32, 70
Soderberg, B., 262, 266, 270, 274-280, 282, 284-285, 288-291
Sodini, C., 92, 139
Soffa, M., 214, 221, 257
Soffa, M.L., 244, 255
Sompolinsky, H., 267, 289
Song, Yimin, 592, 602
Soong, N.L., 524, 560
Soriano, P., 23, 36-37, 58, 72
Sós, V.T., 9, 72
Sosić, R., 391, 394-396, 411, 416-417, 423, 428-432, 436, 483, 486, 527, 523, 566-567
Soukup, J., 524, 567
Soule, T., 35, 57
Specher, E., 411, 556
Speckenmeyer, E., 231, 233, 257, 391, 394-396, 399, 412, 415, 467, 486, 531, 542, 559
Spector, A., 524, 565
Spencer, T., 40-41, 59
Spiegelhalter, D.J., 246, 254
Spieksma, F.C.R., 124, 131, 136, 139
Spirakis, P., 491, 552
Srinivas, M.A., 32, 50
Srinivasan, V., 102, 148
Stahl, S., 329, 376
Stallman, R., 415, 567

- Stamm, H., 233-234, 257
 Stankovic, J., 524, 567
 Stankovic, J.A., 524, 567
 Staton, W., 231, 250
 Stearns, R.E., 12, 62
 Steiglitz, K., 261, 290, 523, 560
 Stein, S.K., 43, 72
 Stephan, P.R., 414, 565
 Stephens, N., 362, 371
 Stephens, N.M., 361, 364, 371-372
 Sterling, L., 567
 Stix, V., 11, 25, 27, 35, 39, 53
 Stolorz, P., 271, 278, 281, 292, 293
 Stone, H.S., 400, 413, 421, 524,
 550, 567
 Stone, J.M., 400, 413, 421, 567
 Straus, E.G., 8-9, 67, 72
 Sudakov, B., 18, 48
 Sudan, M., 14, 48, 51, 233, 240,
 243, 247, 252, 374, 530
 Suen, C.Y., 524, 559, 567
 Suen, S., 491, 499, 542
 Sun, X., 474, 532
 Sussman, G.J., 415, 567
 Sutanthavibul, S., 524, 567
 Swain, M.J., 395, 536
 Swaminathan, R., 413, 472, 525,
 541-542, 565
 Swaminathan, R.P., 542, 567
 Szabó, S., 43, 72
 Szabo, S., 55
 Szegedy, M., 13-14, 48, 57, 530
 Szeliski, R., 279, 287
 Szemerédi, E., 390, 407, 415, 491,
 504, 535
 Szolovits, P., 246, 256
 Tadei, R., 23-24, 55
 Taillard, E., 36, 61
 Takahashi, H., 20-21, 23, 29, 73
 Takashima, M., 524, 568
 Takefuji, Y., 15, 32-33, 58, 72, 274,
 284, 289-290
 Talamo, M., 180, 202
 Tanaka, A., 20-21, 73
 Tang, C.Y., 392, 400, 523, 525,
 549, 571
 Tang, Chuan Yi, 216, 251
 Tanimoto, S.L., 592, 602
 Tank, D.W., 32, 62, 261, 265, 274,
 278, 289
 Tanner, M.C., 523, 552
 Tarjan, R.E., 15, 21-23, 71-72, 80,
 92, 95, 114-115, 135, 141-
 142, 148, 253, 257, 392,
 530, 575, 586
 Taulbee, O.E., 19, 51
 Taylor, H., 323, 326-327, 372
 Taylor, J.G., 361, 371
 Terlaky, T., 341, 366-370, 377
 Tesman, B.A., 317, 321, 326-330,
 349, 353-354, 376-377
 Thayse, A., 568
 Thoai, N.V., 285, 289
 Thomas, D.A., 576, 589
 Thomas, D.E., 524, 554
 Thomborson, C.D., 524, 534
 Thompson, G.L., 102-105, 107, 145,
 148
 Timermans, X., 524, 569
 Ting, D.W., 194, 201
 Tinhofer, G., 23-24, 49
 Tiourine, S., 336, 377
 Tishby, N., 525, 553
 Toft, B., 309, 317, 348, 374
 Tokuda, H., 524, 568
 Tomati, L., 354, 371
 Tomita, E., 20-21, 23, 29, 73

- Törn, A., 453, 568
Torng, E., 191, 204
Toth, P., 21, 50, 88, 92, 105-106,
 114, 117, 122, 136, 139,
 144-145
Totik, V., 195, 201
Tou, J.T., 20, 68
Trabado, P.P., 524, 568
Trafalis, Theodore B., 259
Traiger, I.L., 523, 540
Trick, M.A., 18, 28, 35, 39-40, 63,
 526, 551
Trojanowski, A.E., 21-23, 72
Tront, J.G., 393-395, 557
Trotter, L.E., 25-27, 67
Troyansky, L., 525, 553
Truemper, K., 390-391, 472-473,
 568
Tseitin, G.S., 390, 407-409, 568
Tsouros, C., 20, 22, 26, 65
Tsukiyama, S., 20-21, 73
Turán, P., 8, 73
Turan, G., 182, 191, 201
Turcotte, M., 557
Turner, K.J., 523, 568
Tutte, W.T., 79, 148
Tuy, H., 549
Tygar, J.D., 524, 568
Tyler, J.E., 393-395, 568
Tyshkevich, R.I., 523, 572

Ueno, S., 257
Ueno, S.-I., 284, 292
Uesaka, Y., 275, 292
Uhry, J.P., 15, 53
Ullman, J.D., 392, 523-524, 529,
 569
Ullmann, J.D., 152-153, 159, 161,
 171-172, 202, 204

