Studies in Computational Intelligence 114

Christian Blum
Maria José Blesa Aguilera
Andrea Roli · Michael Sampels
(Eds.)

Hybrid Metaheuristics

An Emerging Approach to Optimization

Springer

Christian Blum, Maria José Blesa Aguilera, Andrea Roli and Michael Sampels (Eds.)

Hybrid Metaheuristics

Studies in Computational Intelligence, Volume 114

Editor-in-chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 93. Manolis Wallace, Marios Angelides and Phivos Mylonas (Eds.)
Advances in Semantic Media Adaptation and Personalization, 2008
ISBN 978-3-540-76359-8

Vol. 94. Arpad Kelemen, Ajith Abraham and Yuehui Chen (Eds.)
Computational Intelligence in Bioinformatics, 2008
ISBN 978-3-540-76802-9

Vol. 95. Radu Dogaru
Systematic Design for Emergence in Cellular Nonlinear Networks, 2008
ISBN 978-3-540-76800-5

Vol. 96. Aboul-Ella Hassanien, Ajith Abraham and Janusz Kacprzyk (Eds.)
Computational Intelligence in Multimedia Processing: Recent Advances, 2008
ISBN 978-3-540-76826-5

Vol. 97. Gloria Phillips-Wren, Nikhil Ichalkaranje and Lakhmi C. Jain (Eds.)
Intelligent Decision Making: An AI-Based Approach, 2008
ISBN 978-3-540-76829-9

Vol. 98. Ashish Ghosh, Satchidananda Dehuri and Susmita Ghosh (Eds.)
Multi-Objective Evolutionary Algorithms for Knowledge Discovery from Databases, 2008
ISBN 978-3-540-77466-2
Vol. 99. George Meghabghab and Abraham Kandel
Search Engines, Link Analysis, and User's Web Behavior, 2008
ISBN 978-3-540-77468-6

Vol. 100. Anthony Brabazon and Michael O'Neill (Eds.)
Natural Computing in Computational Finance, 2008
ISBN 978-3-540-77476-1

Vol. 101. Michael Granitzer, Mathias Lux and Marc Spaniol (Eds.)
Multimedia Semantics - The Role of Metadata, 2008
ISBN 978-3-540-77472-3

Vol. 102. Carlos Cotta, Simeon Reich, Robert Schaefer and Antoni Ligeza (Eds.)
Knowledge-Driven Computing, 2008
ISBN 978-3-540-77474-7

Vol. 103. Devendra K. Chaturvedi
Soft Computing Techniques and its Applications in Electrical Engineering, 2008
ISBN 978-3-540-77480-8

Vol. 104. Maria Virvou and Lakhmi C. Jain (Eds.)
Intelligent Interactive Systems in Knowledge-Based Environment, 2008
ISBN 978-3-540-77470-9

Vol. 105. Wolfgang Guenthner
Enhancing Cognitive Assistance Systems with Inertial Measurement Units, 2008
ISBN 978-3-540-76996-5

Vol. 106. Jacqueline Jarvis, Dennis Jarvis, Ralph Rönnquist and Lakhmi C. Jain (Eds.)
Holonic Execution: A BDI Approach, 2008
ISBN 978-3-540-77478-5

Vol. 107. Margarita Sordo, Sachin Vaidya and Lakhmi C. Jain (Eds.)
Advanced Computational Intelligence Paradigms in Healthcare - 3, 2008
ISBN 978-3-540-77661-1

Vol. 108. Vito Trianni
Evolutionary Swarm Robotics, 2008
ISBN 978-3-540-77611-6

Vol. 109. Panagiotis Chountas, Ilias Petrounias and Janusz Kacprzyk (Eds.)
Intelligent Techniques and Tools for Novel System Architectures, 2008
ISBN 978-3-540-77621-5

Vol. 110. Makoto Yokoo, Takayuki Ito, Minjie Zhang, Juhnyoung Lee and Tokuro Matsuo (Eds.)
Electronic Commerce, 2008
ISBN 978-3-540-77808-0

Vol. 111. David Elmakias (Ed.)
New Computational Methods in Power System Reliability, 2008
ISBN 978-3-540-77810-3

Vol. 112. Edgar N. Sanchez, Alma Y. Alanís and Alexander G. Loukianov
Discrete-Time High Order Neural Control: Trained with Kalman Filtering, 2008
ISBN 978-3-540-78288-9

Vol. 113. Gemma Bel-Enguix, M. Dolores Jimenez-Lopez and Carlos Martín-Vide (Eds.)
New Developments in Formal Languages and Applications, 2008
ISBN 978-3-540-78290-2

Vol. 114. Christian Blum, Maria José Blesa Aguilera, Andrea Roli and Michael Sampels (Eds.)
Hybrid Metaheuristics, 2008
ISBN 978-3-540-78294-0

Christian Blum
Maria José Blesa Aguilera
Andrea Roli
Michael Sampels
(Eds.)

Hybrid Metaheuristics

An Emerging Approach to Optimization

Springer

Dr. Christian Blum
ALBCOM
Dept. Llenguatges i Sistemes Informátics
Universitat Politècnica de Catalunya
Jordi Girona 1-3 Omega 112,
Campus Nord
E-08034 Barcelona
Spain
cblum@lsi.upc.edu

Dr. Andrea Roli
DEIS, Campus of Cesena
Alma Mater Studiorum
Università di Bologna
Via Venezia 52
I-47023 Cesena
Italy
andrea.roli@unibo.it

Dr. Maria José Blesa Aguilera
ALBCOM
Dept. Llenguatges i Sistemes Informátics
Universitat Politècnica de Catalunya
Jordi Girona 1-3 Omega 213,
Campus Nord
E-08034 Barcelona
Spain
mjblesa@lsi.upc.edu

Dr. Michael Sampels
IRIDIA
Université Libre de Bruxelles
Avenue Franklin Roosevelt 50, CP 194/6
B-1050 Brussels
Belgium
msampels@ulb.ac.be

ISBN 978-3-540-78294-0 e-ISBN 978-3-540-78295-7

Studies in Computational Intelligence ISSN 1860-949X

Library of Congress Control Number: 2008921710

Cover design: Deblik, Berlin, Germany

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

For my parents Maria and Dieter
(Christian Blum)

For Christian and Marc
(María J. Blesa)

For Elisabetta, Raffaella and Paolo
(Andrea Roli)

For Astrid, Julian and Carlotta
(Michael Sampels)

Preface

When facing complex new optimization problems, it is very natural to use rules of thumb, common sense, trial and error, which are called heuristics, in order to find possible answers. Such approaches are, at first sight, quite different from rigorous scientific approaches, which are usually based on characterizations, deductions, hypotheses and experiments. It is common knowledge that many heuristic criteria and strategies, which are used to find good solutions for particular problems, share common aspects and are often independent of the problems themselves.

In the computer science and artificial intelligence community, the term *metaheuristic* was created and is now well accepted for general techniques which are not specific to a particular problem. Genetic and evolutionary algorithms, tabu search, simulated annealing, iterated local search, ant colony optimization, scatter search, etc. are typical representatives of this generic term. Research in metaheuristics has been very active during the last decades, which is easy to understand, when looking at the wide spectrum of fascinating problems that have been successfully tackled and the beauty of the techniques, many of them inspired by nature. Even though many combinatorial optimization problems are very hard to solve optimally, the quality of the results obtained by somewhat unsophisticated metaheuristics is often impressive.

This success record has motivated researchers to focus on why a given metaheuristic is successful, on which problem instance characteristics should be exploited and on which problem model is best for the metaheuristic of choice. Investigations on theoretical aspects have begun, and formal theories of the working of metaheuristics are being developed. Questions as to which metaheuristic is best for a given problem used to be quite common and, more prosaically, often led to a defensive attitude towards other metaheuristics.

Finally, it also became evident that the concentration on a single metaheuristic is rather restrictive for advancing the state of the art when tackling both academic and practical optimization problems. Examples showed that a skillful combination of metaheuristics with concepts originating from other types of algorithms for optimization can lead to more efficient behavior and greater flexibility. For example, the incorporation of typical operations research (OR) techniques, such as mathematical programming, into metaheuristics may be beneficial. Also, the combination of metaheuristics with other techniques known from artificial intelligence (AI), such as constraint programming and data mining, can be fruitful. Nowadays, such a combination of one metaheuristic with components from other metaheuristics or with techniques from AI and OR techniques is called a *hybrid metaheuristic.*

The lack of a precise definition of the term hybrid metaheuristics is sometimes subject to criticism. On the contrary, we believe that this relatively open definition is helpful, because strict borderlines between related fields of research often block creative research directions. A vital research community needs new ideas and creativity, not overly strict definitions and limitations.

In 2004, we founded with the First International Workshop on Hybrid Metaheuristics (HM 2004) a series of annual workshops. These workshops have developed into a forum for researchers who direct their work towards hybrid algorithms that go beyond the scope of single metaheuristics. The growing interest in these workshops is an indication that questions regarding the proper integration of different algorithmic components and the adequate analysis of results can now emerge from the shadows. With this backdrop, it becomes evident that hybrid metaheuristics is now a part of experimental science and that its strong interdisciplinarity supports cooperation between researchers with different expertise.

In the light of the above, we feel that it is now time for a textbook on hybrid metaheuristics, presenting the most important achievements and developments in this domain. We have invited key experts in the field to supply chapters with the objective of providing an introduction to the themes of hybrid metaheuristics and discussing associated theoretical aspects or applications. We hope that, by reading this book, either researchers or students will have an easy entry point to this fascinating field and will get a clear overview of its research directions.

Barcelona, Bologna, Brussels
November 2007

Christian Blum
Maria José Blesa Aguilera
Andrea Roli
Michael Sampels

Contents

Hybrid Metaheuristics: An Introduction

Christian Blum[1] and Andrea Roli[2]

[1] ALBCOM research group
Universitat Politècnica de Catalunya, Barcelona, Spain
cblum@lsi.upc.edu

[2] DEIS, Campus of Cesena
Alma Mater Studiorum Università di Bologna, Bologna, Italy
andrea.roli@unibo.it

Summary. In many real life settings, high quality solutions to hard optimization problems such as flight scheduling or load balancing in telecommunication networks are required in a short amount of time. Due to the practical importance of optimization problems for industry and science, many algorithms to tackle them have been developed. One important class of such algorithms are metaheuristics. The field of metaheuristic research has enjoyed a considerable popularity in the last decades. In this introductory chapter we first provide a general overview on metaheuristics. Then we turn towards a new and highly successful branch of metaheuristic research, namely the hybridization of metaheuristics with algorithmic components originating from other techniques for optimization. The chapter ends with an outline of the remaining book chapters.

1 Introduction

In the context of combinatorial optimization (CO), algorithms can be classified as either *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time (see [80, 76]). Yet, for CO problems that are $\mathcal{NP}$-hard [42], no polynomial time algorithm exists, assuming that $\mathcal{P} \neq \mathcal{NP}$. Therefore, complete methods might need exponential computation time in the worst-case. This often leads to computation times too high for practical purposes. In approximate methods such as metaheuristics we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. Thus, the use of metaheuristics has received more and more attention in the last 30 years. This was also the case in continuous optimization; due to other reasons: Metaheuristics are usually easier to implement than classical gradient-based techniques. Moreover, metaheuristics do not require gradient information. This is convenient for optimization problems where the objective function is only implicitly given (e.g.,

C. Blum and A. Roli: *Hybrid Metaheuristics: An Introduction*, Studies in Computational Intelligence (SCI) **114**, 1–30 (2008)
www.springerlink.com

when objective function values are obtained by simulation), or where the objective function is not differentiable.

The first two decades of research on metaheuristics were characterized by the application of rather standard metaheuristics. However, in recent years it has become evident that the concentration on a sole metaheuristic is rather restrictive. A skilled combination of a metaheuristic with other optimization techniques, a so called *hybrid metaheuristic*, can provide a more efficient behavior and a higher flexibility when dealing with real-world and large-scale problems. This can be achieved, for instance, by combining the complementary strengths of metaheuristics on one side and of complete methods such as branch & bound techniques or mathematical programming on the other side. In general, hybrid metaheuristic approaches can be classified as either *collaborative combinations* or *integrative combinations* (see [85, 86]). Collaborative combinations are based on the exchange of information between several optimization techniques run sequentially (or in parallel). This kind of combination is more related to cooperative and parallel search and we forward the interested reader to the specific literature on the subject [55, 24, 105, 98, 19, 18, 3]. Most contributions of this book deal with interesting and representative cases of integrative combinations.

In this introductory chapter we first aim at giving an overview over some of the most important metaheuristics. Then we deal with the hybridization of metaheuristics with other techniques for optimization. Finally, we shortly outline the books' contents, with the aim of providing a general guidance to the reader.

2 Basics

An optimization problem $\mathcal{P}$ can be described as a triple $(\mathcal{S}, \Omega, f)$, where

1. $\mathcal{S}$ is the search space defined over a finite set of decision variables X_i, $i = 1, \ldots, n$. In case these variables have discrete domains we deal with discrete optimization (or combinatorial optimization), and in case of continuous domains $\mathcal{P}$ is called a continuous optimization problem. Mixed variable problems also exist. Ω is a set of constraints among the variables;
2. $f : \mathcal{S} \rightarrow I\!R^+$ is the objective function that assigns a positive cost value to each element (or solution) of $\mathcal{S}$.

The goal is to find a solution $s \in \mathcal{S}$ such that $f(s) \leq f(s')$, $\forall\ s' \in \mathcal{S}$ (in case we want to minimize the objective function), or $f(s) \geq f(s')$, $\forall\ s' \in \mathcal{S}$ (in case the objective function must be maximized). In real-life problems the goal is often to optimize several objective functions at the same time. This form of optimization is labelled multi-objective optimization.

Approximate methods are generally based on two basic principles: *constructive heuristics* and *local search methods*.

2.1 Constructive Heuristics

Constructive heuristics are typically the fastest approximate methods. They generate solutions from scratch by adding opportunely defined solution components to an initially empty partial solution. This is done until a solution is complete or other stopping criteria are satisfied. A well-known class of constructive heuristics are *greedy heuristics*. They make use of a weighting function that assigns at each construction step a positive weight to each feasible solution component. The solution component with the highest weight is chosen at each construction step to extend the current partial solution. An example of a greedy heuristic is the *nearest neighbor heuristic* for the famous traveling salesman problem (TSP) [57].

2.2 Local Search Methods

Constructive heuristics are often very fast, yet they often return solutions of inferior quality when compared to local search algorithms. Local search algorithms start from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution, where the neighborhood is formally defined as follows.

Definition 1. *A* **neighborhood structure** *is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s. Often, neighborhood structures are implicitly defined by specifying the changes that must be applied to a solution s in order to generate all its neighbors. The application of such an operator that produces a neighbor $s' \in \mathcal{N}(s)$ of a solution s is commonly called a* **move***.*

A neighborhood structure together with a problem instance define the topology of a so-called search (or fitness) landscape [100, 61, 40]. A search landscape can be visualized as a labelled graph in which the nodes are solutions (labels indicate their objective function value) and arcs represent the neighborhood relation between solutions. A solution $s^* \in \mathcal{S}$ is called a *globally minimal solution*[1] (or global minimum) if for all $s \in \mathcal{S}$ it holds that $f(s^*) \leq f(s)$. The set of all globally minimal solutions is henceforth denoted by $\mathcal{S}^*$. The introduction of a neighborhood structure enables us to additionally define the concept of *locally* minimal solutions.

Definition 2. *A* **locally minimal solution (or local minimum)** *with respect to a neighborhood structure $\mathcal{N}$ is a solution $\hat{s}$ such that $\forall\ s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \leq f(s)$. We call $\hat{s}$ a strict locally minimal solution if $f(\hat{s}) < f(s)\ \forall\ s \in \mathcal{N}(\hat{s})$.*

[1] Without loss of generality, in the remainder of this chapter we restrict the discussion to minimization problems.

Algorithm 1 Iterative improvement local search

```
$s \leftarrow$ GenerateInitialSolution()
**while** $\exists\ s' \in \mathcal{N}(s)$ such that $f(s') < f(s)$ **do**
    $s \leftarrow$ ChooseImprovingNeighbor($\mathcal{N}(s)$)
**end while**
```

The most basic local search method is usually called *iterative improvement local search*, since each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it has reached a local minimum. The high level algorithm is sketched in Alg. 1.

Function ChooseImprovingNeighbor($\mathcal{N}(s)$) can be implemented in several ways. In the following we present the two most typical ones. The first way is called *first-improvement.* A first-improvement function scans the neighborhood $\mathcal{N}(s)$ and returns the first solution that is better than s. In contrast, a *best-improvement* function exhaustively explores the neighborhood and returns one of the solutions with the lowest objective function value. An iterative improvement procedure that uses a first-improvement function is called *first-improvement local search*, respectively *best-improvement local search* (or steepest descent local search) in the case of a best-improvement function. Both methods stop at local minima. Therefore, their performance strongly depends on the definition of the neighborhood structure $\mathcal{N}$. A deterministic iterative improvement local search algorithm partitions the search space $\mathcal{S}$ into so-called *basins of attraction* of local minima [84, 92]. The basin of attraction of a local minimum $\hat{s} \in \mathcal{S}$ is the set of all solutions s for which the search terminates in $\hat{s}$ when started from the initial solution s.[2]

2.3 Metaheuristics

In the 70ies, a new kind of approximate algorithm has emerged which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called *metaheuristics*.[3] The term *metaheuristic*, first introduced in [45], derives from the composition of two Greek words. *Heuristic* derives from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means "to find", while the suffix *meta* means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called *modern heuristics* [88].

This class of algorithms includes[4]—but is not restricted to—ant colony optimization (ant colony optimization), evolutionary computation (EC) including

[2] The definition can also be extended to the case of stochastic local search algorithms.

[3] The increasing importance of metaheuristics is underlined by the biannual Metaheuristics International Conference (MIC). The 7th was held in Montreal in June 2007 (*www.crt.umontreal.ca/mic2007/*).

[4] In alphabetical order.

genetic algorithms (GAs), iterated local search (ILS), simulated annealing (SA), and tabu search (TS). Due to the generality of the metaheuristic concept it is hardly possible to give a precise definition of what a metaheuristic exactly is. In the following we quote some characterizations that appeared in the literature:

"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." I. Osman and G. Laporte in [78].

"A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method." S. Voß et al. in [109].

"Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local minima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form." Stützle in [101].

"A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem." Metaheuristics Network at [26].

In short, we may characterize metaheuristics as high level strategies for exploring search spaces by using different methods. Of great importance for the functioning of a metaheuristic are the concepts called *diversification* and *intensification.* The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. Each metaheuristic application is characterized by a balance between diversification and intensification. This is important, on one side to quickly identify regions in the search space with high quality solutions, and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high quality solutions.

There are different ways to classify and describe metaheuristic algorithms, each of them being the result of a specific viewpoint. For example, we might classify metaheuristics as *nature-inspired metaheuristics vs. non-nature inspired metaheuristics.* This classification is based on the origins of the different algorithms. There are nature-inspired algorithms, such as evolutionary computation and ant colony optimization, and non nature-inspired ones such as tabu search and iterated local search. We might also classify metaheuristics as *memory-based vs. memory-less methods.* This classification scheme refers to the use metaheuristics make of the search history, that is, whether they use memory or not. Memory-less algorithms, for example, perform a Markov process, as the information they exclusively use to determine the next action is the current state of the search process. The use of memory is nowadays recognized as one of the fundamental elements of a powerful metaheuristic. Finally, metaheuristics may also be classified into methods that perform a *single point vs. population-based search.* This classification refers to the number of solutions used by a metaheuristic at any time. Generally, algorithms that work on a single solution at any time are referred to as *trajectory methods.* They comprise all metaheuristics that are based on local search, such as tabu search, iterated local search and variable neighborhood search. They all share the property that the search process describes a trajectory in the search space. Population-based metaheuristics, on the contrary, either perform search processes which can be described as the evolution of a set of points in the search space (as for example in evolutionary computation), or they perform search processes which can be described as the evolution of a probability distribution over the search space (as for example in ant colony optimization).

3 Overview on Important Metaheuristics

In the following, we outline the main principles of some of the most important metaheuristics. However, an introduction to such a vast research area has to focus on certain aspects and therefore has unfortunately to neglect other aspects. We refer the interested reader to [114, 49, 33, 93, 110] for works that deal also with other aspects such as, for example, software libraries. Furthermore, we refer to [11, 57] for a more detailed introduction to metaheuristics.

3.1 Metaheuristics Based on Local Search

The performance of simple iterative improvement local search procedures (see Sect. 2.2) is in general unsatisfactory. The quality of the obtained local minimum heavily depends on the starting point for the local search process. As the basin of attraction of a global minimum is generally not known, iterative improvement local search might end up in a poor quality local minimum. A first simple strategy of extending iterative improvement local search consists in the iterative application of the local search starting at each iteration from a different starting point, which may be randomly generated. For this type of multi-start local search it is sometimes possible to obtain theoretical performance guarantees. However, they are usually still far from being satisfactory (see, for example, [94]).

Therefore, several metaheuristic techniques have been developed with the aim of adding an *exploration* component to iterative improvement local search. This exploration component is responsible for guiding the exporation of the search space in the search for better and better local minima. Obviously, these algorithms need termination criteria other than simply reaching a local minimum. Commonly used termination criteria are a maximum CPU time, a maximum number of iterations, a solution s of sufficient quality is found, or a maximum number of iterations without improvement. In the following we present some of the most important local-search based metaheuristics.

Simulated Annealing

Simulated annealing (SA) is commonly said to be the oldest among the metaheuristics and was one of the first algorithms that had an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (see the Metropolis algorithm [70]). The idea of SA is inspired by the annealing process of metal and glass, which assume a low energy configuration when first heated up and then cooled down sufficiently slowly. SA was first presented as a search algorithm for CO problems in [63] and [14]. The fundamental idea is to allow moves to solutions with objective function values that are worse than the objective function value of the current solution. This kind of move is often called uphill move.

The algorithmic framework of SA is described in Alg. 2. It works as follows. The algorithm starts by generating an initial solution in function GenerateInitialSolution(). The initial solution may be randomly or heuristically constructed. Then, the initial temperature value is determined in function SetInitialTemperature() such that the probability for an uphill move is quite high at the start of the algorithm. At each iteration a solution $s' \in \mathcal{N}(s)$ is randomly chosen in function PickNeighborAtRandom($\mathcal{N}(s)$). If s' is better than s (i.e., has a lower objective function value), then s' is accepted as new current solution. Otherwise, if the move from s to s' is an uphill move, s' is accepted with a probability which is a function of a temperature parameter T_k and

Algorithm 2 Simulated annealing (SA)

```
$s \leftarrow$ GenerateInitialSolution()
$k \leftarrow 0$
$T_k \leftarrow$ SetInitialTemperature()
while termination conditions not met do
    $s' \leftarrow$ PickNeighborAtRandom($\mathcal{N}(s)$)
    if $(f(s') < f(s))$ then
        $s \leftarrow s'$        {$s'$ replaces $s$}
    else
        Accept $s'$ as new solution with probability $\mathbf{p}(s' \mid T_k, s)$  {see Equation 1}
    end if
    $k \leftarrow k + 1$
    $T_k \leftarrow$ AdaptTemperature()
end while
```

$f(s') - f(s)$. Usually this probability is computed following the Boltzmann distribution:

$$\mathbf{p}(s' \mid T_k, s) = e^{-\frac{f(s')-f(s)}{T_k}} \quad . \tag{1}$$

The temperature T_k is adapted at each iteration according to a *cooling schedule* (or cooling scheme) in function AdaptTemperature(). The cooling schedule defines the value of T_k at each iteration k. The choice of an appropriate cooling schedule is crucial for the performance of the algorithm. At the beginning of the search the probability of accepting uphill moves should be high. Then, this probability should be gradually decreased during the search. Note that this is not necessarily done in a monotonic fashion.

The dynamic process described by basic SA is a *Markov chain* [30], because the choice of the next solution exclusively depends on the current solution, that is, the algorithm does not use memory. Nevertheless, the use of memory can be beneficial for SA approaches (see for example [15]). Interestingly, theoretical results on non-homogeneous Markov chains [1] state that under particular conditions on the cooling schedule, SA converges in probability to a global minimum for $k \to \infty$.

For representative applications of SA we refer the interested reader to [17, 107, 2, 59, 34]. Interesting variants of SA are Threshold Accepting, respectively the Great Deluge Algorithm [29, 28], and Extremal Optimization [12].

Tabu Search

The basic ideas of tabu search (TS) were introduced in [45], based on earlier ideas formulated in [44].[5] A description of the method and its concepts can be found in [47].

[5] Related ideas were labelled steepest ascent/mildest descent method in [50].

Algorithm 3 Tabu search (TS)

```
s ← GenerateInitialSolution()
InitializeTabuLists($TL_1, \ldots, TL_r$)
while termination conditions not met do
    $\mathcal{N}_a(s) \leftarrow \{s' \in \mathcal{N}(s) \mid s'$ does not violate a tabu condition, or it satisfies
        at least one aspiration condition$\}$
    $s' \leftarrow \mathsf{argmin}\{f(s'') \mid s'' \in \mathcal{N}_a(s)\}$
    UpdateTabuLists($TL_1, \ldots, TL_r, s, s'$)
    $s \leftarrow s'$        {i.e., $s'$ replaces $s$}
end while
```

The basic idea of TS is the explicit use of search history, both to escape from local minima and to implement a strategy for exploring the search space. A basic TS algorithm (see Alg. 3) uses *short term memory* in the form of so-called *tabu lists* to escape from local minima and to avoid cycles.[6] In standard TS algorithms tabu lists are implemented in a FIFO (first in first out) manner. The tabu lists generally store features of recently visited solutions. The algorithm may work with a different tabu list for each type of considered solution feature. At the start of the algorithm the tabu lists are initialized as empty lists in function InitializeTabuLists($TL_1, \ldots, TL_r$). For performing a move, the algorithm first determines those solutions from the neighborhood $\mathcal{N}(s)$ of the current solution s that contain solution features currently to be found in the tabu lists. These solutions are said to violate the tabu conditions. They are excluded from the neighborhood, resulting in a restricted set of neighbors $\mathcal{N}_a(s)$. However, note that storing only features of solutions allows the possibility that unvisited solutions of high quality are excluded from the set of neighbors. To overcome this problem, *aspiration criteria* are defined which allow to include a solution in the restricted set of neighbors even though it violates a tabu condition. The most commonly used aspiration criterion applies to solutions which are better than the best solution found so far. At each iteration the best solution s' from $\mathcal{N}_a(s)$ is chosen as the new current solution. Furthermore, in procedure UpdateTabuLists($TL_1, \ldots, TL_r, s, s'$) the corresponding features of this solution are added to the tabu lists and—in case the tabu lists have reached their maximally allowed length—the oldest solution features are deleted. The algorithm stops when a termination condition is met.

In general, the use of a tabu list prevents from returning to recently visited solutions, and may force the search to accept even uphill moves. The length l of a tabu list—known in the literature as the *tabu tenure*—controls the memory of the search process. With small tabu tenures the search will concentrate on small areas of the search space. On the opposite, a large tabu tenure forces the search process to explore larger regions, because it forbids

[6] A cycle is a sequence of moves that constantly repeats itself.

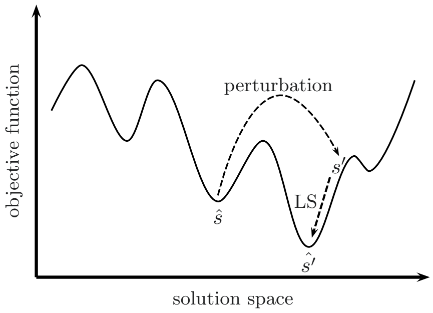


Fig. 1. A desirable ILS step: the current solution $\hat{s}$ is perturbed, then local search is applied to the perturbed solution s' and a new (even better) local minimum $\hat{s'}$ is found.

revisiting a higher number of solutions. The tabu tenure can be varied during the search, leading to more robust algorithms. An example can be found in [103], where the tabu tenure is periodically reinitialized at random from the interval $[l_{min}, l_{max}]$. A more advanced use of a dynamic tabu tenure is presented in [7, 6], where the tabu tenure is increased in case of evidence for repetitions of solutions (thus a higher diversification is needed), while it is decreased in case of no improvements (thus intensification should be boosted). More advanced ways of applying dynamic tabu tenures are described in [46].

The interested reader can find representative applications of TS in [103, 6, 21, 77, 43]. Further references to applications can be found in [47].

Iterated Local Search

Iterated local search (ILS) [101, 65, 64, 68] is a metaheuristic that is based on a simple but effective concept. Instead of repeatedly applying an iterative improvement local search to randomly generated starting solutions, an ILS algorithm produces the starting solution for the next iteration by perturbing the local minimum obtained by the previous application of iterative improvement local search. The hope is that the perturbation mechanism produces a solution that is located in the basin of attraction of a local minimum that is better than the current solution, and that is close to the current solution. Fig. 1 shows such a situtation graphically.

The pseudo-code of ILS is shown in Alg. 4. It works as follows. First, an initial solution is generated in function GenerateInitialSolution(). This solution is subsequently improved by the application of a local search method in function LocalSearch(s). The construction of initial solutions should be fast (computationally not expensive), and—if possible—initial solutions should be a good starting point for local search. The fastest way of producing an initial solution is often to generate it at random. Another possibility is to use

Algorithm 4 Iterated local search (ILS)

```
$s \leftarrow$ GenerateInitialSolution()
$\hat{s} \leftarrow$ LocalSearch($s$)
while termination conditions not met do
    $s' \leftarrow$ Perturbation($\hat{s}$, $history$)
    $\widehat{s'} \leftarrow$ LocalSearch($s'$)
    $\hat{s} \leftarrow$ ApplyAcceptanceCriterion($\widehat{s'}$, $\hat{s}$, $history$)
end while
```

constructive heuristics such as greedy heuristics. At each iteration, the current solution $\hat{s}$ is perturbed in function Perturbation($\hat{s}$,$history$), resulting in a perturbed solution s'. The perturbation is usually non-deterministic in order to avoid cycling. The importance of the perturbation mechanism is obvious: a perturbation that is not strong enough might not enable the algorithm to escape from the basin of attraction of the current solution. On the other side, a perturbation that is too strong would make the algorithm similar to a random restart local search. The requirement on the perturbation method is to produce a starting point for local search such that a local minimum different from the current solution is reached. However, this new local minimum should be *closer* to the current solution than a local minimum produced by the application of the local search to a randomly generated solution. After the application of local search to the perturbed solution, the resulting solution $\widehat{s'}$ may either be accepted as new current solution, or not. This is decided in function ApplyAcceptanceCriterion($\widehat{s'}$, $\hat{s}$, $history$). Two extreme examples are (1) accepting the new local minimum only in case of improvement and (2) always accepting the new solution. Inbetween, there are several possibilities. For example, it is possible to adopt an acceptance criterion that is similar to the one of simulated annealing, like non-monotonic cooling schedules which exploit the history of the search process. For example, when the recent history of the search process indicates that intensification seems no longer effective, a diversification phase is needed and the solution reached after the local search phase is always accepted, or the probability to accept it is increased.

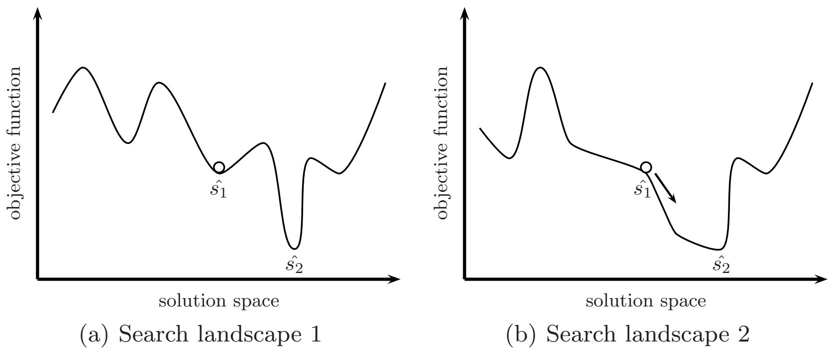
For examples of successful applications of ILS we refer the interested reader to [67, 60, 65, 22]. References to other applications can be found in [65].

Other Metaheuristics Based on Local Search

Besides the metaheuristics outlined above, there are some other ones that are based on local search. In the following we present shortly their basic principles.

Variable neighborhood search (VNS).

This metaheuristic was proposed in [52, 53]. The main idea of VNS is based on the fact that a change of the neighborhood structure changes the shape of the



(a) Search landscape 1

(b) Search landscape 2

Fig. 2. Two search landscapes defined by two different neighborhood structures. On the landscape that is shown in (a), the best-improvement local search stops at $\hat{s_1}$, while it proceeds till a better local minimum $\hat{s_2}$ on the landscape that is shown in (b).

search landscape. In fact, a local minimum with respect to a neighborhood function $\mathcal{N}_1$ is not necessarily a local minimum with respect to a different neighborhood function $\mathcal{N}_2$ (see Fig. 2 for an example).

Based on this observation, the main idea of VNS is to define more than one neighborhood structure, and to swap between different neighborhood structures in a strategic way during the search process. Observe that the process of changing neighborhoods (for example, in case of no improvements) corresponds to a diversification of the search. The effectiveness of VNS can be explained by the fact that a "bad" place on the search landscape given by a certain neighborhood structure could be a "good" place on the search landscape given by another neighborhood structure.[7] Moreover, a solution that is locally optimal with respect to a neighborhood is probably not locally optimal with respect to another neighborhood. Concerning applications of VNS we refer the interested reader to [51, 91, 113, 106]. More references can be found in [53].

Guided local search (GLS).

The strategy applied by GLS (see [112, 111]) for exploring the search space is conceptually very different to the strategies that are employed by the other local search based metaheuristics. This strategy consists in dynamically changing the objective function by *penalizing* solution features that occur frequently in visited solutions. The use of penalties aims at increasing the objective function value of solutions that contain these features. This change in the objective function results in a change of the search landscape. The aim is to make the current local minimum gradually "less desirable" over time in order to be

[7] A "good" place in the search space is an area from which a good local minimum can be reached.

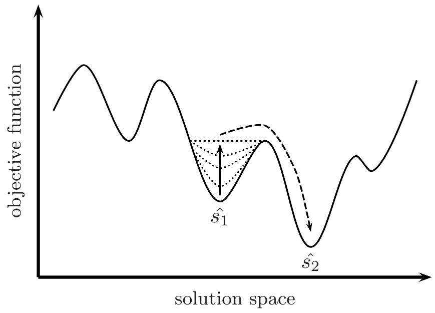


Fig. 3. Basic GLS idea: escaping from a valley in the landscape by increasing the objective function value of its solutions.

able to move to other areas of the search space. A pictorial description of this idea is given in Fig. 3. The interested reader is referred to [73, 62, 112] for applications of GLS.

Greedy randomized adaptive search procedure (GRASP).

This metaheuristic (see [31, 81]) is a simple technique that combines constructive heuristics with local search. At each iteration, a solution to the tackled problem is constructed in a randomized way. Subsequently the constructed solution is improved by the application of a local search algorithm. GRASP is a metaheuristic that does not make use of the search history. This is one of the reasons why GRASP is often outperformed by other metaheuristics. However, due to its simplicity, it is generally very fast and it is able to produce quite good solutions in a very short amount of computation time. It can be effective if—at least—two conditions are satisfied: (1) the solution construction mechanism samples the most promising regions of the search space, and (2) the solutions constructed by the constructive heuristic belong to basins of attraction of different local minima. Representative applications of GRASP include [8, 90, 82]. A detailed and annotated bibliography references many more applications [32].

3.2 Population-Based Metaheuristics

Population-based metaheuristics deal at each algorithm iteration with a set of solutions rather than with a single solution. From this set of solutions the population of the next iteration is produced by the application of certain operators. Population-based metaheuristics provide a natural, intrinsic way for the exploration of the search space. Yet, the final performance strongly depends on the way the population is manipulated. The most studied population-based methods are evolutionary computation (EC) and ant colony optimization (ant colony optimization). In EC algorithms, a population of individuals is modified by recombination and mutation operators, and in ant colony optimization a colony of artificial ants is used to construct solutions guided by the pheromone trails and heuristic information.

Ant Colony Optimization

Ant colony optimization (ant colony optimization) [27, 9] is a metaheuristic approach that was inspired by the foraging behavior of real ants. Their way of foraging—as described by Deneubourg et al. in [23]—enables ants to find shortest paths between food sources and their nest. Initially, ants explore the area surrounding their nest in a random manner. As soon as an ant finds a source of food, it evaluates quantity and quality of the food and carries some of this food to the nest. During the return trip, the ant deposits a chemical pheromone trail on the ground. The quantity of pheromone deposited, which may depend on the quantity and quality of the food, will guide other ants to the food source. The indirect communication between the ants via the pheromone trails allows them to find shortest paths between their nest and food sources. This functionality of real ant colonies is exploited in artificial ant colonies in order to solve hard optimization problems.

In ant colony optimization algorithms the chemical pheromone trails are simulated via a parametrized probabilistic model that is called the *pheromone model*. It consists of a set of model parameters whose values are called the *pheromone values*. These values act as the memory that keeps track of the search process. The basic ingredient of ant colony optimization algorithms is a constructive heuristic that is used for probabilistically constructing solutions using the pheromone values. In general, the ant colony optimization approach attempts to solve a CO problem by iterating the following two steps:

- Solutions are constructed using a pheromone model, that is, a parametrized probability distribution over the solution space.
- The constructed solutions and possibly solutions that were constructed in earlier iterations are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

A very general pseudo-code for the ant colony optimization metaheuristic is shown in Alg. 5. It consists of three algorithmic components that are gathered in the ScheduleActivities construct. At each iteration, the algorithm probabilistically generates a number of solutions to the tackled problem in function AntBasedSolutionConstruction() by assembling them from a set of solution components. Subsequently, the pheromone values are modified in function PheromoneUpdate(). A standard pheromone update consists of two parts. First, a *pheromone evaporation*, which uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas in the search space. Then, one or more solutions from the current and/or from earlier iterations are used to increase the values of

Algorithm 5 Ant colony optimization (ant colony optimization)

```
while termination conditions not met do
   ScheduleActivities
      AntBasedSolutionConstruction()
      PheromoneUpdate()
      DaemonActions()        {optional}
   end ScheduleActivities
end while
```

pheromone trail parameters on solution components that are part of these solutions. Other types of pheromone update are rather optional and mostly aim at the intensification or the diversification of the search process. Daemon actions (see function DaemonActions()) can be used to implement centralized actions; in contrast to the localized decision making of the solution construction process. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

In general, different versions of ant colony optimization algorithms mostly differ in the way they update the pheromone values. This also holds for two of the currently best-performing ant colony optimization variants in practice, which are Ant Colony System (ACS) [25] and $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [102].

We refer the interested reader to [41, 69, 10, 96] for applications of ACO algorithms. Further references to applications of ant colony optimization can be found in [27].

Evolutionary Computation

Evolutionary computation (EC) can be regarded as a metaphor for building, applying, and studying algorithms based on Darwinian principles of natural selection. The instances of algorithms that are based on evolutionary principles are called evolutionary algorithms (EAs) ([5]). EAs can be characterized as computational models of evolutionary processes. They are inspired by nature's capability to evolve living beings well adapted to their environment. At the core of each EA is a population of individuals. At each algorithm iteration a number of *reproduction* operators is applied to the individuals of the current population to generate the individuals of the population of the next generation. EAs might use operators called *recombination* or *crossover* to recombine two or more individuals to produce new individuals. They also can use *mutation* or *modification* operators which cause a self-adaptation of individuals. The driving force in EAs is the *selection* of individuals based on their *fitness*

Algorithm 6 Evolutionary computation (EC)

```
$P \leftarrow$ GenerateInitialPopulation()
Evaluate($P$)
while termination conditions not met do
    $P' \leftarrow$ Recombine($P$)
    $P'' \leftarrow$ Mutate($P'$)
    Evaluate($P''$)
    $P \leftarrow$ Select($P''$,$P$)
end while
```

(which might be based on the objective function, the result of a simulation experiment, or some other kind of quality measure). Individuals with a higher fitness have a higher probability to be chosen as members of the population of the next generation (or as parents for the generation of new individuals). This corresponds to the principle of *survival of the fittest* in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave the inspiration for EAs.

Alg. 6 shows the basic structure of EC algorithms. In this algorithm, P denotes the population of individuals. A population of offspring is generated by the application of *recombination* and *mutation* operators and the individuals for the next population are *selected* from the union of the old population and the offspring population.

There has been a variety of slightly different EAs proposed over the years. Three different strands of EAs were developed independently of each other over time. These are evolutionary programming (EP) as introduced by Fogel in [38] and Fogel et al. in [39], evolutionary strategies (ES) proposed by Rechenberg in [87] and genetic algorithms (GAs) initiated by Holland in [56] (see [48], [74], [89], and [108] for further references). EP arose from the desire to generate machine intelligence. While EP originally was proposed to operate on discrete representations of finite state machines, most of the present variants are used for continuous optimization problems. The latter also holds for most present variants of ES, whereas GAs are mainly applied to solve discrete problems. More recently, other members of the EA family such as for example genetic programming (GP) and scatter search (SS) were developed. Despite this subdivision into different strands, EAs can be understood from a unified point of view with respect to their main components and the way they explore the search space. Over the years there have been quite a few overviews and surveys about EC methods. Among those are the ones by Bäck [4], by Fogel et al. [36], by Kobler and Hertz [54], by Spears et al. [99], and by Michalewicz and Michalewicz [71]. In [13] a taxonomy of EAs is proposed.

EC algorithms have been applied to most CO problems and optimization problems in general. Recent successes are documented works such as [37, 16, 97]. For an extensive collection of references to EC applications we refer to [5].

4 Hybridization of Metaheuristics

The concept of hybrid metaheuristics has been commonly accepted only in recent years, even if the idea of combining different metaheuristic strategies and algorithms dates back to the 1980s. Today, we can observe a generalized common agreement on the advantage of combining components from different search techniques and the tendency of designing hybrid techniques is widespread in the fields of operations research and artificial intelligence. The consolidated interest around hybrid metaheuristics is also demonstrated by publications on classifications, taxonomies and overviews on the subject [86, 85, 104, 72].

In this book, we adopt the definition of hybrid metaheuristic in the broad sense of integration of a metaheuristic related concept with some other techniques (possibly another metaheuristic). We may distinguish between two categories: the first consists in designing a solver including components from a metaheuristic into another one, while the second combines metaheuristics with other techniques typical of fields such as operations research and artificial intelligence. A prominent representant of the first category is the use of trajectory methods into population based techniques or the use of a specific local search method into a more general trajectory method such as ILS. The second category includes hybrids resulting from the combination of metaheuristics with constraint programming (CP), integer programming (IP), tree-based search methods, data mining techniques, etc. Both categories contain countless instances and an exhaustive description is not possible. Nevertheless, we believe that a brief description of some notable examples could give the flavour of the ideas that characterize these forms of hybridization.

4.1 Component Exchange Among Metaheuristics

One of the most popular ways of metaheuristic hybridization consists in the use of trajectory methods inside population-based methods. Indeed, most of the successful applications of EC and ACO make use of local search procedures and the effectiveness of many memetic algorithms [75] relies indeed on this synergic integration. The reason for that becomes apparent when analyzing the respective strengths of trajectory methods and population-based methods.

The power of population-based methods is certainly to be found in the capability of recombining solutions to obtain new ones. In EC algorithms and scatter search explicit recombinations are implemented by one or more recombination operators. In ACO and EDAs recombination is implicit, because new solutions are generated by using a distribution over the search space which is a function of earlier populations. This enables the search process to perform a guided sampling of the search space, usually resulting in a coarse grained exploration. Therefore, these technique can effectively find promising areas of the search space.