Unbehauen, R., 275-278, 281, 287,
 536
Unger, S.H., 19, 69
Upfal, E., 500-501, 525, 532
Urahama, K., 284, 292
Urban, S.D., 523, 569
Urbani, G., 414, 543
Urquhart, A., 390, 407, 415, 561,
 569
Urrutia, J., 15, 71
Utans, J., 271, 293

Vaidya, P., 95, 142
Vairaktarakis, G., 28, 45, 61
Valenti, R., 247, 254
Valiant, L.G., 13, 73, 78, 148
Valtorta, M., 569
Van Den Bout, D.E., 262, 274-277,
 286, 292
Van Hoesel, C.P.M., 341, 369
Van Laarhoven, P.J.M., 262, 292
Vanbekbergen, P., 483, 520, 524,
 569
Vanderbei, R.J., 17, 61
Vannelli, A., 392, 401, 467, 566
Vannicola, V., 125-126, 147
Varadarajan, R., 524, 554
Vardadym, V.A., 117, 136
Vazirani, U.V., 79, 145
Vazirani, V., 229-230, 257
Vazirani, V.V., 79, 145, 253
Vecchi, M.P., 31, 64, 261, 289, 358,
 374, 417, 483, 552, 592,
 602
Vega, W.F. de la, 491, 557
Vemuganti, R.R., 21, 62
Verbitsky, O., 13, 73
Verfaillie, G., 284, 287
Verschoor, A.J., 27, 49

- Vigo, Daniele, 151
 Villa, T., 524, 564
 Vinay, V., 414, 549
 Vincentelli, A.S., 524, 539
 Vingron, M., 41, 73
 Vinograd, S., 79, 139
 Virasoro, M.A., 266-267, 290
 Vlach, J., 392, 401, 467, 566
 Vlach, M., 132, 148
 Vliet, A.van, 165, 170, 185, 191-192, 200, 206
 Vohra, R., 191, 199
 Vohra, R.V., 113, 143
 Volgenant, A., 92-93, 105, 122, 143, 148
 Vornerger, O., 395, 486, 559
 Vušković, K., 471-472, 536
- Wagner, D.K., 567
 Wah, B., 386, 388, 395, 489, 544, 570
 Wah, B.W., 386, 388, 390-391, 395, 429, 455, 460, 512, 517, 519, 535, 544, 555, 570-572
 Wah, Benjamin W., 379
 Waigard, F., 248, 251
 Walford, R.B., 214, 234, 236, 257
 Walkup, D.W., 109, 111-112, 148
 Walsh, T., 525, 544
 Waltz, D., 391, 523, 570
 Wan, P.J., 584-585, 587, 589
 Wang, A., 480, 557
 Wang, C., 214, 221, 257, 331-332, 376
 Wang, C.C., 244, 255
 Wang, D.-I., 321, 349, 377
 Wang, D.I., 320, 325, 349, 353, 372
 Wang, H., 283, 286, 524, 568
- Wang, J., 261, 275, 278, 281, 292, 391, 411, 551
 Wang, W., 391, 393-396, 414-415, 419, 423, 431-432, 436, 483, 486-487, 527, 546, 571
 Wang, Z., 105, 144
 Warmuth, M.K., 175, 199
 Warners, J., 368-370
 Warners, J.P., 366-367, 377
 Watanabe, F., 301, 374
 Watanabe, H., 524, 552
 Wee, T.S., 197, 206
 Wei, W., 132, 148
 Wein, J., 88, 105, 107, 148-149
 Weintraub, A., 41, 50
 Wells, M.B., 413, 571
 Wendorf, J., 524, 568
 Wernisch, L., 15, 57
 Werra, D. de, 117, 144, 347, 372
 Werra, D., 377
 Werra, D.de, 22-23, 36, 57-58, 61
 White, J., 524, 539
 White, M., 262, 274-276, 286
 Wigderson, A., 41, 48, 63
 Wilf, H.S., 17, 23, 73, 343, 370, 377, 523, 559
 Williams, H.P., 392, 445, 571
 Williams, R.J., 274, 292
 Williamson, D.P., 226, 228, 242, 251, 253
 Wilson, A., 88, 105, 139
 Wilson, W.H., 281, 287
 Wimer, S., 41, 73
 Winston, P.H., 390, 571
 Winter, P., 574, 588
 Wisse, A., 337, 377
 Witte, E.E., 571
 Woeginger, G.J., 116, 121, 124, 131, 138, 144, 149, 154, 180,

- 165, 191-192, 200-202, 206
Wolff, P.K., 524, 547, 564
Wolkowicz, H., 17, 61, 317, 375
Wolsey, L.A., 21, 41, 57, 68, 342,
375
Wong, C.K., 183, 197, 200, 524,
555, 591
Wood, D.E., 554
Wos, L., 408, 571
Woude, M. van der, 524, 569
Wu, F.Y., 267, 293
Wu, J., 33, 74
Wu, L.C., 400, 571
Wu, W., 13, 56
Wu, Y., 524, 550
Wu, Yu-Liang, 591
Wunderlich, H.J., 244, 254

Xi-Cheng, L., 524, 550
Xu, C., 524, 559
Xu, K., 172, 206
Xu, L., 274, 293
Xu, Xiangyang, 592, 602
Xue, J., 2, 23-24, 26, 40-41, 50, 69,
74

Yadegar, L., 131, 141
Yakimovsky, Y., 523, 541
Yamada, T., 284, 292
Yamamoto, K., 524, 559
Yan, J.-H., 15, 54
Yannakakis, M., 12-15, 20, 63, 68,
73-74, 211, 216, 248, 253,
255-258
Yao, A.C.-C., 166, 170, 173, 175,
206
Yao, E.N., 576, 587
Ye, Y., 256
Yong, T.Y., 523, 571