The strength of trajectory methods is rather to be found in the way they explore a promising region in the search space. As in those methods local search is the driving component, a promising area in the search space is searched in a more structured way than in population-based methods. In this way the danger of being close to good solutions but “missing” them is not as high as in population-based methods. More formally, local search techniques efficiently drives the search toward the attractors, i.e., local optima or confined areas of the space in which many local optima are condensed.

In summary, population-based methods are better in identifying promising areas in the search space from which trajectory methods can quickly reach good local optima. Therefore, metaheuristic hybrids that can effectively combine the strengths of both population-based methods and trajectory methods are often very successful.

4.2 Integration of Metaheuristics With AI and OR Techniques

One of the most prominent research directions is the integration of metaheuristics with more classical artificial intelligence and operations research methods, such as CP and branch & bound or other tree search techniques. In the following we outline some of the possible ways of integration.

Metaheuristics and tree search methods can be sequentially applied or they can also be interleaved. For instance, a tree search method can be applied to generate a partial solution which will then be completed by a metaheuristic approach. Alternatively, metaheuristics can be applied to improve a solution generated by a tree-search method.

CP techniques can be used to reduce the search space or the neighborhood to be explored by a local search method. In CP, CO problems are modelled by means of variables, domains[8] and constraints, which can be mathematical (as for example in linear programming) or symbolic (also known as *global* constraints). Constraints encapsulate well defined parts of the problem into sub-problems, thus making it possible to design specialized solving algorithms for sub-problems that occur frequently. Every constraint is associated to a *filtering* algorithm that deletes values from a variable domain that do not contribute to feasible solutions.[9] Metaheuristics (especially trajectory methods) may use CP to efficiently explore the neighborhood of the current solution, instead of simply enumerating the neighbors or randomly sampling the neighborhood. A prominent example of such a kind of integration is Large Neighborhood Search [95] and related techniques. These approaches are effective mainly when the neighborhood to explore is very large, or when problems (such as many real-world problems) have additional constraints (usually called *side constraints*). A detailed overview of the possible ways of integration of CP and metaheuristics can be found in [35].

[8] We restrict the discussion to finite domains.

[9] The filtering could be either *complete* if the remaining domain values are guaranteed to be consistent, or *incomplete* otherwise.

Another possible combination consists in introducing concepts or strategies from either class of algorithms into the other. For example, the concepts of tabu list and aspiration criteria—known from Tabu search—can be used to manage the list of open nodes (i.e., the ones whose child nodes are not yet explored) in a tree search algorithm. Examples of these approaches can be found in [83, 20]. Tree-based search is also successfully integrated into ACO in [10], where beam search [79] is used for solution construction.

Integer and linear programming can be also effectively combined with metaheuristics. For instance, linear programming is often used either to solve a sub-problem or to provide dual information to a metaheuristic in order to select the most promising candidate solution or solution component [58, 66, 10].

The kinds of integration we shortly mentioned belong to the class of *integrative combinations* and are the main topic of this book. The other possible way of integration, called either *collaborative combinations* or also *cooperative search* consists in a loose form of hybridization, in that search is performed by possibly different algorithms that exchange information about states, models, entire sub-problems, solutions or search space characteristics. Typically, cooperative search algorithms consist of the parallel execution of search algorithms with a varying level of communication. The algorithms can be different or they can be instances of the same algorithm working on different models or running with different parameter settings. The algorithms composing a cooperative search system can be all approximate, all complete, or a mix of approximate and complete approaches. This area of research shares many issues with the design of parallel algorithms and we forward the interested reader to the specific literature on the subject [3, 55, 24, 105, 98].

5 Outline of the Book

The contributions collected in this book cover some of the main topics of hybrid metaheuristics. Most of the chapters are devoted to the integration of metaheuristics with other techniques from AI and OR, namely, mathematical programming, constraint programming and various combinations of complete and incomplete search techniques. Two additional chapters complement the collection by giving, respectively, an overview of hybrid metaheuristics for multi-objective problems and introducing multilevel refinement for enhancing the performance of standard metaheuristics. In the following we give an outline of the book by providing a short description of each chapter.

5.1 Chapter 2: Integer Linear Programming and Metaheuristics

The chapter by Raidl and Puchinger provides a comprehensive survey on the integration of metaheuristics and exact techniques such as integer linear programming (ILP), cutting plane and column generation approaches, branch-and-cut, branch-and-price, and branch-and-cut-and-price. The authors point out that metaheuristics and exact approaches can be seen as complementary to a large degree, which makes it natural to combine ideas from both streams. They claim that hybrid optimizers are often significantly more effective in terms of running time and/or solution quality since they benefit from synergy. After discussing a structural classifications of strategies for combining metaheuristics and exact optimization techniques, the authors survey several types of different hybridization approaches, including the following ones:

1. Instead of simple heuristics, metaheuristics can be used for finding high-quality upper bounds within a branch & bound algorithm. Hereby, metaheuristics may be applied for deriving the initial solutions as well as deriving upper bounds for subproblems of the branch & bound tree.
2. Problem relaxations may be used for guiding the search process of a metaheuristic, because an optimal solution for a relaxation of the original problem often indicates in which areas of the original problems search space good or even optimal solutions may be found.
3. Generally, branch & bound algorithms choose the next tree node to be processed by a best-first strategy: choose a node with the smallest lower bound. However, with this strategy, high quality upper bounds are only found late in the search process. More sophisticated concepts aim to intensify branch & bound search—in the style of metaheuristics—in an initial phase to neighborhoods of promising incumbents in order to quickly identify high quality upper bounds.
4. In metaheuristics, candidate solutions are sometimes only indirectly or incompletely represented. In these situtations, an (intelligent) decoding function is needed in order to obtain complete solution to the problem at hand. This is the case, for example, in evolutionary algorithms and ant colony optimization. ILP techniques may successfully be used for the decoding step.
5. In cutting plane and column generation based methods the dynamic separation of cutting planes and the pricing of columns, respectively, is sometimes done by means of metaheuristics in order to speed up the optimization process.

5.2 Chapter 3: Relation Between Complete and Incomplete Search

An in-depth analysis of the relation between complete and incomplete search is the subject of the contribution by Prestwich. The author explores the boundaries between these two prototypical approaches for solving combinatorial and constraint satisfaction problems and discusses some possible integrations. The chapter starts with a description of the main complete and incomplete search techniques and then analyzes their peculiarities. Complete techniques explore exhaustively the whole search space and they are based on intelligent enumeration strategies which try to prune the search tree, for example by using lower and upper bounds and nogoods. Incomplete search techniques are the ones which do not guarantee to find the optimal solution (in the case of a CO problem) or a feasible solution (in the case of a Constraint Satisfaction Problem (CSP)) in bounded time and metaheuristics are one of its most representative classes. The author points out that the boundaries between complete and incomplete search are quite blurred. Hybrid approaches which integrate metaheuristics into tree-search and viceversa are surveyed and strengths and weaknesses of the various techniques are discussed. Taking inspiration from the considerations between contrasting complete and incomplete search, in the second part of the chapter the author proposes a hybrid search scheme for CSPs called *Incomplete Dynamic Backtracking* (IDB). This technique tries to build a solution as done in dynamic backtracking style algorithms, i.e., it can backtrack to an already assigned variable without unassigning the intermediate ones, with the difference that the choice of variable(s) to backtrack to is completely free and there is no exhaustiveness to be guaranteed. IDB can also be viewed as a local search with a cost function given by the number of unassigned variables. This is an emblematic case of an algorithm laying the fuzzy border between complete and incomplete search.

5.3 Chapter 4: Hybridizations of Metaheuristics With Branch & Bound Derivates

In their chapter, Blum et al. give a closer look at two specific ways of hybridizing metaheuristics with branch & bound (derivatives). The first one concerns the use of branch & bound features within construction-based metaheuristics such as ant colony optimization or greedy randomized adaptive search procedures in order to increase their efficiency. In particular, the authors deal with the case of Beam-ACO, a hybrid algorithm that results from replacing the standard solution construction procedure of ant colony optimization with a probabilistic beam search, which is an incomplete branch & bound derivative. After explaining the algorithm in general terms, an application to the longest common subsequence problem is presented.

The second part of the chapter concerns the use of a memetic algorithm in order to increase the efficiency of branch & bound, respectively beam search. More specifically, the memetic algorithm is used to obtain upper bounds for open subproblems of the branch & bound tree. The quality of the resulting hybrid technique is demonstrated by means of the application to another classical string problem, the shortest common supersequence problem.

5.4 Chapter 5: Large Scale Neighborhood Search

The chapter by Chiarandini et al. presents some notable examples of so-called large scale neighborhood search. Local search techniques are characterized by following a trajectory in the state space, moving at each iteration from a solution s to a new one s' by means of a so-called move, that is, by choosing the next solution s' in the *neighborhood* of s. Thus, the neighborhood of s is the set of candidates among which to choose the next solution. Small size neighborhoods are often preferred because of efficiency concerns, nevertheless they might make it difficult for the search to explore large portions of the search space or to move away from basins of attraction of local optima. In contrast, large scale neighborhoods, while enlarging the set of candidate (neighboring) solutions, can enable the search to enhance its exploration but at the price of higher computational time. The chapter by Chiarandini et al. discusses both exhaustive and heuristic algorithms for exploring large scale neighborhoods, in particular for the graph coloring problem and one of its extensions named graph set T-coloring problem. In the chapter, the authors emphasize the use of dynamic programming for the exploration of exponential size neighborhoods.

5.5 Chapter 6: Constructive Metaheuristics and Constraint Programming

The contribution by Meyer deals with another type of hybrid metaheuristic. More specifically, the author shows how constraint programming (CP) can be integrated with ACO. The key point in this study is that the construction phase in ACO can be performed by means of CP techniques; this approach can be particularly effective in problems in which the constraints make the search of a feasible solution hard, because in those cases the power of CP can be fully exploited. A dual, equivalent, perspective for combining CP and ACO is also discussed in which the learning mechanism of ACO is introduced in the variable/value selection mechanism during the labelling phase in CP. This phase consists in first choosing an unassigned variable and then selecting a value from the chosen variables' domain. This kind of tight integration of ACO and CP is compared against loose combination and pure versions of ACO and CP. The loose combination is implemented by (conceptually) running in parallel the two techniques and enabling a communication mechanism for exchanging partial solutions and bounds. Results on machine scheduling problem instances show that the tightly coupled approach is superior to the loosely coupled one and ACO and CP alone.

5.6 Chapter 7: Hybrid Metaheuristics for Packing Problems

The contribution by Ibaraki et al. shows the effectiveness of the integration of metaheuristics with several mathematical programming techniques on three variants of the two-dimensional packing problem. The problem consists in packing a set of items into a container with given size without overlaps between items. The authors consider first the formulation in which items are rectangular and have given and fixed size (in terms of width and height), then a variant in which sizes are adjustable within predefined limits and, finally, the problem with items without being restricted to assume a particular shape (also known as *irregular packing* or *nesting* problem). The point made by the authors is that hybrid metaheuristics can profit from the recent advances in software for mathematical programming by exploiting dynamic, linear and nonlinear programming for solving to optimality some specific subproblems. Indeed, the approaches discussed in the chapter rely on solution coding schemes that make the search of good configurations easier by providing a mechanism for representing a set of solutions in a compact way. Such coding schemes require the definition of decoding algorithms to transform a coded solution into an actual and complete one. Solution decoding can be viewed as a subproblem that has to be solved a large number of times during search. Therefore, decoding algorithms should be as much efficient as possible. Ibaraki et al. show how metaheuristic methods, such as iterated local search, can be fruitfully combined with decoding algorithms based on dynamic and (non)linear programming.

5.7 Chapter 8: Hybrid Multi-objective Combinatorial Optimization

While most of the book deals with hybrid methods for combinatorial optimization problems with only one objective, the chapter by Ehrgott and Gandibleux presents an overview on hybrid techniques for multi-objective combinatorial optimization. In fact, many real world optimization problems can be modelled as combinatorial optimization problems with multiple and even conflicting objectives. The authors give among others the example of railway transportation, where the planning of railway network infrastructure has the goals of maximizing the number of trains that can use it and to maximize the robustness of solutions to disruptions in operation. After giving an overview over the most important non-hybrid metheuristic methods proposed for multi-objective optimization, the authors survey the currently existing hybrid approaches. Hereby the authors distinguish between hybridization approaches in order to make the search more aggressive, hybrization in order to drive a metaheuristic, hybridization for exploiting the complementary strength of different techniques, and hybridization with, for example, exact methods. Finally, the authors conclude their chapter with a section on current hybridization trends. One of these trends concerns the used of lower and upper bounds on the non-dominated frontier for deciding if an expensive local search procedure should be started from a solution, or not.

5.8 Chapter 9: The Multilevel Paradigm

Instead of dealing with a typical hybridization method, the chapter by Walshaw rather presents a framework for using metaheuristics and/or other optimization techniques in a potentially more efficient way. This framework is commonly referred to as the multilevel paradigm, or the multilevel method. Its application to combinatorial optimization problems is quite simple. Basically, it involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found, usually at the coarsest level, for example by some metaheuristic algorithm. Then this solution is iteratively refined at each level, coarsest to finest. The same metaheuristic used for finding a solution to the coarsest level may be used for this purpose. Solution extension (or projection) operators can transfer the solution from one level to another. While this strategy has been used for many years, for example, in multigrid techniques, its application in combinatorial optimization is rather new. This chapter gives a survey on recent developments in this directions.

References

1. E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 91–120. John Wiley & Sons, Chichester, UK, 1997.
2. E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK, 1997.
3. E. Alba, editor. *Parallel Metaheuristics: A New Class of Algorithms*. John Wiley, 2005.
4. T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.
5. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing Ltd, Bristol, UK, 1997.
6. R. Battiti and M. Protasi. Reactive Search, a history-base heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2:Article 2, 1997.
7. R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
8. S. Binato, W. J. Hery, D. Loewenstern, and M. G. C. Resende. A greedy randomized adaptive search procedure for job shop scheduling. In P. Hansen and C. C. Ribeiro, editors, *Essays and surveys on metaheuristics*, pages 59–79. Kluwer Academic Publishers, 2001.
9. C. Blum. Ant colony optimization. *Physics of Life Reviews*, 2(4):353–373, 2005.
10. C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.
11. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
12. S. Boettcher and A. G. Percus. Optimization with extremal dynamics. *Complexity*, 8:57–62, 2003.
13. P. Calégary, G. Coray, A. Hertz, D. Kobler, and P. Kuonen. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5:145–158, 1999.
14. V. Černý. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
15. P. Chardaire, J. L. Lutton, and A. Sutter. Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. *European Journal of Operational Research*, 86:565–579, 1995.

16. C. A. Coello Coello. An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.
17. D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.
18. T. G. Crainic and M. Toulouse. Introduction to the special issue on Parallel Meta-Heuristics. *Journal of Heuristics*, 8(3):247–249, 2002.
19. T. G. Crainic and M. Toulouse. Parallel Strategies for Meta-heuristics. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2002.
20. F. Della Croce and V. T'kindt. A Recovering Beam Search algorithm for the one machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53(11):1275–1280, 2002.
21. M. Dell'Amico, A. Lodi, and F. Maffioli. Solution of the Cumulative Assignment Problem with a well–structured Tabu Search method. *Journal of Heuristics*, 5:123–143, 1999.
22. M. L. den Besten, T. Stützle, and M. Dorigo. Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 441–452. Springer-Verlag, Berlin, Germany, 2001.
23. J.-L. Deneubourg, S. Aron, S. Goss, and J.-M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3:159–168, 1990.
24. J. Denzinger and T. Offerman. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of Congress on Evolutionary Computation – CEC'1999*, pages 2317–2324, 1999.
25. M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
26. M. Dorigo and T. Stützle. http://www.metaheuristics.net/, 2000. Visited in January 2003.
27. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.
28. G. Dueck. New Optimization Heuristics. *Journal of Computational Physics*, 104:86–92, 1993.
29. G. Dueck and T. Scheuer. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics*, 90:161–175, 1990.
30. W. Feller. *An Introduction to Probability Theory and its Applications*. John Whiley, 1968.
31. T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
32. P. Festa and M. G. C. Resende. GRASP: An annotated bibliography. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.
33. A. Fink and S. Voß. Generic metaheuristics application to industrial engineering problems. *Computers & Industrial Engineering*, 37:281–284, 1999.

34. M. Fleischer. Simulated Annealing: past, present and future. In C. Alexopoulos, K. Kang, W. R. Lilegdon, and G. Goldsman, editors, *Proceedings of the 1995 Winter Simulation Conference*, pages 155–161, 1995.
35. F. Focacci, F. Laburthe, and A. Lodi. Local Search and Constraint Programming. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, 2002.
36. D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.
37. G. B. Fogel, V. W. Porto, D. G. Weekes, D. B. Fogel, R. H. Griffey, J. A. McNeil, E. Lesnik, D. J. Ecker, and R. Sampath. Discovery of RNA structural elements using evolutionary computation. *Nucleic Acids Research*, 30(23):5310–5317, 2002.
38. L. J. Fogel. Toward inductive inference automata. In *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, Munich, 1962.
39. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.
40. C. Fonlupt, D. Robilliard, P. Preux, and E. G. Talbi. Fitness landscapes and performance of meta-heuristics. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, 1999.
41. L. M. Gambardella and M. Dorigo. Ant Colony System hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.
42. M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W. H. Freeman, 1979.
43. M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristics for the capacitated VRP. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, pages 129–154. SIAM, Philadelphia, 2002.
44. F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8:156–166, 1977.
45. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.
46. F. Glover. Tabu Search Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
47. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
48. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, MA, 1989.
49. J. J. Grefenstette. A user's guide to GENESIS 5.0. Technical report, Navy Centre for Applied Research in Artificial Intelligence, Washington D.C., USA, 1990.
50. P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.
51. P. Hansen and N. Mladenović. Variable Neighborhood Search for the p-Median. *Location Science*, 5:207–226, 1997.
52. P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-*

Heuristics: Advances and Trends in Local Search Paradigms for Optimization, chapter 30, pages 433–458. Kluwer Academic Publishers, 1999.

53. P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
54. A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000.
55. T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *Proceedings of AAAI93*, pages 213–235. AAAI Press, 1993.
56. J. H. Holland. *Adaption in natural and artificial systems*. The University of Michigan Press, Ann Harbor, MI, 1975.
57. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier, Amsterdam, The Netherlands, 2004.
58. T. Ibaraki and K. Nakamura. Packing problems with soft rectangles. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 13–27. Springer-Verlag, Berlin, Germany, 2006.
59. L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics – Special Issue on Simulated Annealing Applied to Combinatorial Optimization*, 25(1):33–54, 1996.
60. D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.
61. T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, Univ. of New Mexico, Albuquerque, NM, 1995.
62. P. Kilby, P. Prosser, and P. Shaw. Guided Local Search for the Vehicle Routing Problem with time windows. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 473–486. Kluwer Academic Publishers, 1999.
63. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
64. H. R. Lourenço, O. Martin, and T. Stützle. A beginner's introduction to Iterated Local Search. In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, volume 1, pages 1–6, 2001.
65. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.
66. V. Maniezzo. Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
67. O. Martin and S. W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.
68. O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326, 1991.
69. D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for Resource-Constrained Project Scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346, 2002.

70. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.
71. Z. Michalewicz and M. Michalewicz. Evolutionary computation techniques and their applications. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems*, pages 14–24, Beijing, China, 1997. Institute of Electrical & Electronics Engineers, Incorporated.
72. M. Milano and A. Roli. MAGMA: A multiagent architecture for metaheuristics. *IEEE Trans. on Systems, Man and Cybernetics – Part B*, 34(2):925–941, 2004.
73. P. Mills and E. Tsang. Guided Local Search for solving SAT and weighted MAX-SAT Problems. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000*, pages 89–106. IOS Press, 2000.
74. M. Mitchell. *An introduction to genetic algorithms.* MIT press, Cambridge, MA, 1998.
75. P. Moscato. Memetic algorithms: A short introduction. In F. Glover D. Corne and M. Dorigo, editors, *New Ideas in Optimization*. McGraw-Hill, 1999.
76. G. L. Nemhauser and A. L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.
77. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.
78. I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.
79. P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.
80. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization – Algorithms and Complexity.* Dover Publications, Inc., New York, 1982.
81. L. S. Pitsoulis and M. G. C. Resende. Greedy Randomized Adaptive Search procedure. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.
82. M. Prais and C. C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.
83. S. Prestwich. Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search. *Annals of Operations Research*, 115:51–72, 2002.
84. S. Prestwich and A. Roli. Symmetry breaking and local search spaces. In *Proceedings of CPAIOR 2005*, volume 3524 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.
85. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J. R. Álvarez, editors, *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer-Verlag, Berlin, Germany, 2005.
86. G. R. Raidl. A unified view on hybrid metaheuristics. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Proceedings of HM 2006 – 3rd International Workshop on Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, Berlin, Germany, 2006.

87. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, 1973.
88. C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publishing, Oxford, England, 1993.
89. C. R. Reeves and J. E. Rowe. *Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory.* Kluwer Academic Publishers, Boston (USA), 2002.
90. M. G. C. Resende and C. C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.
91. C. C. Ribeiro and M. C. Souza. Variable neighborhood search for the degree constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118:43–54, 2002.
92. A. Roli. Symmetry-breaking and local search: A case study. In *SymCon'04 – 4th International Workshop on Symmetry and Constraint Satisfaction Problems.* 2004.
93. A. Schaerf, M. Cadoli, and M. Lenzerini. LOCAL++: A C++ framework for local search algorithms. *Software Practice & Experience*, 30(3):233–257, 2000.
94. G. R. Schreiber and O. C. Martin. Cut size statistics of graph bisection heuristics. *SIAM Journal on Optimization*, 10(1):231–251, 1999.
95. P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.-F. Puget, editors, *Principle and Practice of Constraint Programming – CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer-Verlag, 1998.
96. A. Shmygelska and H. H. Hoos. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformatics*, 6(30):1–22, 2005.
97. M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, 1997.
98. L. Sondergeld and S. Voß. Cooperative intelligent search using adaptive memory techniques. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 21, pages 297–312. Kluwer Academic Publishers, 1999.
99. W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. de Garis. An overview of evolutionary computation. In P. B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria, 1993. Springer-Verlag.
100. P. F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996. Also available as SFI preprint 95-07-067.
101. T. Stützle. *Local Search Algorithms for Combinatorial Problems – Analysis, Algorithms and New Applications.* DISKI – Dissertationen zur Künstliken Intelligenz. infix, Sankt Augustin, Germany, 1999.
102. T. Stützle and H. H. Hoos. $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.
103. É. D. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.
104. E.-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

105. M. Toulouse, T. G. Crainic, and B. Sansò. An experimental study of the systemic behavior of cooperative search algorithms. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter 26, pages 373–392. Kluwer Academic Publishers, 1999.
106. D. Urošević, J. Brimberg, and N. Mladenović. Variable neighborhood decomposition search for the edge weighted k-cardinality tree problem. *Computers & Operations Research*, 31:1205–1213, 2004.
107. P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40:113–125, 1992.
108. M. D. Vose. *The simple genetic algorithm: foundations and theory.* MIT Press, Cambridge, MA, 1999.
109. S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.
110. S. Voß and D. Woodruff, editors. *Optimization Software Class Libraries.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
111. C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems.* PhD thesis, Department of Computer Science, University of Essex, 1997. pp. 166.
112. C. Voudouris and E. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):469–499, 1999.
113. A. S. Wade and V. J. Rayward-Smith. Effective local search for the Steiner tree problem. *Studies in Locational Analysis*, 11:219–241, 1997. Also in *Advances in Steiner Trees*, ed. by Ding-Zhu Du, J. M.Smith and J. H. Rubinstein, Kluwer, 2000.
114. D. Whitley. The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms, ICGA 1989*, pages 116–121. Morgan Kaufmann Publishers, 1989.

Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization

Günther R. Raidl[1] and Jakob Puchinger[2]

[1] Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria,
raidl@ads.tuwien.ac.at

[2] NICTA Victoria Laboratory,
University of Melbourne, Melbourne, Australia,
jakobp@csse.unimelb.edu.au

Summary. Several different ways exist for approaching hard optimization problems. Mathematical programming techniques, including (integer) linear programming based methods, and metaheuristic approaches are two highly successful streams for combinatorial problems. These two have been established by different communities more or less in isolation from each other. Only over the last years a larger number of researchers recognized the advantages and huge potentials of building hybrids of mathematical programming methods and metaheuristics. In fact, many problems can be practically solved much better by exploiting synergies between these different approaches than by "pure" traditional algorithms. The crucial issue is *how* mathematical programming methods and metaheuristics should be combined for achieving those benefits. Many approaches have been proposed in the last few years. After giving a brief introduction to the basics of integer linear programming, this chapter surveys existing techniques for such combinations and classifies them into ten methodological categories.

1 Introduction

Computationally difficult *combinatorial optimization problems* (COPs) frequently appear in many highly important, practical fields. Creating good timetables, determining optimal schedules for jobs which are to be processed in a production line, designing efficient communication networks, container loading, determining efficient vehicle routes, and various problems arising in computational biology are a few examples. All these problems involve finding values for discrete variables such that an optimal solution with respect to a given objective function is identified subject to some problem specific constraints.

G.R. Raidl and J. Puchinger: *Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization*, Studies in Computational Intelligence (SCI) **114**, 31–62 (2008)
www.springerlink.com

Most COPs are difficult to solve. In theoretical computer science, this is captured by the fact that many such problems are NP-hard [38]. Because of the inherent difficulty and the enormous practical importance of NP-hard COPs, a large number of techniques for solving such problems has been proposed in the last decades. The available techniques for solving COPs can roughly be classified into two main categories: *exact* and *heuristic* algorithms. Exact algorithms are guaranteed to find an optimal solution and to prove its optimality for every instance of a COP. The run-time, however, often increases dramatically with a problem instance's size, and often only small or moderately-sized instances can be practically solved to proven optimality. For larger instances the only possibility is usually to turn to heuristic algorithms that trade optimality for run-time, i.e. they are designed to obtain good but not necessarily optimal solutions in acceptable time.

When considering exact approaches, the following techniques have had significant success: *branch-and-bound*, *dynamic programming*, *constraint programming*, and in particular the large class of *integer (linear) programming* (ILP) techniques including linear programming and other relaxation based methods, cutting plane and column generation approaches, branch-and-cut, branch-and-price, and branch-and-cut-and-price. See e.g. [52, 59] for general introductions to these mathematical programming techniques.

On the heuristic side, *metaheuristics* (MHs) have proven to be highly useful in practice. This category of problem solving techniques includes, among others, simulated annealing, tabu search, iterated local search, variable neighborhood search, various population-based models such as evolutionary algorithms, memetic algorithms, and scatter search, and estimation of distribution algorithms such as ant colony optimization. See Chap. 1 of this book as well as e.g. [41, 48] for more general introductions to metaheuristics.

Looking at the assets and drawbacks of ILP techniques and metaheuristics, the approaches can be seen as complementary to a large degree. As a matter of fact, it appears to be natural to combine ideas from both streams. Nevertheless, such hybrid approaches became more popular only over the last years. Nowadays, a multitude of recent publications describe different kinds of such hybrid optimizers that are often significantly more effective in terms of running time and/or solution quality since they benefit from synergy. International scientific events such as the *Hybrid Metaheuristics* workshop series [13, 12, 6], which started in 2004, and the *First Workshop on Mathematical Contributions to Metaheuristics – Matheuristics 2006* further emphasize the promise that is believed to lie in such hybrid systems. In fact, the artificial term "matheuristics" has been established by the latter event for referring to combinations of metaheuristics and mathematical programming methods.

In the next section, we will continue with a brief introduction of previously suggested structural classifications of strategies for combining metaheuristics and exact optimization techniques. Sect. 3 gives an overview on the basics of prominent ILP techniques and introduces used notations. Various different methodologies of utilizing ILP techniques in metaheuristics and vice versa, including annotated references to successful examples, are then reviewed in Sects. 4 to 13. These MH/ILP hybridization methodologies are

- MHs for finding high-quality incumbents and bounds in branch-and-bound,
- relaxations for guiding metaheuristic search,
- using the primal-dual relationship in MHs,
- following the spirit of local search in branch-and-bound,
- ILP techniques for exploring large neighborhoods,
- solution merging,
- ILP techniques as decoders for indirect or incomplete representations,
- multi-stage approaches,
- cut and column generation by metaheuristics,
- strategic guidance of search and collaboration.

2 Structural Models for Combining Metaheuristics With Exact Approaches

Overviews on various structural models of combining exact techniques and metaheuristics are given in [25, 67, 75].

Dumitrescu and Stützle [25] describe existing combinations which primarily focus on local search approaches that are strengthened by the use of exact algorithms. In their survey they concentrate on integration and exclude obvious combinations such as preprocessing.

In [67] we present a more general classification of existing approaches combining exact and metaheuristic algorithms for combinatorial optimization in which the following two main categories are distinguished, see also Fig. 1:

Collaborative Combinations. In a collaborative environment, the algorithms exchange information, but are not part of each other. Exact and heuristic algorithms may be executed sequentially, intertwined, or in parallel.

Integrative Combinations. In integrative models, one technique is a subordinate embedded component of another technique. Thus, there is a distinguished master algorithm, which can be either an exact or a metaheuristic algorithm, and at least one integrated slave.

Danna and Le Pape [21] present a similar classification of hybrid algorithms, further including constraint programming. The authors discern a *decomposition scheme* corresponding to the integrative combinations and a *multiple search scheme* corresponding to collaborative combinations. Four kinds of optimization algorithms are considered in particular, namely polynomial operations research algorithms, constraint programming, mixed integer programming, and various forms of local search and metaheuristics. The main part of their article consists of examples from the literature illustrating six

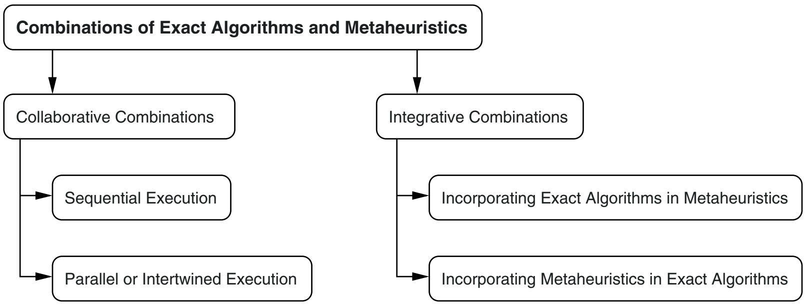

Combinations of Exact Algorithms and Metaheuristics
Collaborative Combinations
Integrative Combinations
Sequential Execution
Parallel or Intertwined Execution
Incorporating Exact Algorithms in Metaheuristics
Incorporating Metaheuristics in Exact Algorithms

Fig. 1. Major structural classification of exact/metaheuristic combinations according to [67].

different collaborative schemes consisting of two of the above mentioned algorithm classes.

A taxonomy on hybrid metaheuristics in general has been proposed by Talbi [82]. Various hybridization schemes involving in particular *evolutionary algorithms* (EAs) are described by Cotta [19]. El-Abd and Kamel [26] particularly addressed cooperative parallel architectures.

Raidl [75] tries to unify previous classifications and taxonomies of hybrid metaheuristics and primarily distinguishes (a) the type of algorithms that are hybridized, (b) the level of hybridization (high- or low-level), (c) the order of execution (batch, interleaved, or parallel), and (d) the control strategy (integrative or collaborative).

3 Linear and Integer Programming at a Glance

This section gives a short overview of the main concepts in integer programming; for an in-depth coverage of the subject we refer to the books on linear optimization by Bertsimas and Tsitsiklis [11] and on combinatorial and integer optimization by Nemhauser and Wolsey [59] and Wolsey [88].

An integer (linear) program is an optimization problem involving integer variables, an objective function linearly depending on the variables, and a set of constraints expressed as linear (in)equalities. We consider the form

$$z_{\mathrm{ILP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}, \tag{1}$$

where x is the n-dimensional integer variable column vector and $c \in \mathbb{R}^n$ an n-dimensional row vector. Their dot-product cx is the *objective function* that should be minimized. Matrix $A \in \mathbb{R}^{m \times n}$ and the m-dimensional column vector $b \in \mathbb{R}^m$ together define m inequality constraints.

Maximization problems can be converted into minimization problems by simply changing the sign of c. Less-than constraints are similarly brought into greater-than-or-equal form by changing the sign of the corresponding coefficients, and equalities can be translated to pairs of inequalities. Thus, we can consider all kinds of linear constraints by appropriate transformations. Without loss of generality, we may therefore restrict our following considerations to minimization problems of the form (1).

A *mixed integer (linear) program* (MIP) involves a combination of integer and real-valued variables, but is otherwise defined in the same way.

3.1 Relaxations and Duality

One of the most important concepts in integer programming are *relaxations*, where some or all constraints of a problem are loosened or omitted. Relaxations are mostly used to obtain related, simpler problems which can be solved efficiently yielding bounds and approximate (not necessarily feasible) solutions for the original problem.

The *linear programming relaxation* of the ILP (1) is obtained by relaxing the integrality constraint, yielding the *linear program* (LP)

$$z_{\mathrm{LP}} = \min\{cx \mid Ax \geq b, x \geq 0, x \in \mathbb{R}^n\}. \tag{2}$$

Large instances of such LPs can be efficiently solved in practice using simplex-based or interior-point algorithms. The linear programming relaxation always provides a lower bound for the original minimization problem, i.e. $z_{\mathrm{ILP}} \geq z_{\mathrm{LP}}$, since the search space of the ILP is contained in the one of the LP and the objective function remains the same.

According to linear programming theory, we can further associate a *dual problem* to each LP (2), which is defined by

$$w_{\mathrm{LP}} = \max\{ub \mid uA \leq c, u \geq 0, u \in \mathbb{R}^m\}. \tag{3}$$

The dual of the dual LP is the original (*primal*) LP again. Important relations between the primal problem and its dual are known as weak and strong duality theorems, respectively:

- The value of every finite feasible solution to the dual problem is a lower bound for the primal problem, and each value of a finite feasible solution to the primal problem is an upper bound for the dual problem. As a consequence, if the dual is unbounded, the primal is infeasible and vice versa.
- If the primal has a finite optimal solution z^*_{LP}, than its dual has the same optimal solution $w^*_{\mathrm{LP}} = z^*_{\mathrm{LP}}$ and vice versa.

The complementary slackness conditions follow from the strong duality theorem: Suppose x and u are feasible solutions for (2) and (3), respectively; then they are optimal if and only if the following conditions hold:

$$u(Ax - b) = 0 \quad \text{and} \tag{4}$$

$$x(c - uA) = 0. \tag{5}$$

In case of an integer linear problem, we have to differentiate between the notions of weak and strong duals. A *weak dual* of an ILP (1) is any maximization problem $w = \max\{w(u) \mid u \in S_{\mathrm{D}}\}$ such that $w(u) \leq cx$ for all $x \in \{Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}$. An obvious weak dual of (1) is the dual (3) of its LP relaxation (2). A *strong dual* w is a weak dual that further has an optimal solution u^* such that $w(u^*) = cx^*$ for an optimal solution x^* of (1). For solving ILPs, weak duals which are iteratively strengthened during the course of the optimization process are often utilized.

Another standard relaxation technique for ILPs, which often yields significantly tighter bounds than the LP relaxation, is *Lagrangian relaxation* [33, 34]. Consider the ILP

$$z_{\mathrm{ILP}} = \min\{cx \mid Ax \geq b, Dx \geq d, x \geq 0, x \in \mathbb{Z}^n\}, \tag{6}$$

where constraints $Ax \geq b$ are "nice" in the sense that the problem can be efficiently solved when the m' "complicating" constraints $Dx \geq b$ are dropped. Simply dropping these constraints of course yields a relaxation, however, the resulting bound will usually be weak due to the total ignorance of part of the inequalities. In Lagrangian relaxation, constraints $Dx \geq d$ are replaced by corresponding additional terms in the objective function:

$$z_{\mathrm{LR}}(\lambda) = \min\{cx + \lambda(d - Dx) \mid Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}. \tag{7}$$

Vector $\lambda \in \mathbb{R}^{m'}$ is the vector of Lagrangian multipliers, and for any $\lambda \geq 0$, $z_{\mathrm{LR}}(\lambda) \leq z_{\mathrm{ILP}}$, i.e. we have a valid relaxation of the ILP. We are now interested in finding a specific vector λ yielding the best possible bound, which leads to the *Lagrangian dual problem*

$$z^*_{\mathrm{LR}} = \max_{\lambda \geq 0}\{z_{\mathrm{LR}}(\lambda)\}. \tag{8}$$

It can be shown that this Lagrangian dual is a piecewise linear and convex function, and usually, it can be well solved by iterative procedures like the subgradient method. A more elaborate algorithm that has been reported to converge faster on several problems is the volume algorithm [10], whose name is inspired by the fact that primal solutions are also considered, whose values come from approximating the volumes below active faces of the dual problem.

Given a solution λ to the Lagrangian dual problem (8) and a corresponding optimal solution x^* to the Lagrangian relaxation (7) which is also feasible to the original problem (6), i.e. $Dx^* \geq d$, the following complementary slackness condition holds: x^* is an optimal solution to the original problem (6) if and only if

$$\lambda(d - Dx^*) = 0. \tag{9}$$

It can be shown that the Lagrangian relaxation always yields a bound that is at least as good as the one of the corresponding linear relaxation, providing the Lagrangian dual problem is solved to optimality.

A third general-purpose relaxation technique for ILPs is *surrogate relaxation* [43]. Here, some or all constraints are scaled by surrogate multipliers and cumulated into a single inequality by addition of the coefficients. Similar as in Lagrangian relaxation, the ultimate goal is to find surrogate multipliers yielding the overall best bound. Unfortunately, this surrogate dual problem has not such nice properties as the Lagrangian dual problem and solving it is often difficult. However, if one is able to determine optimal surrogate multipliers, the bound obtained for the ILP is always at least as good as (and often better than) those obtained from linear and Lagrangian relaxation.

3.2 Cutting Plane Approach

When modeling COPs as ILPs, an important goal is to find a *strong* formulation, for which the LP relaxation provides a solution which lies in general not too far away from the integer optimum. For many COPs it is possible to strengthen an existing ILP formulation significantly by including further inequalities. Often, the number of such constraints grows exponentially with the problem size. This, however, means that already solving the LP relaxation by standard techniques might be too costly in practice due to the exponentially sized LP. Dantzig et al. [23] proposed the *cutting plane approach* for this purpose, which usually only considers a small subset of all constraints explicitly and nevertheless is able to determine an optimal solution to the whole LP.

This cutting plane approach starts with a small subset of initial inequalities and solves this reduced LP. Then, it tries to find inequalities that are not satisfied by the obtained solution but are valid for the original problem (i.e. contained in the full LP). These violated constraints are called *cuts* or *cutting planes*. They are added to the current reduced LP, and the LP is resolved. The whole process is iterated until no further cuts can be found. If the algorithm is able to provide a proof that no further violated inequality exists, the finally obtained solution is also optimal with respect to the original full LP. The subproblem of identifying cuts is called *separation problem*, and it is of crucial importance to solve it efficiently, since many instances of it must usually be solved until the cutting plane approach terminates successfully.

Note that from a theoretical point of view it is possible to solve any ILP using a pure cutting plane approach with appropriate classes of cuts. There exist generic types of cuts, such as the Chvatal-Gomory cuts [88], which guarantee such a result. In practice, however, it may take a long time for such a cutting plane approach to converge to the optimum, partly because it is often a hard subproblem to separate effective cuts. The cutting plane method is therefore often combined with other methods, as we will see below.

3.3 Column Generation Approach

Instead of considering many inequalities, it is often also a reasonable option to formulate a problem in a strong way via a large number of variables, which correspond to columns in the coefficient matrix. The *(delayed) column generation approach* starts with a small subset of these variables and solves the corresponding restricted LP. Then, the algorithm tries to identify one or more not yet considered variables, whose inclusion might lead to an improved solution. This subproblem is called *pricing problem*, and for a minimization problem a variable is suitable in this sense if and only if it has negative reduced costs. After including such newly found variables in the restricted LP, the LP is resolved and the process iterated until it can be proven that no further variables with negative reduced costs exist, i.e. all variables *price out correctly*. An optimal solution for the original complete LP is then obtained. Column generation can be seen as the dual of the cutting plane approach, since inequalities correspond to variables in the dual LP.

A classical example where column generation is highly successful is the cutting stock problem [39]. A decision variable is defined for each possible cutting pattern, clearly yielding an exponential number of variables, and the pricing problem corresponds to the classical knapsack problem, which can be solved in pseudo-polynomial time. For a thorough review on column generation, we refer to [55].

A general technique for obtaining possibly strengthened ILP formulations is the Dantzig-Wolfe decomposition. It transforms original variables into linear combinations of extreme points and extreme rays of the original search space, yielding a potentially exponential number of variables. The resulting problems are usually solved by column generation.

3.4 Branch-and-Bound Methods

By solving the LP relaxation of an ILP problem, we usually only get a lower bound on the optimal integer solution value, and the solution will in general also contain fractional values. For hard COPs, this typically also holds for strengthened formulations and when cutting plane or column generation procedures have been applied, although the obtained bound might be much better. The standard way of continuing in order to finally determine an integer solution is *branch-and-bound* (B&B). This is a divide-and-conquer approach that solves an ILP by recursively splitting it into disjoint subproblems. Bounds are calculated for the subproblems, and only those potentially holding an optimal solution are kept for further processing, whereas the others are pruned from the B&B tree.

The main idea in *LP-based B&B* is to use an LP relaxation of the ILP being solved in order to derive a lower bound for the objective function. A standard way for branching is to pick one of the fractional variables, say x_i with its current LP-value x_i^*, and define as first subproblem the ILP with the additional inequality $x_i \leq \lfloor x_i^* \rfloor$ and as second subproblem the ILP with inequality $x_i \geq \lceil x_i^* \rceil$. For these subproblems with the additional branching constraints, the LP is resolved, eventually leading to increased lower bounds. Usually, primal heuristics are also applied to each subproblem in order to possibly obtain an improved feasible solution and a corresponding global upper bound.

Combining B&B with cutting plane algorithms yields the highly effective class of *branch-and-cut algorithms* which are widely used in commercial ILP-solvers. Cuts are generated at the nodes of the B&B tree to tighten the bounds of the LP relaxations or to exclude infeasible solutions.

The combination of B&B with column generation results in *branch-and-price* algorithms, where new columns may be generated at each node in order to optimally solve their corresponding LP relaxations.

Finally, *branch-and-cut-and-price* refers to the combination of all of the above methods, often resulting in highly specialized and most powerful optimization algorithms.

We now turn to the different methodologies of hybridizing these ILP techniques (and some further mathematical programming approaches) with metaheuristics.

4 Metaheuristics for Finding High-Quality Incumbents and Bounds in B&B

Almost any effective B&B approach depends on some heuristic for deriving a promising initial solution, whose objective value is used as original upper bound. Furthermore, and as already mentioned, heuristics are typically also applied to some or all subproblems of the B&B tree in order to eventually obtain new incumbent solutions and corresponding improved upper bounds. In order to keep the B&B tree relatively small, good upper bounds are of crucial interest. Therefore, metaheuristics are often also applied for these purposes.

However, when performing a relatively expensive metaheuristic at each node of a large B&B tree in a straight-forward, independent way, the additional computational effort often does not pay off. Different calls of the metaheuristic might perform more or less redundant searches in similar areas of the whole search space. A careful selection of the B&B tree nodes for which the metaheuristic is performed and how much effort is put into each call is therefore crucial.