Yoshida, K., 524, 568
Younger, D.H., 247, 255, 258
Yu, C.F., 570-572
Yu, C.S., 14, 22, 26, 40, 50
Yu, L., 15, 74
Yu, M.-S., 15, 74
Yu, Xiangdong, 592, 602
Yu, Y., 274, 285-286
Yue, M., 172, 192, 207
Yuille, A.L., 271-272, 274, 278-280,
283, 287, 289, 293
Yung, M., 384, 423, 529

Zabih, R., 523, 572
Zaki, H., 105, 149
Zang, W., 233-234, 250
Zelikovsky, A., 14-15, 64
Zemlyachenko, V.N., 523, 572
Zenios, S., 88, 105, 107, 146, 148-
149
Zhan, Shouhao, 592, 602
Zhang, B.-T., 40, 74
Zhang, F., 247, 251, 255
Zhang, G., 163, 185-186, 200, 207
Zhang, S., 572
Zhao, C., 231, 255
Zheng, M., 231, 258
Zhiying, J., 114, 135
Zhou, S., 524, 572
Zhu, Hong, 592, 602
Žilinskas, A., 453, 568
Zimmermann, U., 114, 118, 120,
122, 138, 140
Zivković, 247, 254
Zivkovic, T.P., 247, 253
Zoellner, J.A., 297, 299-300, 344,
367, 377
Zosin, L., 228, 231-232, 244, 252
Zubieta, L., 23, 58

Zucker, S.W., 47, 70, 415, 523, 550,
563, 572

Zuckerman, D., 48

Zuiderweg, E.R.P., 283, 286

Subject Index

- $C^{(h)}$ algorithm, 168
 K^{th} Adjacent Channel Constraints, 307
 N_r -coloring, 331
 N_r^k -coloring, 332-333
 ϵ -complementarity conditions, 94
 λ -coloring, 329
 λ -chromatic number, 329
(K+1)-homogeneous indifference system, 353
0-1 linear program, 84
0-1 threshold property, 501
2-3-subgraph, 223
2-distant set, 331
2-SAT formula, 468
2-SAT Solver, 469
3d-queen problem, 421
absolute worst-case ratio, 156
action, 417, 594
activation function, 263
actual gap, 174
added in, 28
adjacency matrix, 3, 9
adjacent channel constraints, 307
adjacent vertices, 310
admissible order, 315
admissible transformations for 3-DAP, 129-130
admissible transformations, 89, 119-121
admissible vertex order, 313
advanced search heuristics, 30-37
affine plane, 574
algebraic assignment problem (AAP), 118
Algorithm 457, 20
algorithm clustering approach, 487
algorithm tool kit, 488
allowed vertices, 223
Almost Any-Fit class, 157, 161-162
Almost Any-Fit constraint, 162
almost minimal feedback vertex set, 227
Almost Worst-Fit (AWF), 162
alternating path, 91
alternating tree, 90-91
an integer point integer programming algorithm, 509
an integer programming formulation for SAT, 461-462
anomalous algorithm, 179
anti-ferromagnetic model, 266
any variable heuristic, 429
Any-Fit class, 157, 161-162
Any-Fit constraint, 162
approximate solutions, 397
approximate(), 447
arboricity, 21
arc capacities, 81
arc set, 81
area cost operator, 103
aspiration level, 36
assignment constraints, 76
assignment polytope, 77
assignment problem, 76, 283
assignment, 76
association graph, 46
asymptotic performance ration (APR) 155-156
asymptotic worst-case ratio, 155-156

asynchronous circuit technique, 483
atomic repacking move, 181
attributed relational structures, 46
auction algorithm, 93-95, 106-107
augmenting path, 91
available item, 197
average 1-SAT model, 401
average time, 399
axial 3-dimensional assignment problems (3-DAP), 123-131

B rule, 185
backtracking algorithm, 409-412
backtracking method, 19, 342-343
backward arcs, 92, 97
bad variable heuristic, 429
balanced ($0, \pm$) matrices, 471-472
balanced assignment problems, 117
balanced solver, 472
Banach-Mazur compactum, 581
Banach-Minkowski planes, 585
band, 300
bandwidth of a graph, 317
base model, 421
basic local search, 433
basin, 425
basis update, 108
Bayesian inference, 245
Bayesian network, 246
BDD SAT solver, 520
BDD SAT-Circuit solver, 521
Best in heuristic, 28
best legal neighbor, 36
best neighbor direction, 428, 440
best neighbor, 440
Best Two-Fit (B2F) algorithm, 173
Best-Fit (BF), 159
Best-Fit Decreasing (BFD) algorithm, 171

Best-Fit Randomized (BFR), 175
Best-k-Fit, 163
best-neighbor heuristics, 428-429
bidding the update, 95
bin capacity, 154
bin current content, 155
bin current level, 155
bin packing anomalies, 178-179
bin packing problem, 152
bin size, 154
Binary-Decision-Diagram (BDD), 479
bipartite graphs, 15
bipartite tournaments, 234
blackout vertices, 223
block Gauss-Seidel methods, 106
block, 362
body, 346
Boolean relaxation procedure, 418
bootstrapping, 231
bottleneck assignment problems, 113-118
bottleneck Monge matrix, 116
bottom level, 426
bounded look-ahead, 181
bounded-space algorithm, 162-166
bounded-space on-line algorithm, 157
branch and bound method, 21, 463-464
branch merging, 413
branchpoint-free cycle, 223
branchy graph, 224
branchpoint, 212
broadcast network, 300
Buffered Next-Fit (BNF), 180
butterflies, 216, 218
butterfly graph, 219

- c-chromatic, 312
c-colorable, 312
c-cycle, 492
canonical ordering, 217
capacity constraints, 81
cardinality of the matching, 77
cardinality, 3
carrier frequency, 300
Cartesian product of vertices, 311
Cauchy/Schwarz inequality, 17
CC-balanced formula, 471-472
cell cost operator, 102
cellular network base station assignment, 301
channel assignment algorithm, 484
channel loss, 299
channel separation matrix, 308
channel, 300
characteristic vector, 4, 9, 229
children chromosome, 363
choice number of a graph, 327
chordal and interval graphs, 216
chordal graph, 311, 315
chromatic number, 17
chromosome, 363
circuit network, 247
Circuit Partitioning Problem, 284
circuit, 311
classical NP-hard tournament problem, 248
clause area, 413
clause order backtracking, 410
clause-flow model, 494
clique number, 3, 15, 312
clique tree algorithm, 246
clique, 3, 124, 312
closed bin, 155
closeness, 447
close_to_solution(), 447
closing rule, 163, 185-186
Clustering Problem, 284
Clustor, 487
co-channel constraints, 307
co-channel FAP, 315
co-site interference, 304-305
cocomparability graphs, 216
code division multiple access (CDMA), 299
cogitative approach, 592
coloring, 14, 312
column reductions, 120
Combinatorial Algorithms for Military Applications (CALMA) project, 336
communications, 524
compatible ordering, 313
compatible, 45
competitive ratio, 156
competitive simplex algorithm, 100
complement graph, 3, 310
complement(x), 489, 496
complete graph, 3, 312
complete local search, 437
complete SAT algorithm (CSAT), 396, 415
complete set, 234
complete_flip(), 438
computer architecture design, 524
computer graphics, 524
computer science and artificial intelligence, 523
computer-aided manufacuring, 523
combined nonlinear optimization algorithm, 451
conditioning method, 246
configuration, 594
conflict, 427
conjugated values, 574

- conjunctive normal form (CNF) formula, 478
 Conjunctive Normal Form (CNF), 386
 connected graph, 15
 consecutive k-tuple coloring, 330
 consecutive k-tuple T-coloring, 330
 consecutive T-set, 318
 conservative algorithm, 179
 constrained programming algorithms, 392-393
 constraint network, 245
 constraint satisfaction problem (CSP), 245, 284, 383
 constructive partitioning, 483
 Continuous Constrained Formulations, 389-390
 continuous function, 386
 continuous Lagrangian search algorithms, 453
 continuous nonconvex optimization problem, 4
 continuous objective function, 452
 continuous objective value, 452
 Continuous Unconstrained Formulations, 388-389
 continuous-based heuristics, 37-39
 contracted graph, 215
 contraction operations, 214
 convergence property, 451
 convergence ratio, 451
 convex bipartite graphs, 80, 216, 221
 convex, 80
 copositivity property, 24
 corner position, 594
 corner-occupying conjecture, 593
 corner-occupying packing sequence, 596
 corner-occupying packing, 593
 cost operation algorithms, 88
 cost operator algorithms, 102-103
 covered column, 82
 covered row, 82
 covering problem, 239
 critical linkpoint, 223
 crossover operation, 364
 cubic graph, 13
 current unit set, 467
 cut sets, 98
 cut, 464
 cutting-plane method, 464
 cycle space, 229
 cycle, 212, 311
 cycle-free set, 217
 cycle-free subgraph, 217
 cycle-free vertex set (CVS), 216
 cycle-packing property, 247
 cyclically reducible graphs, 214
 cyclomatic number, 229
 cyclomatic weight function, 230
 cylinders of solutions, 397
 d-monoid, 118-119
 d-partite graph, 124
 database system, 523
 Davis-Putnam resolution, 408
 Davis-Putnam-Loveland (DPL) procedure, 406
 deadlock prevention problem, 244
 decomposition method, 346
 decomposition phase, 230
 deficit forest, 105
 degree of edging, 595
 degree of packing discontinuity of rational action, 595
 degree of saturation largest first (DSATUR), 24

degree, 310
delta function, 268-269
depth of a corner positions, 595
depth strategy, 595
depth-first, 345
destructive partitioning, 483
dijoin, 242
DIMACS benchmark graphs, 39
DIMACS graph, 33, 35
DIMACS instance, 512
DIMACS testbed, 35
directed acyclic graph (DAG), 422
directed graph, 311
directed multicut problem, 247
Discrete Constrained Feasibility Formulations, 386
Discrete Constrained Formulations, 388
discrete CSP model, 383
discrete gradient, 456
discrete Lagrangian function, 455
Discrete Lagrangian Method (DLM), 456
discrete Lagrangian search algorithm, 455-460
discrete local solution, 8
Discrete Unconstrained Formulations, 386-388
discrete, constrained algorithms, 390-392
discrete, unconstrained algorithms, 392
disguised Horn formulas, 470
Disjunctive Normal Form (DNF), 386
disordered magnetic model, 266
distance, 574
diversify, 36
divide and conquer, 463

divisible sequence, 182-183
dominating set problem, 13
double reduction, 236
doubly stochastic matrix, 77
Dual First-Fit (DFF[r]), 195
Dual Next-Fit (DNF), 195
dual planes, 574
dual quadratic estimates, 6
dual simplex-based algorithms, 99-101
dynamic bin packing, 187-189
earliest opened bin, 155
edge formulation, 5
edge set, 3
edging strategy, 595
effective energy function, 272
eigenvalue bound identity, 17
eigenvalue, 16
endpoint, 211
energy function, 32
enumerative algorithms, 19-21
equal circles packing problem, 592-593
European Cooperation on the Long term in Defense (EUCLID) problem, 336
evaluate_objective_function(), 433, 436-438
evaluate_object_function(), 447
even arc, 100
exact solutions, 397
existence of a perfect matching, 78
expander graphs, 13
Expectation-Maximization (EM) algorithm, 271
extended Horn formula, 471
extended k-multiple, 320
extended literals, 418