As an example, Woodruff [89] describes a chunking-based selection strategy to decide at each node of the B&B tree whether or not reactive tabu search is called. The chunking-based strategy measures a distance between the current node and nodes already explored by the metaheuristic in order to bias the selection toward distant points. Reported computational results indicate that adding the metaheuristic improves the B&B performance.

5 Relaxations for Guiding Metaheuristic Search

An optimal solution for a relaxation of the original problem often indicates in which areas of the original problem's search space good or even optimal solutions might lie. Solutions to relaxations are therefore frequently exploited in (meta-)heuristics. In the following, we study different possibilities for such approaches.

5.1 Creating Promising Initial Solutions

Sometimes an optimal solution to a relaxation can be repaired by a problem-specific procedure in order to make it feasible for the original problem and to use it as promising starting point for a subsequent metaheuristic (or exact) search. Often, the linear programming (LP) relaxation is used for this purpose, and only a simple rounding scheme is needed.

For example, Raidl and Feltl [73] describe a hybrid *genetic algorithm* (GA) for the generalized assignment problem, in which the LP relaxation of the problem is solved, and its solution is exploited by a randomized rounding procedure to create an initial population of promising integral solutions. These solutions are, however, often infeasible; therefore, randomized repair and improvement operators are additionally applied, yielding an even more meaningful initial population for the GA.

Plateau et al. [64] combine interior point methods and metaheuristics for solving the *multidimensional knapsack problem* (MKP). In a first step an interior point method is performed with early termination. By rounding and applying several different ascent heuristics, a population of different feasible candidate solutions is generated. This set of solutions is then used as initial population for a path-relinking/scatter search. Obtained results show that the presented combination is a promising research direction.

5.2 Guiding Repairing, Local Improvement, and Variation Operators

Beside initialization, optima of LP relaxations are often exploited for guiding local improvement or the repairing of infeasible candidate solutions. For example, in [74] the MKP is considered, and variables are sorted according to increasing LP-values. A greedy repair procedure considers the variables in this order and removes items from the knapsack until all constraints are fulfilled. In a greedy improvement procedure, items are considered in reverse order and included in the knapsack as long as no constraint is violated.

Many similar examples for exploiting LP solutions, also including a biasing of variation operators like recombination and mutation in EAs, exist.

5.3 Exploiting Dual Variables

Occasionally, dual variable values are also exploited. Chu and Beasley [15] make use of them in their GA for the MKP by calculating so-called *pseudo-utility ratios* for the primal variables and using them in similar ways as described above for the primal solution values. These pseudo-utility ratios tend to give better indications of the likeliness of the corresponding items to be included in an optimal solution; see [76] for more details on GA approaches for the MKP.

5.4 Variable Fixing: Reduction to Core Problems

Another possibility of exploiting the optimal solution of an LP relaxation is more direct and restrictive: Some of the decision variables having integral values in the LP-optimum are fixed to these values, and the subsequent optimization only considers the remaining variables. Such approaches are sometimes also referred to as *core methods*, since the original problem is reduced and only its "hard core" is further processed. Obviously, the selection of the variables in the core is critical.

The core concept has originally been proposed for the 0–1 knapsack problem [9] and also led to several very successful exact algorithms such as [63]. Puchinger et al. [72] extend this approach for the MKP and investigated several variants for choosing approximate cores. Considering binary decision variables $x_1, \ldots, x_n \in \{0, 1\}$, the basic technique first sorts all variables according to some specific efficiency measure and determines the so-called split-interval, which is the subsequence of the variables starting with the first and ending with the last fractional variable. Different efficiency measures are studied, and it is shown that the above already mentioned pseudo-utility ratios, which are determined from dual variable values, are in general a good choice for the MKP. The split interval is finally extended to an approximate core by adding $\delta > 0$ further variables on each side of the center of the split-interval. Empirical investigations in [72] indicate that already with $\delta = 0.1n$, high quality solutions with average optimality gaps less than 0.1% can be achieved when solving the remaining core problem to proven optimality. Applying an EA and relaxation guided variable neighborhood search to the reduced problem instances yields significantly better solutions in shorter time than when applying these metaheuristics to the original instances.

Staying with the MKP, another example for exploiting the LP relaxation within metaheuristics is the hybrid tabu search algorithm from Vasquez and Hao [86]. Here, the search space is reduced and partitioned via additional constraints fixing the total number of items to be packed. Bounds for these constraints are calculated by solving modified LP relaxations. For each remaining part of the search space, tabu search is independently applied, starting with a solution derived from the LP relaxation of the partial problem. The approach has further been improved in [87] by additional variable fixing.

To our knowledge, this method is currently the one yielding the best results on a commonly used library of MKP benchmark instances.

5.5 Exploiting Lagrangian Relaxation

Also other relaxations besides the LP relaxation are occasionally successfully exploited in conjunction with metaheuristics. The principal techniques for such combinations are similar. A successful example is the hybrid Lagrangian GA for the prize collecting Steiner tree problem from Haouaria and Siala [47]. They perform a Lagrangian decomposition on a minimum spanning tree formulation of the problem and apply the volume algorithm for solving the Lagrangian dual. After termination, the genetic algorithm is started and exploits results obtained from the volume algorithm in several ways:

- Graph reduction: The volume algorithm creates a sequence of intermediate spanning trees as a by-product. All edges appearing in these intermediate trees are marked, and only this reduced edge set is further considered by the GA; i.e. a core of edges is derived from the intermediate primal results when solving the Lagrangian dual.
- Initial population: A subset of diverse initial solutions is created by a Lagrangian heuristic, which greedily generates solutions based on the reduced costs appearing as intermediate results in the volume algorithm.
- Objective function: Instead of the original objective function, an alternate one is used, which is based on the reduced costs that are finally obtained by the volume algorithm. The idea is to guide the search into regions of the search space, where also better solutions with respect to the original objective function can presumably be found.

Pirkwieser et al. [62] described a similar combination of Lagrangian decomposition and a GA for the knapsack constrained maximum spanning tree problem. By Lagrangian relaxation, the problem is decomposed into a minimum spanning tree and a 0–1 knapsack problem. Again, the volume algorithm is employed to solve the Lagrangian dual. While graph reduction takes place as before, the objective function remains unchanged. Instead, final reduced costs are exploited for biasing the initialization, recombination, and mutation operators. In addition, the best feasible solution obtained from the volume algorithm is used as a seed in the GA's initial population. Results indicate that the volume algorithm alone is already able to find solutions of extremely high quality also for large instances. These solutions are polished by the GA, and in most cases proven optimal solutions are finally obtained.

6 Using the Primal-Dual Relationship in Metaheuristics

Using the primal-dual relationship in metaheuristics is a relatively recent approach; only a few papers have been published in this area. One idea is to take advantage of the complementary slackness conditions (5) or (9). Starting from a feasible dual solution u we try to find a primal feasible solution x satisfying these conditions with respect to u. On the other hand, if one searches in the dual as well as in the primal space, one may be able to give meaningful performance guarantees for heuristically obtained primal feasible solutions.

6.1 Generating Tight Bounds

Hansen et al. [44] present a primal-dual *variable neighborhood search* (VNS) for the *simple plant location problem* (SPLP). Since the tackled instances are too big to be solved by linear programming techniques, the authors propose to first perform a *variable neighborhood decomposition search* to the SPLP yielding a primal feasible solution. An initial, possibly infeasible, dual solution is then devised by exploiting the complementary slackness conditions. This solution is locally improved by applying *variable neighborhood descent* (VND), which also reduces a potential infeasibility. An exact dual solution is required to derive a correct lower bound for the SPLP. It is obtained by applying the recently developed sliding simplex method. The authors further use the generated bounds to strengthen a B&B algorithm exactly solving the SPLP. The presented computational experiments show the efficiency of the proposed approach, which is able to solve previously unsolved instances to proven optimality.

6.2 Integrating Primal and Dual Solution Approaches

Rego [77] describes a metaheuristic framework, called *relaxation adaptive memory programming* (RAMP), which combines principles of Lagrangian and surrogate relaxation with those of *adaptive memory programming* (AMP) [81]. He further proposes a primal-dual extension PD-RAMP and a specific implementation of PD-RAMP based on Lagrangian and surrogate constraint relaxation on the dual side and scatter search and path-relinking on the primal side.

Lagrangian and surrogate relaxation are combined into a cross-parametric relaxation method, which uses subgradient optimization to generate good surrogate constraints. Dual solutions are projected into the primal space by applying constructive and improvement heuristics. The approach yields primal solutions as well as dual bounds and may therefore be able to prove optimality or give performance guarantees for generated solutions. Using AMP for projecting solutions from the dual to the primal space yields the RAMP framework. The authors propose to use frequency based tabu search or a method were tabu search and path-relinking are combined. The primal-dual RAMP approach switches back and forth between a relaxation method and a path-relinking in the primal space, both updating the same reference set. The author describes preliminary computational experiments, where PD-RAMP is dominating the performance of the best known methods from the literature for different variants of the generalized assignment problem.

7 Following the Spirit of Local Search in B&B

Most metaheuristics are based on the principle of local search, i.e. starting from an initial solution, a certain neighborhood around it is investigated, and if a better solution can be identified, it becomes the new incumbent solution; this process is repeated. Thus, the central idea is to focus the search for better solutions on regions of the search space nearby already identified, good solutions.

In comparison, most B&B algorithms choose the next B&B tree node to be processed by a *best-first* strategy: a node with smallest lower bound is always selected, since it is considered to be most promising to contain an optimal solution. This approach is often the best strategy for minimizing the total number of nodes that need to be explored until finding an optimum and proving its optimality. However, good complete solutions and thus also tight upper bounds are often found late during this search. The best-first node selection strategy typically "hops around" on the search tree and in the search space, and does not stay focused on subregions. When no strong primal heuristic is applied for determining promising complete solutions, the best-first strategy is often combined with an initial *diving*, in which a depth-first strategy is followed at the beginning until some feasible solution is obtained. In depth-first search, the next node to be processed is always one that has been most recently been created by branching.

In the last years, several more sophisticated concepts have been proposed with the aim to intensify B&B-search in an initial phase to neighborhoods of promising incumbents in order to quickly identify high quality heuristic solutions. In some sense, we can consider these strategies to "virtually" execute a metaheuristic. We will review some of these strategies in the following.

7.1 Guided Dives

Danna et al. [22] describe *guided dives*, which are a minor, but effective modification of the already mentioned simple diving by temporarily switching to depth-first search. Consider a classical branching in LP-based B&B over a fractional variable, as described in Sect. 3.4. The subproblem to be processed next in case of guided dives is always the one in which the branching variable is allowed to take the value it has in a current incumbent solution. Diving is therefore biased towards the neighborhood of the given incumbent. Instead of performing only a single dive at the beginning, guided dives are repeatedly applied in regular intervals during the whole optimization. While this strategy is trivial to implement, experimental results indicate significant advantages over standard node selection strategies.

7.2 Local Branching

Fischetti and Lodi [32] propose *local branching*, an exact approach introducing the spirit of classical k-OPT local search in a generic branch-and-cut based MIP solver. They consider general MIPs with 0–1 variables. Let $x = (x_1, \ldots, x_n)$ be the vector of all variables and $\mathcal{B} \subseteq \{1, \ldots, n\}$ be the index set of the 0–1 variables. The following *local branching constraint* is used for defining a k-OPT neighborhood around a given incumbent solution $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$:

$$\varDelta(x, \overline{x}) := \sum_{j \in \overline{\mathcal{S}}} (1 - x_j) + \sum_{x \in \mathcal{B} \setminus \overline{\mathcal{S}}} (x_j) \leq k, \tag{10}$$

where $\overline{\mathcal{S}} = \{j \in \mathcal{B} \mid \overline{x}_j = 1\}$ being the index set of 0–1 variables set to 1 in the incumbent solution. Note that $\varDelta(x, \overline{x})$ resembles the classical Hamming distance between x and $\overline{x}$.

In the main algorithm, the whole problem is partitioned into the k-OPT neighborhood of an initial solution $\overline{x}$ and the rest by branching according to inequality (10) and the reverse constraint $\varDelta(x, \overline{x}) \geq k + 1$, respectively. The MIP solver is then enforced to completely solve the k-OPT neighborhood before considering the rest.

If an improved solution $\overline{x}'$ has been found in the k-OPT neighborhood, a new subproblem $\varDelta(x, \overline{x}') \leq k$ is split off from the rest and solved in the same way; this process is repeated until no further improvements can be achieved. Finally, the remaining problem corresponding to all not yet considered parts of the search space is processed in a standard way.

This basic mechanism is extended by introducing time limits, automatically modifying the neighborhood size k, and adding diversification strategies in order to improve performance. Furthermore, an extension of the branching constraint for general integer variables is also proposed. Reported results on various benchmark MIP instances using CPLEX[1] as MIP solver indicate the advantages of the approach in terms of an earlier identification of high-quality heuristic solutions.

Hansen et al. [46] present a variant of the local branching approach in which they follow more closely the standard VNS strategy [45] when switching between neighborhoods. Improved results are reported.

Another variant of the original local branching scheme is described by Fischetti et al. [31]. They consider in particular problems in which the set of variables partitions naturally into two levels, with the property that fixing the values of the first-level variables yields a substantially easier subproblem.

Lichtenberger [53] describes an extended local branching framework in which several k-OPT neighborhoods induced by a set of candidate solutions can be processed in a pseudo-simultaneous (intertwined) way. This allows the "virtual" implementation of population-based metaheuristics like EAs on top of a B&B-based MIP solver. The framework was tested on the MKP. In order to keep the computational effort for processing the k-OPT neighborhoods reasonably low, an additional variable fixing strategy is applied.

[1] http://www.ilog.com

7.3 The Feasibility Pump

Sometimes, it is already hard to identify any feasible initial solution for a MIP. For this purpose, Fischetti et al. [30] suggest an algorithm called *feasibility pump*. The method starts by solving the LP relaxation yielding a fractional solution x^*. A (usually infeasible) integer solution $\overline{x}$ is derived by simple rounding. From it, the nearest feasible point in the polytope defined by the LP relaxation is determined by solving a linear program with the Hamming distance $\Delta(x, \overline{x})$ as objective function. When the obtained solution is integral, a feasible solution for the original MIP has been found; otherwise, the process is repeated.

7.4 Relaxation Induced Neighborhood Search

Danna et al. [22] further suggest an alternative approach called *relaxation induced neighborhood search* (RINS) in order to explore the neighborhoods of promising MIP solutions more intensively. The main idea is to occasionally devise a sub-MIP at a node of the B&B tree that corresponds to a special neighborhood of an incumbent solution: First, variables having the same values in the incumbent and in the current solution of the LP relaxation are fixed. Second, an objective cutoff based on the objective value of the incumbent is set. Third, a sub-MIP is solved on the remaining variables. The time for solving this sub-MIP is limited. If a better incumbent could be found during this process, it is passed to the global MIP-search which is resumed after the sub-MIP termination. In the authors' experiments, CPLEX is used as MIP solver, and RINS is compared to standard CPLEX, local branching, combinations of RINS and local branching, and guided dives. Results indicate that RINS often performs best. The current version 10 of CPLEX also includes RINS as a standard strategy for quickly obtaining good heuristic solutions.

8 ILP Techniques for Exploring Large Neighborhoods

A common approach in more sophisticated local search based metaheuristics is to search neighborhoods by means of clever exact algorithms. If the neighborhoods are chosen appropriately, they can be relatively large and nevertheless an efficient search for the best neighbor is still reasonable. Such techniques are known as *very large-scale neighborhood* (VLSN) search [3]. Probably most of today's combinations of local search based metaheuristics and ILP techniques follow this approach. In the following, we present some examples.

In *Dynasearch* [17, 18] exponentially large neighborhoods are explored by dynamic programming. A neighborhood where the search is performed consists of all possible combinations of mutually independent simple search steps, and one Dynasearch move corresponds to a set of independent moves that are executed in parallel in a single local search iteration. Independence in the context of Dynasearch means that the individual moves do not interfere with each other; in this case, dynamic programming can be used to find the best combination of independent moves. Dynasearch is restricted to problems where the single search steps are independent, and to our knowledge it has so far only been applied to problems where solutions are represented by permutations. Ergun and Orlin [28] investigated several such neighborhoods in particular for the traveling salesman problem.

For a class of partitioning problems, Thompson et al. [84, 85] suggest the concept of a cyclic exchange neighborhood, which is based on the transfer of single elements between an unrestricted number of subsets in a cyclic manner. A 2-exchange move can be seen as the simplest case of a cyclic exchange having length two. To efficiently determine a best cyclic exchange for a current solution, a weighted, directed graph is constructed, in which each arc represents a possible transfer of a single element and the arc's weight corresponds to the induced difference in the objective value of the solution. A best cyclic exchange can then be derived by finding a smallest negative-cost subset-disjoint cycle in this graph. The authors consider exact and heuristic methods for this purpose.

Puchinger et al. [71] describe a combined GA/B&B approach for solving a real-world glass cutting problem. The GA uses an order-based representation, which is decoded using a greedy heuristic. The B&B algorithm is applied with a certain probability enhancing the decoding phase by generating locally optimal subpatterns. Reported results indicate that the approach of occasionally solving subpatterns to optimality often increase the overall solution quality.

Büdenbender et al. [14] present a tabu search hybrid for solving a real-world direct flight network design problem. Neighborhoods are created by fixing a large subset of the integer variables corresponding to the performed flights and allowing the other variables to be changed. CPLEX is used to solve the reduced problems corresponding to these neighborhoods. Diversification is performed by closing flights frequently occurring in previously devised solutions.

Prandtstetter and Raidl [65] apply variable neighborhood search to the car sequencing problem and also use CPLEX for searching large neighborhoods. A subset of the scheduled cars is selected, removed from the schedule, and reinserted in an optimal way. The neighborhoods differ in the technique used to choose the cars and their number. Results indicate that this approach can compete well with leading algorithms from a competition organized by the French Operations Research Society ROADEF in 2005.

Hu et al. [49] propose a VNS metaheuristic for the generalized minimum spanning tree problem. The approach uses two dual types of representations and associated exponentially large neighborhoods. Best neighbors are identified by means of dynamic programming algorithms, and – in case of the so-called global subtree optimization neighborhood – by solving an ILP formulation with CPLEX. Experimental results indicate that each considered neighborhood contributes well to the whole success, and the algorithm obtains significantly better solutions than previous metaheuristics.

Puchinger and Raidl [68] suggest a new variant of VNS: *relaxation guided variable neighborhood search*. It is based on the general VNS scheme and a new VND algorithm. The ordering of the neighborhood structures in this VND is determined dynamically by solving relaxations of them. The objective values of these relaxations are used as indicators for the potential gains of searching the corresponding neighborhoods. The proposed approach has been tested on the MKP. Computational experiments involving several ILP-based neighborhoods show that relaxation guided VNS is beneficial to the search, improving the obtained results. The concept is more generally applicable and seems to be promising for many other combinatorial optimization problems approached by VNS.

9 Solution Merging

In *evolutionary algorithms* (EAs), recombination is a traditionally essential operator. Its purpose is to derive a new candidate solution from two (or more) selected parental solutions by merging their attributes. Usually, this is done in a simple way, which is heavily based on random decisions. While such an operation is computationally cheap, created offspring is often worse than respective parent solutions, and many repetitions are typically necessary for achieving improvements.

As an alternative, one can put more effort into the determination of a new solution that is constructed entirely or mainly of attributes appearing in the parents. An established example from the domain of metaheuristics following this idea is *path-relinking* [42]. In the search space, this approach traces a path from one parent to another by always only exchanging a single attribute (or, more generally, performing a simple move towards the second parent). An overall best solution found on this path is finally taken as result.

This concept can further be extended by considering not just solutions on an individual path between two parents, but the whole subspace of solutions made up of parental properties only. An optimal *merging* operation returns a best solution from this set. Identifying such a solution often is a hard optimization problem on its own, but due to the limited number of different properties appearing in the parents, it can often be solved in reasonable time in practice.

Merging has already been successfully applied multiple times. Applegate et al. [7] were one of the first and describe such an approach for the traveling salesman problem. They derive a set of diverse tours by a series of runs of an iterated local search algorithm. The edge-sets of these solutions are merged and the traveling salesman problem is finally solved to optimality on this strongly restricted graph. In this way a solution is achieved that is typically superior to the best solution of the iterated local search.

Klau et al. [50] follow a similar idea and combine a memetic algorithm with integer programming to heuristically solve the prize-collecting Steiner tree problem. The proposed algorithmic framework consists of three parts: extensive preprocessing, a memetic algorithm, and an exact branch-and-cut algorithm applied as post-optimization procedure to the merged final solutions of the memetic algorithm.

Besides the one-time application of merging to a set of heuristically determined solutions, merging can also replace the classical crossover operator in EAs. Aggarwal et al. [1] originally suggested such an approach for the independent set problem and called it *optimized crossover*. The subproblem of combining two independent sets to obtain the largest independent set in their union can be solved by an efficient algorithm.

Ahuja et al. [2] extend this concept to genetic algorithms for the quadratic assignment problem. They present a matching-based optimized crossover heuristic that finds an optimized child quickly in practice. This technique can also be applied to other assignment-type problems, as it relies on the structure of the problem rather than the objective function.

Cotta et al. [20] discuss the concept of merging in the light of a framework for hybridizing B&B with EAs. The authors recall the theoretical concepts on formal analysis (formae are generalized schemata), such as the dynastic potential of two chromosomes x and y, which is the set of individuals that only carry information contained in x and y. Based on these concepts the idea of dynastically optimal recombination is developed. This results in an operator exploring the potential of the recombined solutions using B&B, providing the best possible combination of the ancestors' features that can be attained without introducing implicit mutation. Extensive computational experiments on different benchmark sets show the usefulness of the approach.

Marino et al. [56] present an approach where a GA is combined with an exact method for the *linear assignment problem* (LAP) to solve the graph coloring problem. The LAP algorithm is incorporated into the crossover operator and generates an optimal permutation of colors within a cluster of nodes, thereby preventing the offspring from being less fit than its parents. The algorithm does not outperform other approaches, but provides comparable results. The main conclusion is that solving the LAP in the crossover operator strongly improves the performance of the GA in comparison to the GA using a classical crossover.