- extended partition, 236
- extended resolution, 409
- extension phase, 230
- F rule, 185
- F*D integer constraints, 308-309
- far-site interference, 303-304
- favorable case, 396
- feasible (primal or dual) forests, 104
- feedback arc set problem, 239-244
- feedback set problem, 210-211
- feedback vertex set problem, 212-238
- feedback vertex set, 212
- Fermat function, 579
- ferromagnetic model, 266
- FFD using Largest bins, and Repack (FFDLR), 186
- FFD using Largest bins, and Shift (FFDLS), 186
- first improvement strategy, 361
- First-Fit Decreasing (FFD) algorithm, 171
- First-Fit Increasing (FFI) heuristic, 194
- first-order necessary condition for discrete problems, 456
- First_Fit (FF), 159
- fixed-clause-length model, 401
- flat-move strategy, 460
- flow conservation constraints, 81
- flow update, 108
- flow, 81
- forbidden difference graph, 312
- forcing number, 247
- forcing problem, 247
- forest algorithms, 88, 104-105
- forward arcs, 92, 97
- free edges, 90-91
- free nodes, 90-91
- frequency assignment problem (FAP), 298-309
- frequency assignment, 296
- frequency, 300
- frequency-distance function (F*D), 300
- frequency-distance interference constraints, 300
- fully polynomial approximation scheme (FAS), 221
- Gauss-Seidel methods, 106
- Gauss-Seidel version of the auction algorithm, 95
- Generating Radiolink frequency Assignment Problems Heuristically (GRAPH), 368
- generic discrete Lagrangian algorithm, 456
- genetic algorithms (GA), 34-36, 362-364
- Genetic Engineering via Negative Fitness (GENF), 25
- GENF algorithm, 36
- GENF procedure, 35
- gentle_flip(), 438
- Gilbert-Pollak conjecture, 577
- girth, 231
- global convergence, 386
- global heuristics, 357
- global maximizer, 9-10
- global minimum point, 507
- global quadratic zero-one problem, 7-8
- goal of SAT problem, 382
- Golomb conjecture, 157
- good, 45

- Graph Bi-Partitioning Problem, 276
graph bipartization problem, 211
graph degree, 211
graph packing, 198
Graph Partitioning Problem, 277
graph, 310-311
gravity strategy, 594-595
greedy algorithm, 225-226, 313
Greedy k-tuple T-coloring algorithm,
 353-354
greedy local search algorithm, 428
Greedy local search, 440-441
greedy randomized adaptive search
 (GRASP) heuristics, 126
Greedy Randomized Adaptive Search
 Procedure (GRASP), 238
greedy strategy, 457
Greedy T-coloring algorithm, 347-
 353
Greedy T-coloring of complete graph,
 348-351
greedy, 23
Group-X Fit Grouped (GXFG), 174
GSAT algorithm, 508
GSAT procedure, 440
GSAT with random walk, 510-511
GUC(F) algorithm, 498
GUCB, 499-500

Hamiltonian cycle, 311
Hamiltonian path, 311
Hamiltonian r-path, 311
Hamiltonian, 77, 311
Hamming distance, 42
Hamming graph, 42
hard-and-easy distributions, 402
HARMONIC+1 algorithm, 169
Harmonic-Fit algorithm, 164
head, 346

Hessian, 386
hidden Horn 1-SAT, 492
hidden Horn, 492
high-speed networking, 524
hill-climbing, 457
hitting cycle problem, 210
homomorphic, 310
Hopfield NN, 267
Horn formulas, 469-470
Horn 1-SAT, 492
Horn solver, 470
Hungarian method, 89-91
hybrid algorithm, 487
hyper-queen problem model, 421

identity matrix, 9
improve, 91
improving phase, 362
incident, 211, 310
incoming arc, 211
incoming edge, 211
incomplete SAT algorithm, 396
independent graph, 312
independent set formulation, 5
independent set, 3
independent subgraph, 312
indifference graph, 314
induced subgraph, 310
input neurons, 263
integer programming problem, 4,
 461-466
integrality gap, 226
integrated circuit design automa-
 tion, 524
intelligent backtracking, 415
intensify, 36
intercommodity constraints, 232
interference diagram, 301
interference power, 299

- interior point algorithm, 88
- interior point method, 465
- interior point, 465-466
- intermodulating co-site interference, 305-306
- intersaturated vertices, 232
- Interset graph, 355-357
- interval graphs, 15
- interval representation, 216
- inverse bottleneck Monge matrix, 116
- inverse Monge property, 85
- Ising-glass model, 267-272
- isomorphic, 310
- Iterated First-Fit Increasing (IFFD) algorithm, 194
- Iterated Lowest-Fit Decreasing (ILFD) algorithm, 196
- Iterated Worst-Fit Decreasing (IWFD) algorithm, 189
- iteration, 456
- j-unit sphere graph, 313
- Jacobi methods, 106
- Jeroslow-Wang, 411-412
- Jerrum conjecture, 18
- job dispatcher, 488
- job queen, 422
- k-bounded space, 162
- k-choosable graph, 327
- k-dimensional cube connected cycle (CCC.k), 220
- k-interchange heuristics, 29
- k-multiple, 319
- k-neighbor, 29
- k-tuple coloring number, 328
- k-tuple coloring, 328
- k-tuple T-coloring, 330
- Karmarkar's method, 466
- Knapsack Problem, 282-283
- KORBX(R) parallel/vector computer 509-510
- König's Matching Theorem, 82
- L-reduction, 13
- l-SAT distribution, 489-
- l-SAT model, 401
- Lagrangian methods, 453
- Largest Memory First (LMF) algorithm, 189
- largest-first, 344
- last maximal independent set, 12
- Latin squares, 132-133
- lattice tiling, 43
- leftmost bin, 155
- Lexicographic bottleneck assignment problem (LexBAP), 117
- lexicographic product, 311
- likelihood ratio, 126
- linear algorithm, 493
- linear bottleneck assignment problem (LBAP), 87, 113-116
- linear program relaxation, 462
- linear relaxation problem, 5
- linear relaxation, 365-366
- linear resolution, 408
- Linear Sum Assignment Problems (LSAPs), 83-108
- linkpoint, 211
- list coloring, 326
- list concatenation, 155
- List Scheduling (LS) algorithm, 191
- local conflict minimization procedure, 418
- local convergence rate, 386
- local descent, 435
- local handler, 429