Clements et al. [16] propose a column generation approach in order to solve a production-line scheduling problem. Each feasible solution of the problem consists of a line-schedule for each production line. First, the *squeaky wheel optimization* (SWO) heuristic is used to generate feasible solutions to the problem. SWO is a heuristic using a greedy algorithm to construct a solution, which is then analyzed in order to find the problematic elements. Higher priorities, indicating that these elements should be considered earlier by the greedy algorithm, are assigned to them and the process restarts until a termination condition is reached. SWO is called several times in a randomized way in order to generate a set of diverse solutions. In the second phase, the line-schedules contained in these solutions are used as columns of a set-partitioning formulation for the problem, which is solved by a general purpose MIP solver. This process always provides a solution which is at least as good as, but usually better than the best solution devised by SWO. Reported results indicate that SWO performs better than a tabu search algorithm.

From a more theoretical point, Eremeev [27] studies the computational complexity of producing the best possible offspring in an optimized crossover for 0–1 ILPs. By means of efficient reductions of the merging subproblem, he shows the polynomial solvability for the maximum weight set packing problem, the minimum weight set partition problem, and for a version of the simple plant location problem.

For general mixed integer programming, Rothberg [79] describes a tight integration of an EA in a branch-and-cut based MIP solver. In regular intervals, a certain number of iterations of the EA is performed as B&B tree node heuristic. Recombination follows the idea of solution merging by first fixing all variables that are common in selected parental solutions. The values of the remaining variables are then determined by applying the MIP solver to the reduced subproblem. Mutation is performed by selecting one parent, fixing a randomly chosen set of variables, and again solving the resulting reduced subproblem by the MIP solver. Since the number of variables to be fixed is a critical parameter, an adaptive scheme is used to control it. Performed experiments indicate that this hybrid approach is able to find significantly better solutions than other heuristic methods for several very difficult MIPs. The method is now also integrated in version 10 of the commercial MIP solver CPLEX.

Last but not least, it should be pointed out that there exists a strong relation between large neighborhood search and solution merging. In fact, solution merging can also be seen as exploring a large neighborhood defined by two or more parental solutions.

10 ILP Techniques as Decoders for Indirect or Incomplete Representations

Often, candidate solutions are only indirectly or incompletely represented in metaheuristics, and an “intelligent” decoding function is applied for determining an actual, complete solution. This in particular holds for many GAs. Sometimes, ILP techniques are successfully used for the decoding step.

It is relatively straight-forward to approach a MIP by splitting it into the integer and the continuous variable parts. One can then apply a metaheuristic to optimize the integer part only; before evaluating a solution, a linear programming solver is applied in order to augment the integer part with an optimal choice of continuous variable values. Such approaches are described in conjunction with GRASP by Net and Pedroso [60] and in conjunction with tabu search by Pedroso [61].

Glover [40] suggests a parametric tabu search for heuristically solving MIPs. This approach also makes use of an underlying LP-solver to obtain complete solution candidates. The current search point is indirectly represented by the LP relaxation of the MIP plus additional *goal conditions* that restrict the domains of a subset of the integer variables. These goal conditions are, however, not directly considered as hard constraints when applying the LP-solver, but are relaxed and brought into the objective function similarly as in Lagrangian relaxation. In this way, the approach can also be applied to problems where it is hard to find any feasible integer solutions (constraint satisfaction problems). Glover suggests a variety of intensification and diversification strategies based on adaptive tabu memory for making the heuristic search more efficient.

A more problem-specific example is the hybrid GA presented by Staggemeier et al. [80] for solving a lot-sizing and scheduling problem minimizing inventory and backlog costs of multiple products on parallel machines. Solutions are represented as product subsets for each machine at each period. Corresponding optimal lot sizes are determined when the solution is decoded by solving a linear program. The approach outperforms a MIP formulation of the problem directly solved by CPLEX.

11 Multi-Stage Approaches

Some optimization approaches consist of multiple sequentially performed stages, and different techniques are applied at the individual phases.

In many real-world applications, the problem naturally decomposes into multiple levels, and if the decision variables associated to the lower level(s) have a significantly weaker impact on the objective value than the higher-level variables, it is a reasonable approach to optimize the individual levels in a strictly sequential manner. Metaheuristics and ILP techniques can be considered and in combination be applied at the individual levels.

Multi-stage approaches are sometimes even applied when such a problem decomposition is not so obvious. For example, in Sect. 9, we considered approaches, where a metaheuristic is used to derive a set of heuristic solutions and an exact technique is used for merging them. Further examples are variable fixing strategies as described in Sect. 5.4.

Tamura et al. [83] tackle a job-shop scheduling problem and start from its ILP formulation. For each variable, they take the range of possible values and partition it into a set of subranges, which are then indexed. The encoded solutions of a GA are defined so that each position represents a variable, and its value corresponds to the index of one of the subranges. The fitness of such a chromosome is calculated using Lagrangian relaxation in order to obtain a bound on the optimal solution subject to the constraints that the values of the variables fall within the represented ranges. When the GA terminates, an exhaustive search of the region identified as the most promising is carried out to produce the final solution.

Lin et al. [54] propose an exact algorithm for generating the minimal set of affine functions that describes the value function of the finite horizon partially observed Markov decision process. In the first step a GA is used to generate a set Γ of witness points, which is as large as possible. In the second step a component-wise domination procedure is performed in order to eliminate redundant points in Γ. The set generated so far does not, in general, fully describe the value function. Therefore, a MIP is solved to generate the missing points in the final third step of the algorithm. Reported results indicate that this approach requires less time than some other numerical procedures.

Another kind of sequential combination of B&B and a GA has been described by Nagar et al. [58] for a two-machine flowshop scheduling problem in which solution candidates are represented as permutations of jobs. Prior to running the GA, B&B is executed down to a predetermined depth k and suitable bounds are calculated and recorded at each node of the explicitly stored B&B tree. During the execution of the GA each partial solution up to position k is mapped onto the corresponding tree node. If the associated bounds indicate that no path below this node can lead to an optimal solution, the permutation is subjected to a mutation operator that has been specifically designed to change the early part of the permutation in a favorable way.

12 Cut and Column Generation by Metaheuristics

In cutting plane and column generation based methods, which we addressed in Sects. 3.2 and 3.3, the dynamic separation of cutting planes and the pricing of columns, respectively, is sometimes done by means of (meta-)heuristics in order to speed up the whole optimization process. We consider these hybrid approaches in the following in more detail.

12.1 Heuristic Cut Separation

In cutting plane and branch-and-cut algorithms, effective techniques are needed for deriving cuts, i.e. inequalities that are satisfied by feasible integer solutions but violated by the current solution to the LP relaxation. Although heuristic separation routines are commonly applied for this purpose, more sophisticated metaheuristics have only rarely been used.

An example is the work from Augerat et al. [8], who present a constructive algorithm, a randomized greedy method, and a tabu search for separating capacity constraints to solve a capacitated vehicle routing problem. The ILP formulation includes an exponential number of capacity constraints ensuring that for any given subset of customers S at least $\lceil \frac{d(S)}{C} \rceil$ vehicles are needed to satisfy the demand in S ($d(S)$ corresponds to the sum of the demands of the customers in set S and C is the capacity of one vehicle). A combination of a cutting plane algorithm and branch-and-bound is used to solve the problem optimally. The presented results indicate that using tabu search for identifying violated valid inequalities is promising and the use of metaheuristics in separation procedures is worth investigating.

Another example concerns the acceleration of Benders decomposition by local branching, as described by Rei et al. [78]. Benders decomposition is a promising solution approach in particular for MIPs with diagonal block structure. The basic principle is to project the MIP into the space of complicating integer variables only; real variables and the constraints involving them are replaced by corresponding constraints on the integer variables. These constraints, however, are not directly available but need to be dynamically separated in a cutting plane algorithm-like approach. According to the classical method, an optimal solution to the relaxed master problem (including only the already separated cuts) is needed and a linear program involving this solution must be solved in order to separate a single new cut. Rei et al. [78] improved this method by introducing phases of local branching on the original problem in order to obtain multiple feasible heuristic solutions. These solutions provide improved upper bounds on one hand, but also allow the derivation of multiple additional cuts before the relaxed master problem needs to be resolved. Tests on certain multicommodity flow formulations of a capacitated network design problem indicate the advantages over the traditional Benders decomposition approach.

12.2 Heuristic Column Generation

In column generation approaches and branch-and-price algorithms, it is important to have fast algorithms available for repeatedly solving the pricing subproblem, i.e. identifying a variable (column) with negative reduced costs. For many hard problems, however, this subproblem is also hard. Fast heuristics are therefore sometimes used for approaching the pricing problem. Note that it is fine when pricing in a column with negative reduced costs even when it is not one with minimum reduced costs. However, at the end of column generation it is necessary to prove that no further column with negative reduced costs exists, i.e. the pricing problem must finally be solved exactly. Otherwise, no quality guarantees can be given for the final solution of the whole column generation or branch-and-price algorithm, and they must be considered to be heuristic methods only.

Most heuristic approaches for solving pricing problems are relatively simple construction methods. More sophisticated metaheuristics have so far been used less frequently. Filho and Lorena [29] apply a heuristic column generation approach to graph coloring. A GA is used to generate initial columns and to solve the pricing problem, which corresponds to the weighted maximum independent set problem, at every iteration. Column generation is performed as long as the GA finds columns with negative reduced costs. The master problem is solved using CPLEX. Some encouraging results are shown.

Puchinger and Raidl [66, 69] describe a branch-and-price approach for the three-stage two-dimensional bin packing problem. The pricing problem corresponds to the NP-hard three-stage two-dimensional knapsack problem with additional side-constraints coming from a special branching technique. Fast column generation is performed by applying a hierarchy of four methods: (a) a greedy heuristic, (b) an EA, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX. From this hierarchy, a strategy is always only applied when all lower level methods have been tried and were not successful in finding a column with negative reduced costs. Computational experiments on standard benchmark instances document the benefits of this fine-grained approach. The combination of all four pricing algorithms in the proposed branch-and-price framework yields the best results in terms of the average objective value, the average run-time, and the number of instances solved to proven optimality.

13 Strategic Guidance of Search and Collaboration

Last but not least, we consider approaches where metaheuristics are applied in order to explicitly guide ILP techniques and collaborative combinations where metaheuristics as well as ILP techniques provide each other mutual guidance.

13.1 Guidance of ILP Search

In principle, any metaheuristic that provides incumbent solutions to a B&B-based approach might already be considered to fall into this class of approaches; see also Sect. 4. Two more sophisticated methods, which go beyond this, are the following.

French et al. [35] suggest an EA/B&B hybrid to solve general ILPs. This hybrid algorithm combines the generic B&B of the MIP solver XPRESS-MP[2] with a steady-state EA. It starts with a B&B phase, in which information from the B&B tree nodes is collected in order to derive candidate solutions which are added to the originally randomly initialized EA-population. When a certain criterion is fulfilled, the EA takes over for a certain time using the augmented initial population. After termination of the EA, its best solutions are passed back and grafted onto the B&B tree. Full control is given back to the B&B-engine after the newly added nodes had been examined to a certain degree. Reported results on instances of the maximum satisfiability problem show that this hybrid approach yields better solutions than B&B or the EA alone.

[2] http://www.dashoptimization.com/

Kotsikas and Fragakis [51] determine improved node selection strategies within B&B for solving MIPs by using genetic programming. After running B&B for a certain amount of time, information is collected from the B&B tree and used as a training set for genetic programming, which is performed to find a node selection strategy more appropriate for the specific problem at hand. The following second B&B phase then uses this new node selection strategy. Reported results show that this approach has potential, but needs to be enhanced in order to be able to compete with today's state-of-the-art node selection strategies.

13.2 Mutual Guidance

Several systems have been proposed where different optimization techniques, including metaheuristics and ILP methods, run in parallel or in an intertwined way and communicate with each other in order to provide mutual guidance.

Denzinger and Offerman [24] described a multi-agent based approach called TECHS (TEams for Cooperative Heterogenous Search). It consists of teams of one or more agents using the same search paradigm. Communication between the agents is controlled by so-called send- and receive-referees, in order to filter exchanged data. Each agent is in a cycle between searching and processing received information. In order to demonstrate the usefulness of TECHS, a system with multiple GA and B&B agents is considered for job-shop scheduling. GA and B&B agents exchange only positive information (solutions), whereas B&B agents can also exchange negative information (closed subtrees) among each other. Computational experiments show that this cooperation results in finding better solutions given a fixed time-limit and in finding solutions comparable to the ones of the best individual system alone in less total time.

Gallardo, Cotta, and Fernández [36] present another EA/B&B hybrid evaluated on the MKP. The algorithms are executed in an intertwined way and are cooperating by exchanging information. The EA provides bounds for B&B, while B&B provides best and partial solutions to the EA. In more detail, the EA is executed first until a certain convergence criterion is reached, yielding an initial bound. Then B&B is performed until it obtains an improved solution. Next, control is again given back to the EA, which possibly incorporates the new incumbent solution as well as some promising partial solutions from the ongoing B&B search into its population. Control is switched between the algorithms until a run-time limit is reached. Experimental results show that the collaborative approach yields better results than the individual techniques executed on their own.

In [37], the same authors described a refined variant of their approach, which uses beam search as truncated B&B. The method is also applied to the shortest common supersequence problem, where the results are again very encouraging.

Another cooperative approach involving a memetic algorithm and branch-and-cut has been described by Puchinger et al. [70] for the MKP. Both methods are performed in parallel and exchange information in a bidirectional asynchronous way. In addition to promising primal solutions, the memetic algorithm also receives dual variable values of certain LP relaxations and uses them for improving its repair and local improvement functions by updating the items' pseudo-utility ratios (see also Sect. 5.3).

The MALLBA project [4, 5] and its follow-up TRACER facilitate the direct development of parallel hybrid algorithms over local and wide area networks. It consists of a library of skeletons for combinatorial optimization, hiding complex parallelization and hybridization implementation details from the user. Several skeletons of exact and heuristic methods such as B&B, dynamic programming, tabu search, and GAs are available.

14 Conclusions

We have surveyed a multitude of examples where more powerful optimization systems were constructed by combining mathematical programming techniques and metaheuristics. Many very different ways exist for such hybridizations, and we have classified them into ten major methodological categories. The probably most traditional approach is to use some metaheuristic for providing high-quality incumbents and bounds to a B&B-based exact method. On the other hand, quickly solved relaxations or the primal-dual relationship are often used for guiding or narrowing the search in metaheuristics. A relatively new and highly promising stream are those methods in which B&B is modified in some way in order to follow the spirit of local search based metaheuristics. A nowadays frequently and successfully applied approach is large neighborhood search by means of ILP techniques. When extending this concept towards searching the neighborhood defined by the common and disjoint properties of two or more parental solutions, we come to solution merging approaches. Then, we have considered ILP techniques as decoders for indirect or incomplete representations. Furthermore, some problems are naturally approached by multi-stage approaches. So far less frequently applied, but in the opinion of the authors highly promising hybrid approaches are those where metaheuristics are utilized within more complex branch-and-cut and branch-and-price algorithms for cut separation and column generation, respectively. Last but not least we have considered collaborative hybrid systems in which one method provides some kind of strategic guidance for the other or even mutual guidance is achieved. As noted, some approaches from the literature can be considered to fall into several of the methodological categories we have identified.

Although a lot of experience already exists with such hybrid systems, it is usually still a tough question which algorithms and kinds of combinations are most promising for a new problem at hand. Despite the many successful examples of hybrids, the reader should also keep in mind that a more complex system does not automatically perform better than a simpler "pure" algorithm. Many less successful trials of combining mathematical programming techniques and metaheuristics also exist, but they are usually not published. The primary advice the authors are able to give for developing superior hybrid systems is to carefully study the literature looking for most successful approaches to similar problems and to adopt and eventually recombine (hybridize) their key-features. We hope that this chapter provides a good starting point and some references for this purpose.

Acknowledgements

This work is partly supported by the European RTN ADONET under grant 504438 and the "Hochschuljubiläumsstiftung" of Vienna, Austria, under contract number H-759/2005.

National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

1. C. Aggarwal, J. Orlin, and R. Tai. Optimized crossover for the independent set problem. *Operations Research*, 45:226–234, 1997.
2. R. Ahuja, J. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27:917–934, 2000.
3. R. K. Ahuja, Ö Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3): 75–102, 2002.
4. E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, J. González, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. MALLBA: Towards a combinatorial optimization library for geographically distributed systems. In *Proceedings of the XII Jornadas de Paralelismo*, pages 105–110. Editorial U.P.V., 2001.
5. E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, J. González C., León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. MALLBA: A library of skeletons for combinatorial optimisation. In B. Monien and R. Feldman, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 927–932. Springer-Verlag, Berlin, Germany, 2002.
6. F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics – Third International Workshop, HM 2006*, volume 4030 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2006.
7. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. On the solution of the traveling salesman problem. *Documenta Mathematica*, Extra Volume ICM III:645–656, 1998.

8. P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2):546–557, 1999.
9. E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980.
10. F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming, Series A*, 87(3):385–399, 2000.
11. D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
12. M. Blesa, C. Blum, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics – Second International Workshop, HM 2005*, volume 3636 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.
13. C. Blum, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics – First International Workshop, HM 2004*. Proceedings, Valencia, Spain, 2004.
14. K. Büdenbender, T. Grünert, and H.-J. Sebastian. A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem. *Transportation Science*, 34(4):364–380, 2000.
15. P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.
16. D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh. Heuristic optimization: A hybrid AI/OR approach. In A. Davenport and C. Beck, editors, *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, 1997. Held in conjunction with the Third International Conference on Principles and Practice of Constraint Programming (CP97).
17. R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*. PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK, 2000.
18. R. K. Congram, C. N. Potts, and S. L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.
19. C. Cotta. A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications*, 11(3–4):223–224, 1998.
20. C. Cotta and J. M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18:137–153, 2003.
21. E. Danna and C. Le Pape. Two generic schemes for efficient and robust cooperative algorithms. In Michela Milano, editor, *Constraint and Integer Programming*, pages 33–57. Kluwer Academic Publishers, 2003.
22. E. Danna, E. Rothberg, and C. Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming, Series A*, 102:71–90, 2005.
23. G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.
24. J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In William Porto et al., editors, *Proceedings of the 1999 Congress on Evolutionary Computation (CEC)*, volume 3, pages 2317–2324. IEEE Press, 1999.
25. I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. In Günther R. Raidl et al., editors, *Applications of Evolutionary Computation*,

volume 2611 of *Lecture Notes in Computer Science*, pages 211–223. Springer-Verlag, Berlin, Germany, 2003.

26. M. El-Abd and M. Kamel. A taxonomy of cooperative search algorithms. In Blesa Aguilera et al. [12], pages 32–41.
27. A. Eremeev. On complexity of optimized crossover for binary representations. In Dirk V. Arnold, Thomas Jansen, Michael D. Vose, and Jonathan E. Rowe, editors, *Theory of Evolutionary Algorithms*, number 06061 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
28. Ö. Ergun and J. B. Orlin. A dynamic programming methodology in very large scale neighborhood search applied to the traveling salesman problem. *Discrete Optimization*, 3(1):78–85, 2006.
29. G. Ribeiro Filho and L. A. Nogueira Lorena. Constructive genetic algorithm and column generation: an application to graph coloring. In Lui Pao Chuen, editor, *Proceedings of APORS 2000, the Fifth Conference of the Association of Asian-Pacific Operations Research Societies within IFORS*, 2000.
30. M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.
31. M. Fischetti, C. Polo, and M. Scantamburlo. Local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks*, 44(2):61–72, 2004.
32. M. Fischetti and A. Lodi. Local Branching. *Mathematical Programming, Series B*, 98:23–47, 2003.
33. M. L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1):1–18, 1981.
34. A. Frangioni. About Lagrangian methods in integer optimization. *Annals of Operations Research*, 139(1):163–193, 2005.
35. A. P. French, A. C. Robinson, and J. M. Wilson. Using a hybrid genetic algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7:551–564, 2001.
36. J. E. Gallardo, C. Cotta, and A. J. Fernández. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In Mira and Álvarez [57], pages 21–30.
37. J. E. Gallardo, C. Cotta, and A. J. Fernández. On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(1):77–83, 2007.
38. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.
39. P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.
40. F. Glover. Parametric tabu-search for mixed integer programming. *Computers & Operations Research*, 33(9):2449–2494, 2006.
41. F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, 2003.
42. F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.
43. F. Glover. Surrogate constraints. *Operations Research*, 16(4):741–749, 1968.