- local maximizer, 9
local minimum point, 507-508
local optimization, 416
local optimum, 416
local search algorithms, 30
local search heuristics, 29-30
local search, 416-442
local tracking heuristics, 431
local-handler, 449
locality of the circuit, 236
Longest Processing Time (LPT), 192
lookahead processor, 414
loop cutset problem, 225
loosely-coupled parallel computing, 485
Lovász number, 17
lowest indexed bin, 155

machine vision, 523
Marriage theorem, 78
match, 45-46
matched edges, 90-91
matched forest, 104
matched nodes, 90-91
matching in directed bipartite graph, 109-113
matching problem, 84
matching, 77
mathematics, 523
MAX Π_1 , 13
MAX 3-SAT problem, 13
max cut problem, 13
Max Flow-Min Cut Theorem, 82
MAX NP, 12
MAX SNP, 12-13
maximal 2-3-subgraph, 223
maximal alternating tree, 91
maximal clique, 4, 8-9, 11, 312
maximal independent subgraph, 312
maximal match, 46
maximal strongly connected component of a graph, 234
maximum clique problem, 2, 3-5, 7-8, 12-47, 284
maximum clique, 4, 7, 9
Maximum distance, 574
maximum independent set problem, 3, 7
maximum independent set, 7, 12
maximum match, 46
maximum matching problem, 77-80
maximum network flow problem, 81
maximum satisfiability (Max-SAT) problem, 424
maximum weight clique problem, 3
maximum weight clique, 11
maximum weight independent set problem, 3, 6, 8
maximum weight independent set, 8
maximum weighted 2-colorable subgraph problem, 217
maximum weighted 2-independent set problem, 217
Max_Time, 438
Max_Trapping-Times, 438
mean field annealing (MFA), 261-262, 272-274
mean field annealing heuristic, 33
mean field theory, 266-274
meaningful measurement, 349
measurement theory, 349
meshes, 216, 218
Metropolis process, 31

- Meyniel graphs, 15
 MFA algorithm, 273
 middle level, 426
 min-conflict formulation, 417-421
 min-conflict heuristic (boolean space), 427
 min-conflict heuristic (discrete space), 427
 min-conflict heuristic, 419, 427
 minimum feedback arc set problem, 239
 minimum feedback vertex (arc) set problem, 211
 minimum feedback vertex set, 212
 Minimum Spanning Tree (MST), 575-576
 minimum vertex cover problem, 3, 12
 minimum weighted feedback vertex set problem (MWFVS), 213
 minimum weighted graph bipartition problem, 239
 minimum weighted vertex cover problem, 4
 Minkowski's conjecture, 43
 mixed phase, 362
 mobility, 336
 Modified First-Fit (MFF), 188
 Modified First-Fit Decreasing (MFFD), 173
 Modified Harmonic-2 (MH2), 168
 Modified Harmonic-Fit (MHF), 167
 Monge array, 124
 Monge matrix, 85, 121
 Monge properties, 85
 monotonic algorithm, 179
 most-constrained search, 413
 Most-k-Fit (MF_k) algorithm, 173
 Motzkin-Straus class, 12
 Motzkin-Straus problem, 10
 Motzkin-Straus theorem, 9-12
 moved out, 28
 multi-dimensional assignment problems (MAP), 123-134
 multi-index assignment problems, 123
 multi-level bottleneck assignment problem, 116
 multi-node algorithms, 105-106
 multi-SAT algorithm, 486-488
 Multifit algorithm, 192
 multigraph, 310
 multiphase search heuristics, 432
 multispace search algorithm, 483
 multispace search heuristics, 432
 multispace search, 481
 mutation operation, 364
 mwff generator, 510
 N-coloring, 331
 n-queen problem, 421
 n-queen scheduling model, 421-423
 n-queens problem, 512
 N-resolution, 408
 natural ordering, 313
 near-optimal N_r -coloring, 332
 near-optimality, 332
 neighborhood, 3, 416
 network layer, 299
 neural networks (NNs), 32-33, 261-266, 364-365
 neurons, 263
 New heuristic, 29
 Next-Fit (NF), 158
 Next-Fit Decreasing (NFD) algorithm, 171
 Next-Fit rule, 171

Next-Fit, using Largest possible bins, 185
NFL, 185
Next-k-Fit (NF_k), 163
nm queen, 421
no-hole ($r+1$)-distant coloring, 331
no-hole coloring, 331
no-hole k-tuple ($r+1$)-distant coloring, 333
node set, 81
node, 3
non-binary instances, 512
nonclausal inference algorithm, 478
nongreedy search strategy, 26
nongreedy, 23
nonlexicographic ordering search, 413
NP-complete, 12

objective function, 7, 76
obstruction, 313
obtain_a_SAT_instance(), 445
occurrences(x, C), 489
odd arc, 100
off-diagonal elements, 7
off-line algorithm, 153, 171-179
on-line algorithm, 153, 157-171
on-line bin-packing algorithm, 157
open bin, 154-155
open search, 426
opening rule, 185-186
optimum set, 234
order, 315
Ordered Binary Decision Diagrams (OBDDs), 479
Ordered First-Fit Decreasing (OFFD) algorithm, 197
oriented graph, 311
outgoing arc, 211
outgoing edge, 211

output neurons, 263

p-norm, 574
P-resolution, 408
packing problem, 213, 239
packing rule, 163, 185-186
packing value, 597
pair labelling, 331
parallel local search, 436
parallel, constrained programming algorithms, 395
parallel, discrete, constrained algorithms, 393
parallel, discrete, unconstrained algorithms, 395
parallel, unconstrained, nonlinear optimization algorithms, 395
parent chromosome, 363
partial backtracking technique, 345
partial k-tree, 477
partial random variable selection heuristics, 430
partition function, 268
path, 212, 311
peak search, 426
penalty function approach, 31
perfect elimination ordering, 313
perfect graph, 6, 14, 213-214, 312
perfect matching in directed bipartite graph, 109-113
perfect matching, 77-80
perfect vertex order, 313
perfectly orderable graphs, 15, 314
performance ratio, 222
perform_flip(), 433
perform_min(), 447
permanent, 78
permutation array, 127
permutation graphs, 216

permutation matrix, 76
 permutation, 76
 permuted Monge cost matrix, 85
 permuted Monge matrix, 85
 Perron eigenvector, 16-18
 Perron root, 16
 physical layer, 299
 pivot processor, 107
 pivot rule, 98
 $\text{PL}(\mathbf{F})$, 500
 planar 3-dimensional assignment problems (3-PAP), 132-134
 planar counting problems, 12
 planar embedding, 217
 planted forest, 104
 plateau, 425
 plateaus in the search space, 458
 point removal method, 19
 polyhedral algorithm, 298
 polyhedral model, 298
 polyhedral pointed cone, 11
 polynomial approximation scheme (PAS), 222
 polynomial-time approximation algorithm, 12
 polynomial-time approximation scheme, 12
 Potts-glass model, 267-272
 practical application problems, 405
 pre-random variable selection heuristics, 430
 prefix algorithm, 194
 primal simplex-based algorithms, 96-98
 primal-dual algorithms, 88-96
 primal-dual shortest path algorithms, 105-106
 primal-simplex-based algorithms, 107-108

probabilistic time, 400
 probe order backtracking, 410
 problem instance database, 488
 product of graphs, 311
 pseudoflow algorithms, 95-96
 pseudoflow, 95
 pure literal rule algorithm, 412
 pure literal, 409
 purely infeasible simplex algorithm, 101
 q-queen problem, 421
 q-blocked c-cycle, 492
 q-Horn formulas, 474-475, 492
 QS algorithm, 421-423
 Quadratic Assignment Problem, 281-282
 quadratic expressions, 7
 quadratic formulation, 366
 quadratic Lyapunov function, 32
 quadratic relaxation, 366-367
 quadratic terms, 7
 quadratic zero-one problems, 6
 quadratically constrained global optimization problem, 6
 quasi parity graphs, 15
 quasi-polynomial time, 13
 queuing strategy, 596
 quotient cuts, 242
 r-initial T-set, 318
 r-path, 311
 racing condition problem, 244
 random 2-out bipartite graphs, 113
 random exchange, 436
 random flip heuristic, 429
 random flips, 435-436
 random 1-SAT formula, 490
 random swap, 429

- random value (assignment) heuristic, 429
random variable (selection) heuristic, 429
random vertex order, 344
random walk, 441
random-clause-length model, 401
randomized heuristics, 30
randomized local search, 432-437
randomized polynomial-time approximation scheme, 13
ratio scaled T-set, 349
rational action, 594
reassignable denying set, 345
recovery procedure, 126
recursive proofs, 21
reduced costs with respect to dual solution, 89
reduction graph, 224
Refined First-Fit (RFF) algorithm, 166
Refined First-Fit Decreasing (RFFD), 173
Refined Harmonic-Fit (RHF), 167
regular resolution, 408
regular SAT models, 405
reinforcement learning strategy, 33
relational structures, 45
relaxation approaches, 88
relaxation labeling algorithms, 38
relaxed multicommodity flow, 232
relaxing, 298
Repacking algorithm (REP_3), 180
replicator equations, 38
reservation techniques, 164
Restricted Candidate List (RCL), 238
restricted span coloring, 357
reverse of perfect elimination ordering, 313-314
Revised Warehouse (RW) algorithm, 181
robotics, 523
rounding, 462
row and column reductions, 90
row reductions, 120
S-list T-coloring of a graph, 327
saddle point approximation, 268-272
Saddle Point Theorem, 454
saddle-point, 454
Sandwich theorem, 17
sat state, 437
SAT-circuit solver, 519
SAT1.5 algorithm, 513
SAT14.11 procedure, 450
SAT14.5 procedure, 448
SAT6.0 procedure, 446
satisfiability (SAT) problem, 382
satisfiability index, 475
saturation degree of vertices, 346
SC(F) algorithm, 499
scaling techniques, 94
Scheduling Problem, 284
search algorithm, 385
search rearrangement, 413
security, 524
select_an_initial_solution(), 445
self-dual, 574
semi-on-line algorithms, 180-182
semidefinite programming, 17
semidisjoint cycle, 227
sequential assignment method, 297
sequential corner occupying manner of packing, 593
sequential greedy heuristics, 28-29