44. P. Hansen, J. Brimberg, N. Mladenović, and D. Urosević. Primal-dual variable neighborhood search for the simple plant location problem. *INFORMS Journal on Computing*, to appear.
45. P. Hansen and N. Mladenović. An introduction to variable neighborhood search. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics: advances and trends in local search paradigms for optimization*, pages 433–438. Kluwer Academic Publishers, 1999.
46. P. Hansen, N. Mladenović, and D. Urosević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33(10):3034–3045, 2006.
47. M. Haouaria and J. C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.
48. H. Hoos and T. Stützle. *Stochastic Local Search – Foundations and Applications*. Morgan Kaufmann, 2004.
49. B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, to appear.
50. G. W. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. R. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In K. Deb et al., editors, *Genetic and Evolutionary Computation – GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*, pages 1304–1315. Springer-Verlag, Berlin, Germany, 2004.
51. K. Kostikas and C. Fragakis. Genetic programming applied to mixed integer programming. In Maarten Keijzer et al., editors, *Genetic Programming – EuroGP 2004*, volume 3003 of *Lecture Notes in Computer Science*, pages 113–124. Springer-Verlag, Berlin, Germany, 2004.
52. E. L. Lawler and D. E. Wood. Branch and bounds methods: A survey. *Operations Research*, 4(4):669–719, 1966.
53. D. Lichtenberger. An extended local branching framework and its application to the multidimensional knapsack problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria, March 2005.
54. A. Z.-Z. Lin, J. Bean, and C. C. White. A hybrid genetic/optimization algorithm for finite horizon partially observed Markov decision processes. *Journal on Computing*, 16(1):27–38, 2004.
55. M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.
56. A. Marino, A. Prügel-Bennett, and C. A. Glass. Improving graph colouring with linear programming and genetic algorithms. In K. Miettinen, M. M. Makela, and J. Toivanen, editors, *Proceedings of EUROGEN 99*, pages 113–118, Jyväskyiä, Finland, 1999.
57. J. Mira and J. R. Álvarez, editors. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.
58. A. Nagar, S. S. Heragu, and J. Haddock. A meta-heuristic algorithm for a bi-criteria scheduling problem. *Annals of Operations Research*, 63:397–414, 1995.
59. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

60. T. Neto and J. P. Pedroso. GRASP for linear integer programming. In J. P. Sousa and M. G. C. Resende, editors, *Metaheuristics: Computer Decision Making*, Combinatorial Optimization Book Series, pages 545–574. Kluwer Academic Publishers, 2003.
61. J. P. Pedroso. Tabu search for mixed integer programming. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution*, volume 30 of *Operations Research/Computer Science Interfaces Series*, pages 247–261. Springer-Verlag, Berlin, Germany, 2005.
62. S. Pirkwieser, G. R. Raidl, and J. Puchinger. Combining Lagrangian decomposition with an evolutionary algorithm for the knapsack constrained maximum spanning tree problem. In Carlos Cotta and Jano van Hemert, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2007*, volume 4446 of *Lecture Notes in Computer Science*, pages 176–187. Springer-Verlag, Berlin, Germany, 2007.
63. D. Pisinger. An expanding-core algorithm for the exact 0–1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1995.
64. A. Plateau, D. Tachat, and P. Tolla. A hybrid search combining interior point methods and metaheuristics for 0–1 programming. *International Transactions in Operational Research*, 9:731–746, 2002.
65. M. Prandtstetter and G. R. Raidl. A variable neighborhood search approach for solving the car sequencing problem. In Pierre Hansen et al., editors, *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search*, Tenerife, Spain, 2005.
66. J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao et al., editors, *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 642–651. Springer-Verlag, Berlin, Germany, 2004.
67. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer-Verlag, Berlin, Germany, 2005.
68. J. Puchinger and G. R. Raidl. Bringing order into the neighborhoods: Relaxation guided variable neighborhood search. *Journal of Heuristics*, to appear.
69. J. Puchinger and G. R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, to appear.
70. J. Puchinger, G. R. Raidl, and M. Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *Proceedings of MIC 2005, the 6th Metaheuristics International Conference*, pages 775–780, Vienna, Austria, 2005.
71. J. Puchinger, G. R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 162–173. Springer-Verlag, Berlin, Germany, 2004.
72. J. Puchinger, G. R. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006*, volume 3906 of *Lecture Notes in Computer Science*, pages 195–208. Springer-Verlag, Berlin, Germany, 2006.

73. G. R. Raidl and H. Feltl. An improved hybrid genetic algorithm for the generalized assignment problem. In H. M. Haddadd et al., editors, *Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 990–995. ACM Press, 2004.
74. G. R. Raidl. An improved genetic algorithm for the multiconstrained 0–1 knapsack problem. In D. B. Fogel et al., editors, *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 207–211. IEEE Press, 1998.
75. G. R. Raidl. A unified view on hybrid metaheuristics. In Almeida et al. [6], pages 1–12.
76. G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation Journal*, 13(4):441–475, 2005.
77. C. Rego. RAMP: A new metaheuristic framework for combinatorial optimization. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution*, pages 441–460. Kluwer Academic Publishers, 2005.
78. W. Rei, J.-F. Cordeau, M. Gendreau, and P. Soriano. Accelerating Benders decomposition by local branching. Technical Report C7PQMR PO2006-02-X, HEC Montréal, Canada, 2006.
79. E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.
80. A. Toniolo Staggemeier, A. R. Clark, U. Aickelin, and J. Smith. A hybrid genetic algorithm to solve a lot-sizing and scheduling problem. In B. Lev, editor, *Proceedings of the 16th triannual Conference of the International Federation of Operational Research Societies*, Edinburgh, U.K., 2002.
81. E. D. Taillard, L.-M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming: A unified view of meta-heuristics. *European Journal of Operational Research*, 135:1–16, 2001.
82. E. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–565, 2002.
83. H. Tamura, A. Hirahara, I. Hatono, and M. Umano. An approximate solution method for combinatorial optimisation. *Transactions of the Society of Instrument and Control Engineers*, 130:329–336, 1994.
84. P. M. Thompson and J. B. Orlin. The theory of cycle transfers. Technical Report OR-200-89, MIT Operations Research Center, Boston, MA, 1989.
85. P. M. Thompson and H. N. Psaraftis. Cycle transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
86. M. Vasquez and J.-K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 328–333, Seattle, Washington, 2001. Morgan Kaufman.
87. M. Vasquez and Y. Vimont. Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165:70–81, 2005.
88. L. A. Wolsey. *Integer Programming*. Wiley-Interscience, 1998.
89. D. L. Woodruff. A chunking based selection strategy for integrating metaheuristics with branch and bound. In S. Voß et al., editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 499–511. Kluwer Academic Publishers, 1999.

The Relation Between Complete and Incomplete Search

Steven Prestwich

Cork Constraint Computation Centre
University College, Cork, Ireland
s.prestwich@cs.ucc.ie

Summary. This chapter compares complete and incomplete search methods, discusses hybrid approaches, contrasts modelling techniques, and speculates that the boundary between the two is more blurred than it might seem.

1 Introduction

Complete and incomplete search are two quite distinct approaches to solving combinatorial problems. Most constraint programming (CP) and mixed integer programming (MIP) solvers are complete, but there also exist software libraries for genetic algorithms and local search, and stand-alone local search algorithms for Boolean satisfiability (SAT). Both types of search algorithm are highly successful on different problems, and on some problems they are competitive with each other. But choosing between them is not easy, especially as features of the problem or the chosen model, such as symmetry, might affect the choice of algorithm.

In this chapter we compare and contrast the two search paradigms and their hybrids, discuss their strengths and weaknesses, and argue that the boundary between the two is more blurred than it might seem. The remainder of this section provides background on complete and incomplete search search, and discusses their differences. Sect. 2 surveys hybrids of complete and incomplete search. Sect. 3 uses a case study to explore the boundary between the two paradigms. Sect. 4 discusses problem symmetry and its effect on different search algorithms. Finally, Sect. 5 concludes the chapter.

It should be noted that, while effort has been made to provide a balanced account, this chapter naturally reflects the author's technical background and interests. This results in a bias towards CP, SAT, artificial intelligence (AI) and local search, at the expense of MIP, operations research and evolutionary computation.

S. Prestwich: *The Relation Between Complete and Incomplete Search*, Studies in Computational Intelligence (SCI) **114**, 63–83 (2008)
www.springerlink.com

1.1 Complete Search

Complete search algorithms have some very desirable properties. They are complete: on a feasibility problem (such as SAT) they are guaranteed either to find a solution, or to prove that there is no solution; on an optimisation problem (such as job-shop scheduling) they are guaranteed to find an optimum solution and to prove it optimum, or to prove that there is no solution. They usually have no runtime parameters to tune, making them easy to use (though the user still has to choose a problem model, and possibly branching or other heuristics).

A variety of complete search algorithms are described in standard AI texts, including best-first search, breadth-first search and iterative deepening, but CP and MIP solvers are usually based on a particular complete search algorithm: depth-first search (DFS) with chronological backtracking. DFS has in fact been known for thousands of years. It was supposedly used by Theseus to locate the Minotaur at the centre of a maze, who unwound a silk thread to locate the last choice-point, then chose the leftmost corridor that he had not already explored. DFS is *exhaustive:* it explores all possibilities, except those ruled out by (for example) constraint propagation, relaxation or some other form of reasoning. This property makes it complete. DFS is also *non-redundant:* it never explores a possibility more than once, unlike iterative deepening which is exhaustive but performs some redundant recomputation. DFS has the advantage over breadth-first search of requiring far less memory: the latter must maintain multiple states, whereas DFS can reclaim memory on backtracking. DFS can also be extended in several ways to so-called *intelligent backtracking* algorithms such as backjumping, which are able to backtrack to higher points in the search tree and thus avoid some unnecessary computation.

1.2 Incomplete Search

In contrast to DFS and other complete search algorithms, metaheuristic algorithms are almost always redundant (they may recompute the same results more than once) and incomplete. The latter property means that they may fail to return an answer on a satisfiable problem, or fail to find an optimal solution to an optimisation problem. Many modern search heuristics are designed to escape from any local minima, and have the property of *probabilistic approximate completeness* (PAC): the probability of finding a solution tends to 1 as search time tends to infinity [31]. A simple way to achieve this is to allow, with some small probability, an arbitrary move at each step; then there is a non-zero probability of making a sequence of moves that leads directly to an optimal solution. Another way is to periodically restart the search from a random state; such a state has a non-zero probability of being an optimal solution, or of being sufficiently close to one that hill-climbing will move directly to it.

Despite the PAC property, metaheuristic algorithms cannot usually prove a problem unsatisfiable. However, two recent local search algorithms [1, 59] can prove the unsatisfiability of SAT problems, though not their satisfiability. They use the general resolution rule in a greedy, nonsystematic way to search for a refutation by deriving the empty clause. The hope is that such algorithms will find refutations more quickly than backtrack search, at least on some problems, though these new algorithms have not yet found their niche. Local search algorithms with learning, such as Complete Local Search [15] and Partial-Order Dynamic Backtracking [22], can prove unsatisfiability as well as satisfiability. But the aim of these algorithms is to improve performance on satisfiable problems, not to speed up proof of unsatisfiability. Learning algorithms may also require exponential memory in the worst case, though in practice polynomial memory is often sufficient.

Metaheuristic algorithms come in many forms. They explore a search space of states, with an objective function defined on the space. Some maintain a single search state and attempt to improve it by exploring neighbourhoods, while others maintain a population of states and combine them to form new ones that are intended to be better. Metaheuristics algorithms are most naturally applied to unconstrained optimisation problems, but they can handle constraints in several ways.

1.3 Contrasting the Two Approaches

The relationship between complete and incomplete search is an area of active research. In principle the No Free Lunch Theorem [71] guarantees that no search algorithm is better than any other. However, in practice this seems to be an unprofitable observation, as both complete and incomplete search have distinct strengths and weaknesses.

It is interesting to ask: what are the essential differences between the two paradigms and can their strengths be combined? Answering these questions may lead to useful insights and interesting new hybrids. In a panel discussion on systematic versus stochastic constraint satisfaction [17] a group of experts debated which are the important properties of each class of algorithm for solving satisfiable problems, and whether properties from both classes can profitably be combined. From our point of view several very interesting points were made:

- Each type of search beats the other on some classes of problem, so a useful exercise is to identify class-superior algorithms.
- The two paradigms can be combined in a brute force way via algorithm portfolios, for example by parallel execution, but more fine-grained hybrids have greater potential. A likely advantage of incomplete search is its ability to follow gradients in the search space, whereas complete search usually ignores such gradients so that it can enumerate possibilities in a non-redundant way. An interesting research direction is therefore to allow complete search to follow local gradients [15, 22, 45].

- It may never be possible to combine all the advantages of both in a single algorithm, because there is a fundamental asymmetry between finding a solution and proving a problem unsatisfiable. To boost complete search we need more powerful inference methods, and an interesting research direction is the nonsystematic derivation of proofs [1, 59].
- Because very large problems are too hard to solve to completion, the supposed disadvantage of incompleteness may be illusory. In some real applications we can only perform an incomplete search whichever algorithm we use, so we may as well use the one that gives the best results in the shortest time – often an incomplete, metaheuristic algorithm.

These viewpoints are somewhat incompatible, showing that we are far from a consensus on this issue. But this can be seen as a healthy situation that spurs competition between different types of algorithm on hard problems.

2 Hybrid Approaches

Neither backtracking nor local search is seen as adequate for all problems. Real-world problems may have a great deal of structure and are often solved most effectively by backtrackers, which are able to exploit structure via constraint propagation. Unfortunately, backtrackers do not always scale well to very large problems, even when augmented with constraint propagation, relaxation and intelligent backtracking. For some large problems it is more efficient to use incomplete search, which tends to have superior scalability. However, incomplete search often cannot exploit structure, and is quite unsuitable for certain problems. This situation has motivated research into hybrid approaches, with the aim of combining features of both classes of algorithm in order to solve currently intractable problems.

The simplest hybrid is a parallel or distributed implementation of more than one algorithm in an *algorithm portfolio*. This approach can be augmented by learning techniques to predict the most successful algorithm to try next. This is a practical way of using the most powerful algorithms known. However, it can do no better than the best known algorithm on a given problem, whereas a more fine-grained hybrid (perhaps allowing communication between different algorithms) might do better. In this section we survey hybrid approaches. These are divided (in some cases arbitrarily) into incomplete algorithms that exploit reasoning techniques from complete search, and complete algorithms that try to improve scalability by exploiting incomplete search techniques. We also discuss what kinds of problem demand hybrid approaches.

2.1 Complete Techniques in Incomplete Search

A variety of hybrids apply incomplete search without violating constraints, or only sometimes violating them, or use constraint propagation within local search. In [13] partial assignments to key variables are generated by local search, then passed to a constraint solver that checks consistency. The EFLOP algorithm [73] uses constraint propagation to escape local minima, while allowing some constraint violation. The timetabling algorithm of [65], extended to constraint satisfaction problems, searches the space of *all* partial assignments (not only the consistent ones) using an objective function that includes a measure of constraint violation. Decision Repair [36] is designed for constraint satisfaction problems and applied to open shop problems, and is described as a generalisation of the method of [65]. It uses learning (allowing complete versions to be devised) and has heuristics such as clause weighting, a TABU list and greedy hill climbing. Weak Commitment Search [72] greedily constructs consistent partial assignments. On reaching a dead-end (from which any further assignment leads to inconsistency) it randomly restarts, and it uses learning to maintain completeness. A SAT algorithm called learn-SAT based on Weak Commitment Search is described in [64]. UnitWalk [29] is a SAT local search algorithm that uses unit propagation. It starts in the same way as a backtracker, selecting a variable, assigning a value to it, propagating where possible, and repeating, until a dead-end is reached (propagation leads to a domain wipe-out). It then restarts using a slightly different variable ordering, and possibly a modified value ordering. It has also been hybridised with standard local search techniques to improve its performance on some benchmarks. In [37] preprocessing is used to add extra constraints enforcing local consistency, which improves the performance of GSAT [67] on structured problems. In [57] relaxation is used to prune local search space on an optimisation problem, by randomising the backtracking component of a branch-and-bound algorithm. In [43] constraint propagation is used to prune candidate solutions for an Ant Colony Optimisation algorithm. The evolutionary computation literature contains several techniques for constraint handling: penalty functions, rejection of infeasible solutions, repair of infeasible solutions, decoders (separate algorithms that handle constraints and are controlled indirectly by the genotype), and coevolution between populations of solutions and constraints; see the survey of [44].

Complete search may be used to explore local search neighbourhoods. In [49] branch-and-bound is used to explore local search neighbourhoods efficiently. Large Neighbourhood Search [68] performs local search and uses backtracking to test the legality of moves. Some exponentially large neighbourhoods can be explored in polynomial time by dynamic programming [51]. A different approach is Complete Local Search [15], which uses learning to achieve completeness in a local search algorithm for SAT. Though in principle this may require exponential memory, in practice the memory requirements seem to be manageable. Local search has also been used to finding a good variable ordering, which is then used to speed up a DPLL proof of unsatisfiability [8]. This author's Incomplete Dynamic Backtracking algorithm has been applied to a variety of problems (see Sect. 3 for a description of the algorithm and its applications).

2.2 Incomplete Techniques in Complete Search

Other hybrids are based on backtrack search, but improve scalability by borrowing techniques from incomplete search. Several exploit the idea of restarts. Iterative Sampling [39] restarts a backtracker using randomised heuristics every time a dead-end is reached (so no actual backtracking takes place). Squeaky Wheel Optimization algorithm [34] operates in two search spaces: a solution space and a prioritisation space. Both searches influence each other: each solution is analysed and used to change the prioritisation, which guides the search strategy used to find the next solution, found by restarting the search. Bounded Backtrack Search [26] is a hybrid of Iterative Sampling and chronological backtracking, alternating a limited amount of chronological backtracking with random restarts. In [24] chronological or intelligent backtracking is periodically restarted with slightly randomised heuristics. Disco-Novo-GoGo [66] aims to enhance backtracking so that it can solve underconstrained problems in a time comparable to local search. It regularly restarts a backtracker with random variable ordering, learning good value orderings between restarts. Multi-Point Constructive Search [6] regularly restarts a backtracker, and maintains a small set of elite solutions. At each restart it either begins from a random state or an elite solution, depending on a probability parameter. Whenever an improved solution is found, it replaces one of the elite set. In [4] search restarts are combined with learning for solving hard, real-world SAT instances. Their algorithm is complete, since the backtrack cutoff value increases after each restart.

Other approaches aim to make backtrack search more flexible or guided. In [41] backtrack points within backtrack search are randomly selected. Restricted Backtracking [23] constructs maximum cliques using a trade-off between solution quality and search completeness. Partial Order Dynamic Backtracking [22] is a hybrid of the GSAT [67] local search algorithm and Dynamic Backtracking [21] that increases flexibility in the choice of backtracking variable. They note that to achieve total flexibility while preserving completeness would require exponential memory, and recommend a less flexible version using only polynomial memory. Local Changes algorithm [69] is a complete search strategy used to solve dynamic and static constraint satisfaction problems. It extends a consistent partial assignment to a larger one by unassigning variables that are inconsistent with a new assignment, performing the new assignment then reassigning variables to the same values where possible. Limited Discrepancy Search [27] searches the neighbourhood of a consistent partial assignment, trying neighbours in increasing order of distance from the partial assignment. It relies on the user having a “good” value selection heuristic that is likely to lead to a near-solution, which could be derived from a greedy or local search algorithm. It is complete though slightly redundant. In [11] local search is used within a complete SAT solver to select the best branching variable.

Hybrid approaches have also been designed for QBF (Quantified Boolean Formulae, a quantified generalisation of SAT). WalkQSAT [19] has two main components. The first is the QBF engine, which performs a backjumping search based on conflict and solution directed backjumping. The second is the SAT engine, which is a slightly adapted version of the WalkSAT local search algorithm used as an auxiliary search procedure to find satisfying assignments quickly. The resulting solver is incomplete as it can terminate without a definite result. WalkMinQBF [32] also has two main components. The first is a local search algorithm that attempts to find an assignment to the universal variables that is a witness for unsatisfiability. The second is a complete SAT solver that tests the truth or falsity of SAT formulae that result from assigning the universal variables. WalkMinQBF is also incomplete: it outputs *unsatisfiable* if a certificate of unsatisfiability is found, otherwise it outputs *unknown.*

2.3 Application to Structured Problems

For what kind of problems are hybrid approaches most useful? In recent SAT solver competitions, local search has performed much worse than backtracking algorithms on "structured" problems classed as *industrial* (see for example [74]). The recent GASAT genetic algorithm performed even more badly on these problems [40]. Why is this? If we could determine the cause then we might be able to improve both methods by hybridisation, possibly solving larger industrial SAT problems than is currently possible.

Most local search and population-based algorithms have a drawback besides that of incompleteness: they do not exploit the powerful pruning techniques available to backtrackers. MCHC [46] was found to perform poorly on crossword puzzles and some graph colouring problems [33], while GSAT and other more recent local search algorithms for SAT are beaten by backtrackers on problems such as quasigroup existence [76]. This makes local search unsuitable for certain problems, typically those with a great deal of structure and few solutions. However, the word *structure* is used vaguely in this context, and it is unclear exactly what types of problem local search is good at. In fact local search is the best way of solving some problems that are highly regular and thus structured in some sense (for example round-robin sports scheduling).

A possible explanation is that real-world problems often contain *dependent variables,* that is variables that are functionally dependent on others. These are known to slow down local search [38], and when they form long chains they may cause local search to take polynomial or even exponential time to solve problems that can be solved in linear time by constraint propagation or other techniques [55, 70]. An attempt to alleviate this problem is the Dagsat algorithm [38], which uses a special SAT problem representation (directed acyclic graphs) that makes variable dependencies explicit. Local search is then applied to the independent variables. The Incomplete Dynamic Backtracking (see Sect. 3) and UnitWalk [29] hybrid local search algorithms handle variable dependencies by unit propagation.

Similar observations have been made in the genetic algorithm literature. It is well known that a problem representation must be designed to minimise *epistasis,* a concept similar to variable dependency. An example of a modelling technique explicitly designed to reduce epistasis is *expansive coding* [5]. This involves splitting the task into a number of separate subproblems. By adding extra dimensions to the problem much of the gene interaction can be eliminated. Though the problem space becomes larger the problem becomes easier to solve.