- series-parallel graphs, 15
- set coloring, 328
- set covering approach, 342-343
- set covering problem, 213
- set measure, 234
- set packing, 198
- set T-coloring, 329-330
- shifting strategy, 592
- short-term memories, 36
- shortest augmenting path algorithms, 91-93
- shortest clause backtracking, 410
- sifting property of delta function, 268-269
- signature methods, 99
- simple backtracking, 410
- simple genetic algorithm, 34
- Simplex method, 465
- simplex-based algorithms, 96-101
- simplex-like methods, 88
- simplicial vertex, 312
- Simplified Harmonic (SH_k) algorithm, 165
- simulated annealing, 30-32, 261-162, 358-361
- single instruction stream multiple data stream (SIMD), 105
- single node algorithms, 105
- single-lookahead unit resolution, 472
- singleton, 212
- sink, 81
- SLUR algorithm, 472-474
- smallest clause rule, 497, 501
- smallest-last, 344
- Smith-Walford one-reducible, 214
- solution_testing(), 447
- source, 81
- span, 315
- sphere growing, 243
- spin-glass model, 266
- spin-glass, 266
- spurious solution problem, 9
- stability number, 3
- stable set, 3
- Stamm, 414
- standard quadratic problems, 24
- standard simplex, 4, 8
- Steiner Minimal Tree (SMT), 574-575
- Steiner points, 574-575
- Steiner Ratio, 576-586
- Steiner's Problem of Minimal Trees, 575
- stochastic steepest descent heuristic, 33
- strong trees, 101
- strongly divisible, 183
- strongly dual feasible trees, 100
- strongly feasible trees, 97
- strongly t-perfect graphs, 15
- strong_flip(), 438
- subgraph approach, 40
- subgraph induced by, 3
- subgraph, 212, 310
- subset feedback vertex (arc) set problem, 211
- subset minimum feedback vertex (arc) set problem, 211
- subset, 362
- subsumption, 409
- superforests, 104-105
- support set, 408
- surplus forest, 105
- switch mode, 86
- switch set, 492
- system-level fault diagnosis, 43-44
- T-admissible coloring, 322-323

- T-admissible sequence, 322-323
T-choice number, 327
T-coloring theory, 297
T-coloring, 316
T-edge span, 317, 324
T-graph G_T , 312
T-k-choosable graph, 327
t-labelling, 331
T-order, 316, 324
t-perfect graphs, 15
T-set, 317
T-span, 316, 324
tabu list, 36, 361, 460
tabu local search, 441
Tabu search for SAT, 511
tabu search, 36-37, 361
Tabu Thresholding (TT), 362
tail, 346
task graph, 422
telecommunication problem, 86
temperature, 31
terminal assignment problems, 80
terrace, 425
testing, 417
test_flip(), 433
test_min(), 447
text processing, 524
the algorithm of Balinski and Go-
mory, 102
the chain of subsets in G , 20
thinning out the underlying bipar-
tite graph, 92
threshold algorithms, 92, 114
threshold value, 92, 114
tightly coupled parallel computing,
486
tiling, 43
time division multiple access (TDMA),
299
time slot assignment problem, 116
time-division-multiple-access (TDMA)
model, 87
topological graph, 236
toroidal butterflies, 216, 218
toroidal butterfly graph, 219
toroidal meshes, 216, 218
totally dual integral system (TDI),
233-234
totally unimodular, 77
tournament problem, 248
transitively orientable, 217
transpose of the vector, 4
trap handling heuristics, 431-432
trap search, 426
trap, 425
trapezoid diagram, 216
trapezoid graphs, 216
Trapping-Times, 438
Traveling Salesman Problem, 32,
278-281
tree search, 22
tree-level search space model, 425-
426
triangulated graphs, 15, 311
trival transportation problem, 103
TSAT, 511
tunneling heuristic, 431, 433-435
Turán theorem, 8
Tutte matrix, 79
two-dimensional Banach space, 574
U-Solve(F) algorithm, 502-504
UC(F) algorithm, 495
UCL(F) algorithm, 497
UHF taboo problem, 308
unconstrained, nonlinear optimiza-
tion algorithms, 393
unifying theory for FAP, 308

- unimodular graphs, 15
- UniSAT model, 443-445
- UniSAT, 388-389
- UniSAT4 model, 444
- UniSAT5 model, 444
- UniSAT7 model, 444
- unit ball, 585
- unit clause backtracking, 410
- unit clause rule, 467, 494, 501
- unit clause, 467
- unit disk graph, 314
- unit resolution, 467
- unsat clause, 438
- unsat state, 437
- unweighted feedback vertex set problem (UFVS), 213
- unweighted performance ratio, 222
- updating problem, 246
- UR algorithm, 468
- valid graph, 223
- valley, 425
- valued algorithms, 597
- variable domain, 484
- Variable Harmonic-Fit (VH), 185
- vertex 3
- vertex cover problem, 13
- vertex cover, 3, 82
- vertex order, 344
- vertex ordering, 313
- vertex saturation, 367
- vertex sequence method, 19
- vertex set, 3
- vertices, 310
- violated set, 229
- VIOLATION, 229
- w-queen problem, 421
- weak cancellation rule, 118
- weak Monge property, 85
- weakly γ -perfect graph, 312
- weakly divisible, 183
- weakly triangulated graphs, 15
- weak_flip(), 438
- weight barycenter, 4
- weight vector, 3
- weighted characteristic vector, 4
- weighted clique number, 3
- weighted n-queen model, 421
- weighted relational structures, 46
- weighting function, 159-161
- Worst out heuristic, 28
- Worst-Fit (WF), 159
- WSAT, 510-511
- zero-cover, 82-83