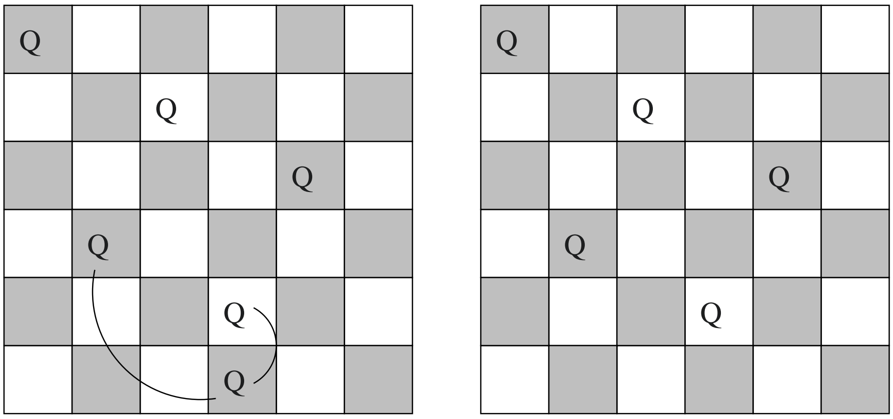
3 Boundary Between Complete and Incomplete Search

Despite the existence of many hybrids, complete and incomplete search appear to be quite different (though there are some similarities such as conflict-based heuristics). Firstly, they typically explore different search spaces: complete backtrackers explore partial variable assignments that are consistent under some form of propagation or relaxation, while incomplete algorithms usually explore inconsistent total variable assignments. Secondly, backtrackers may use randomised value and variable selection (though they often use heuristics to guide these choices) but are very restricted in their choice of backtrack variable, whereas incomplete algorithms are totally flexible and tend to be more random. Thirdly, they tend to use different heuristics: backtrackers focus on branching rules that maximise the amount of reasoning that can be used to prune the search tree, whereas incomplete algorithms try to follow local gradients.

Despite these differences, we argue that the two paradigms are more closely related than they appear. We illustrate this viewpoint using a case study: a particular hybrid called Incomplete Dynamic Backtracking (IDB) applied to the well-known N-queens problem. This hybrid was chosen partly because it is most familiar to the author, but also because it illustrates clearly that the boundary between complete and incomplete search can be blurred. We use a popular model of the N-queens problem: N variables each taking a value from the finite domain $\{1, \ldots, N\}$, each variable corresponding to a queen and row, and each value to a column. The constraints for this problem are that no two queens may attack each other (a queen attacks another if they lie on the same row, column or diagonal).

3.1 Two Forms of Local Search for N-queens

The usual local search approach is to explore a space of total assignments, that is with all N queens placed somewhere on the board. Fig. 1(i) shows a total assignment containing two constraint violations: the last queen can attack two others and vice-versa (attack is symmetrical). We may try to remove these violations, at the risk of introducing new ones, by repositioning a queen involved in a conflict – that is by reassigning a variable to another value. This is the idea behind most local search algorithms for constraint satisfaction, which has been highly successful on many problems; for example Min-Conflicts Hill Climbing [46] for binary CSPs, GSAT [67] for Boolean satisfiability, and many more recent algorithms.

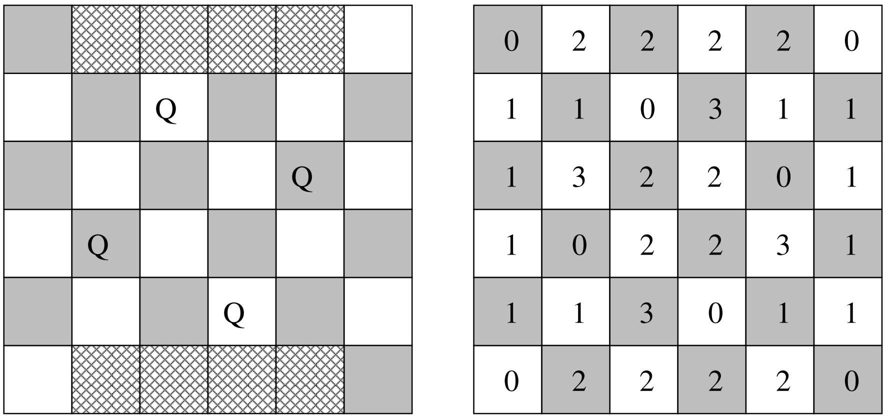


(i) total assignment (ii) consistent partial assignment

Fig. 1. Local search spaces for N-queens

However, a non-computer scientist (for example a child) might design a very different form of local search for N-queens: begin placing queens randomly in non-attacked positions; when a queen cannot be placed, randomly remove one or more placed queens (let us call this number B); continue until all queens are placed. The states of this algorithm correspond to consistent partial assignments in our model, as shown in Fig. 1(ii). We may add and remove B queens randomly, or bias the choice using heuristics. This algorithm explores a different space but is still local search. We may view the number of unassigned variables (unplaced queens) as the cost function to be minimized. No queens can be added to the state in Fig. 1(ii), which is therefore a local minimum under this function. This approach is taken by the IMPASSE class of graph colouring algorithms [47], where a consistent partial assignment is called a *coloration neighbourhood.* A heuristic we shall use below is to set the number B to a fixed positive integer before starting the search. This integer is chosen by trial and error to suit the problem to be solved, as metaheuristic algorithm parameters often are.

An advantage of this type of local search is that constraints are never violated, avoiding the need for weighted sums of cost and infeasibility functions



		Q			
				Q	
	Q				
			Q		

(i) FC-consistent partial assignment

0	2	2	2	2	0
1	1	0	3	1	1
1	3	2	2	0	1
1	0	2	2	3	1
1	1	3	0	1	1
0	2	2	2	2	0

(ii) corresponding conflict counts

Fig. 2. Forward checking in local search

in optimisation problems. Another advantage is that this form of local search can be integrated with at least some forms of constraint propagation, by using it to prune states from the search space as in backtracking search. This idea was applied to graph colouring in [58], where IMPASSE's coloration neighbourhoods were restricted to those consistent under forward checking (FC), a simple form of constraint propagation, with improved results over IMPASSE. Notice that although the state in Fig. 1(ii) is consistent, under FC it is inconsistent: the domain of the variable corresponding to the last row is empty because all its squares are attacked, so this state cannot be extended to a feasible solution. Local search enhanced with constraint propagation can prune such states. Removing the queen on row 1 makes it consistent under FC as shown in Fig. 2(i); squares on free rows that are under attack by at least one queen are shaded, and both free rows contain empty squares. However, this state cannot be extended while maintaining FC-consistency (placing a queen in column 1 or 6 on row 1 or 6 causes all squares to be under attack on the other row) so it is a local minimum.

3.2 IDB With Forward Checking

This form of search is called IDB because of a similarity to Dynamic Backtracking (DB) [21]: it can backtrack to an earlier variable without unassigning the variables in between. DB does this in a careful way to maintain completeness, and PODB [22] (cited above) allows more flexible choices but is still complete, but IDB allows totally free choices and is incomplete. However, we need a special technique to combine it with constraint propagation. To update variable domains while backtracking we maintain a *conflict count* for each variable-value pair. The conflict counts for the state in Fig. 2(i) are shown in Fig. 2(ii). A conflict count denotes the number of constraints that would be violated if the corresponding assignment were to be made. (This is distinct from local search algorithms in which conflicts *do* occur and are counted.) They are computed incrementally on assigning and unassigning a variable. Notice that the number of attacking queens *on other rows* are counted (each potential attack corresponds to a binary constraint that would be violated), and that the queens are placed on squares with zero conflict counts (corresponding to values that have not been deleted from domains by FC). Conflict counts can be generalised to non-binary constraints, and to enforce more powerful forms of constraint processing such as arc consistency.

Now that we have a local search algorithm for exploring this search space, which is usually explored by backtrack search, we are free to experiment with heuristics. An obvious direction is to try standard variable ordering heuristics, for example selecting the variable with smallest domain (optionally breaking ties by maximum forward degree in the constraint graph). An additional benefit of conflict counts is that we can obtain a domain size for assigned variables as well as unassigned ones. In the example of Fig. 2 the variables for rows 2–5 each have domain size 1, but in general these may be different. This can be used to guide the selection of variables for unassignment, for example the variable with *greatest* domain size. We mention here a value ordering heuristic that has been found useful: reassign each variable to its previous assigned value where possible, otherwise to a different value. This speeds up the rediscovery of consistent partial assignments, but to avoid stagnation it attempts to use a different (randomly chosen) value for just one variable after each dead-end.

3.3 Empirical Results

We now demonstrate empirically that IDB compares well with other algorithms on this problem, behaving like a local search algorithm or even better. In [46] the performance of DFS and local search on N-queens problems up to $N = 10^6$ were compared. They executed each algorithm 100 times for various values of N, with an upper bound of $100n$ on the number of steps (backtracks or repairs), and reported the mean number of steps and the success rate as a percentage. We reproduce the experiment up to $N = 1000$, citing their results for the Min-Conflicts hill climbing algorithm (denoted here by MCHC) and a backtracker augmented with the Min-Conflicts heuristic (denoted by MCBT). We compute results for DFS alone with random variable selection (DFS1), DFS1 with FC (DFS2), and DFS2 with dynamic variable ordering based on minimum domain size (DFS3). (Our results differ from those of [46], possibly because of algorithmic details such as random tie-breaking during variable ordering.) We also obtain results for these three algorithms with DFS replaced by IDB (denoted by IDB1, IDB2, IDB3); also for IDB3 plus the backtracking heuristic that unassigns variables with maximum domain size (IDB4), and for IDB4 plus the value ordering heuristic described above (IDB5). The IDB parameter is set to $B = 1$ for $N = 1000$ and $N = 100$, and $B = 2$ for $N = 10$ ($B = 1$ sometimes causes stagnation when $N = 10$). All results are taken from [54] and evaluated over 100 runs.

Table 1. Results for N-queens

	$N = 10$		$N = 100$		$N = 1000$	
algorithm	steps	success (%)	steps	success (%)	steps	success (%)
DFS1	81.0	100	9929	1	—	
DFS2	25.4	100	7128	39	98097	3
DFS3	14.7	100	1268	92	77060	24
IDB1	112	100	711	100	1213	100
IDB2	33.0	100	141	100	211	100
IDB3	23.8	100	46.3	100	41.2	100
IDB4	13.0	100	8.7	100	13.3	100
IDB5	12.7	100	8.0	100	12.3	100

The results in Table 1 show that replacing DFS by IDB greatly boosts scalability in three cases: the simple backtracking algorithm, backtracking with FC, and the same algorithm with dynamic variable ordering. Even the basic IDB algorithm scales much better than all the DFS algorithms, and IDB3 performs like MCHC. The additional backtracking and value ordering heuristics further boost performance, making IDB the best reported algorithm in terms of backtracks. It also compares well with Weak Commitment Search [72] which requires approximately 35 steps for large N [50]. In terms of execution time IDB is also efficient, each backtrack taking a time linear in the problem size.

It should be noted that IDB is not the only backtracker to perform like local search on N-queens. Similar results were obtained by MCBT and others because of their Min-Conflicts value ordering heuristic. In MCBT an initial total assignment I is generated by the Min-Conflicts heuristic and used to guide DFS in two ways. Firstly, variables are selected for assignment on the basis of how many violations they cause in I. Secondly, values are tried in ascending order of number of violations with currently unassigned variables. This *informed backtracking* algorithm performs almost identically to MCHC on N-queens and is complete. However, MCBT is still prone to the same drawback as most backtrackers: a poor choice of assignment high in the search tree will still take a very long time to recover from. IDB is able to modify earlier choices (as long as the B parameter is sufficiently high) so it can recover from poor early decisions. This difference is not apparent on N-queens, but it is significant on problems for which value ordering heuristics are of little help. In fact IDB has performed very well on other problems: an IDB-modified constraint solver for Golomb rulers out-performed the systematic original, and also a genetic algorithm [53]; an IDB-modified branch-and-bound algorithm for a hard optimisation problem outperformed the systematic original, and all known local search algorithms at the time [52]; and competitive results were obtained on DIMACS maximum clique benchmarks using FC with IDB [53].

3.4 Blurring the Boundary

The point of describing IDB in detail was to motivate the following question: should IDB and similar hybrids be classed as backtrackers, as local search, as both, or as hybrids? IDB may appear to be simply an inferior version of DB, sacrificing the important property of completeness to no good purpose. A counterexample to this view is the N-queens problem, on which DB performs like DFS [33], whereas IDB performs like local search. Another counterexample is the random 3-SAT problem, on which DB is slower than chronological backtracking (though a modified DB is similar) [3] while IDB again performs like local search. We claim that IDB is in fact a local search algorithm, as described above when discussing alternative forms of local search for N-queens. Our view is that it stochastically explores a space of consistent partial assignments, which is usually explored by backtrack search, while trying to minimise an objective function: the number of unassigned variables. The forward local moves are variable assignments, while the backward local moves are unassignments (backtracks). Under this view, complete search is distinguishable from incomplete search only by its completeness: the fact that they usually explore different search spaces and use different heuristics is merely a matter of technology.

An interesting possibility is to combine all forms of search algorithm into a single, unified framework. Such a framework would lead in a natural way to new hybrids combining techniques from different fields, and would be an antidote to the current proliferation of software systems. This is the aim of [30] in which a wide variety of techniques are generalised into a *search-infer-and-relax* algorithm. In this algorithm, *search* enumerates restrictions of the problem via techniques such as branching, neighbourhood search and subproblem generation. The motivation is to examine a problem that may be easier to solve, for example by setting a binary variable to 0, or by solving a subproblem in Benders decomposition [7]. The algorithm may *infer* cutting planes, domain filtering or nogood generation, which are all constraints that were implicit in the original problem. *Relaxation* sets bounds on the objective function, a common technique being the relaxation of the integrality constraints in a mixed integer program in order to find a lower bound in a minimisation problem. This framework generalises CP solvers, branch-and-bound algorithms, Benders decomposition, local search, GRASP, and hybrids of these, though it does not yet include population-based search.

4 Problem Symmetry

In this section we discuss an important property that is handled differently in complete and incomplete search: *problem symmetry*. This property is exhibited by many combinatorial problems, for example the N-queens problem has eight symmetries: each solution may be rotated by 0, 90, 180 or 270 degrees, each rotation optionally followed by a reflection. A more practical example arises in planning with interchangeable resources, where there is little point in exploring multiple versions of essentially the same plan. Interestingly, it has been found that problem symmetry has different effects on different forms of search.

4.1 Symmetry and Complete Search

Problem symmetry makes the search space larger than necessary for complete search, sometimes slowing it down dramatically, and considerable research has recently been devoted to *symmetry breaking* techniques for reducing search space size. This is especially true in CP and SAT but also in MIP. Probably the simplest and most popular approach is to add constraints to the problem formulation, so that each equivalence class of solutions to the original problem corresponds to a single solution in the new problem. Symmetries may be detected in propositional satisfiability (SAT) encodings [12] or lifted (quantified) versions of constraint satisfaction problems (CSPs) [35], and good results are reported. A formal framework for this approach is given in [62].

The benefit of symmetry breaking for reducing the search space size is clear, though its overheads are sometimes greater than its benefits. However, it is known that symmetry breaking does not necessarily aid the search for a single solution. It is remarked in [20] that the more easily-found solutions may inadvertently be eliminated, and in [28, 63] that symmetry breaking does not always improve performance. Another approach, designed partly to avoid these drawbacks, is to detect and exploit symmetries dynamically during search. An ordering can be defined on solutions, and the search restricted to the first solutions under this ordering within each equivalence class [9]; constraints may be posted [2, 20] or choices pruned [14] at each branch point in the search tree; nogoods may be added during search [16]; and search may be guided towards subspaces with many non-symmetric states [42].

4.2 Symmetry and Population-Based Search

Problem symmetry can also be harmful for population-based search, but for a different reason than for complete search. A symmetric optimisation problem has multiple optimum solutions that are symmetrically equivalent. A genetic algorithm may try to approach more than one of these optima simultaneously, via different population members. But recombination applied to these members may yield offspring with very poor fitness. For example in a graph 3-colouring problem on a clique of 3 vertices, two solutions are (red,green,blue) and (green,blue,red). But single-point crossover between the second and third genes yields the non-solution (red,green,red). This effect can be alleviated by designing more complex genetic operators and problem models [18] or by clustering techniques [48]. Whereas in the CP literature, *symmetry breaking* means adding constraints to exclude symmetric solutions, in the genetic algorithm literature it refers to a clustering of the population within a region of the search space. The risk of this form of symmetry breaking is that genetic diversity may be lost if the clustering effect is too strong [61].

4.3 Symmetry and Local Search

Perhaps surprisingly, recent research has shown that symmetry is *not* harmful for local search, in contrast to complete and population-based search. Moreover, adding symmetry breaking constraints to a model often harms local search performance [56]. The reasons are unclear, but an obvious effect of symmetry breaking is that it reduces the number of solutions to the problem, and therefore the *solution density* of the local search space (defined as the number of solutions divided by the number of possible search states). The idea that greater solution density makes a problem easier to solve seems so natural that it was taken as axiomatic in [25]. It was also pointed out by [10] that one might expect a search algorithm to solve a (satisfiable) problem more quickly if it has higher solution density. In detailed experiments on random problems across solvability phase transitions they did find a correlation between local and backtrack search performance and the number of solutions, though not a simple one. Problem features other than solution density also seem to affect search cost. For example [72] showed that adding more constraints to take a random problem beyond the phase transition removes local minima, making them easier for local search algorithms to solve despite removing solutions.

It was shown by [60], both empirically and by analysis, that symmetry breaking constraints have two distinct negative effects on a local search space: they increase the relative size of local optima, and they reduce the relative size of global basins of attraction. These effects were observed using two different symmetry breaking methods: using symmetry breaking constraints to modify an objective function, and using them to restrict the search space. They also showed that the bad effects of symmetry breaking are reduced by using more complex local search algorithms with good diversification strategies.

The effect of symmetry breaking on local search suggests that these two techniques should not be combined. This is an advantage for highly symmetric problems, for which symmetry breaking may be costly in both time and space. It also suggests a novel modelling technique: artificially adding symmetry to a model in order to improve local search performance. Some with this technique success was reported by [56], and the idea was generalised to artificially increasing the number of dominated solutions in [58].

5 Conclusion

We have seen that complete and incomplete search have complementary strengths and weaknesses, and that their strengths can to some extent be combined in hybrid algorithms. No hybrid currently possesses all strengths, except possibly an algorithm portfolio – and this approach cannot do better than the best algorithm in the portfolio, while novel hybrids can sometimes outperform all other known algorithms.

Some hybrids blur the distinction between the two search paradigms, and in the future it may be possible to generalise them to a more abstract form of search algorithm with a wide range of techniques that can be combined in arbitrary ways. However, to obtain efficient performance the different forms of search must handle features, such as problem symmetry and functionally dependent variables, in different ways. Achieving this within a single framework is likely to be challenging, but it is this sort of challenge that makes the field so interesting.

References

1. G. Audemard, L. Simon. GUNSAT: A Greedy Local Search Algorithm for Unsatisfiability. Poster paper, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence,* 2007.
2. R. Backofen, S. Will. Excluding Symmetries in Constraint-Based Search. *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1713, Springer-Verlag 1999, pp. 73–87.
3. A. B. Baker. The Hazards of Fancy Backtracking. *Proceedings of the Twelfth National Conference on Artificial Intelligence,* vol. 1, AAAI Press, 1994, pp. 288–293.
4. L. Baptista, J. P. Marques-Silva. Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1894, Springer-Verlag 2000, pp. 489–494.
5. D. Beasley, D. R. Bull, R. R. Martin. Reducing Epistasis in Combinatorial Problems by Expansive Coding. *Fifth International Conference on Genetic Algorithms,* 1993, pp. 400–407.
6. J. C. Beck. Solution-Guided, Multi-Point Constructive Search. *Journal of Artificial Intelligence Research* 29:49–77, 2007.
7. J. F. Benders. Partitioning Procedures for Solving Mixed Variables Programming Problems. *Numerische Mathematik* 4:238–252, 1962.
8. F. Boussemart, F. Hemery, C. Lecoutre, L. Saïs. Boosting Systematic Search by Weighting Constraints. *Proceedings of the Sixteenth European Conference on Artificial Intelligence,* IOS Press, 2004, pp. 146–150.
9. C. A. Brown, L. Finkelstein, P. W. Purdom Jr. Backtrack Searching in the Presence of Symmetry. T. Mora (ed.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. *Lecture Notes in Computer Science* vol. 357, Springer-Verlag 1988, pp. 99–110.

10. D. Clark, J. Frank, I. Gent, E. MacIntyre, N. Tomov, T. Walsh. Local Search and the Number of Solutions. *Proceedings of the Second International Conference on Principles and Practices of Constraint Programming,* 1996, pp. 119–133.
11. J. M. Crawford. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. *Second DIMACS Challenge: Cliques, Coloring, and Satisfiability,* October 1993, Rutgers University, NJ, USA.
12. M. Crawford, M. Ginsberg, E. Luks, A. Roy. Symmetry Breaking Predicates for Search Problems. *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning,* 1996, pp. 148–159.
13. B. De Backer, V. Furnon, P. Kilby, P. Prosser, P. Shaw. Local Search in Constraint Programming: Application to the Vehicle Routing Problem. *Proceedings of the CP'97 Workshop on Industrial Constraint-Directed Scheduling,* 1997.
14. T. Fahle, S. Schamberger, M. Sellman. Symmetry Breaking. *Proceedings of the Seventh International Conference on Principles and Practices of Constraint Programming, Lecture Notes in Computer Science* vol. 2239, Springer-Verlag, 2001, pp. 93–107.
15. H. Fang, W. Ruml. Complete Local Search for Propositional Satisfiability. *Proceedings of the Nineteenth National Conference on Artificial Intelligence,* AAAI Press, 2004, pp. 161–166.
16. F. Focacci, M. Milano. Global Cut Framework for Removing Symmetries. *Proceedings of the Seventh International Conference on Principles and Practices of Constraint Programming, Lecture Notes in Computer Science* vol. 2239, Springer-Verlag 2001, pp. 77–92.
17. E. C. Freuder, R. Dechter, M. L. Ginsberg, B. Selman, E. Tsang. Systematic Versus Stochastic Constraint Satisfaction. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* 2027–2032, Morgan Kaufmann, 1995.
18. P. Galinier, J. K. Hao. Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization* 3(4):379–397, 1999.
19. I. P. Gent, H. H. Hoos, A. G. D. Rowley, K. Smyth. Using Stochastic Local Search to Solve Quantified Boolean Formulae. *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, Springer-Verlag, 2003, pp. 348–362.
20. I. P. Gent, B. Smith. Symmetry Breaking During Search in Constraint Programming. *Proceedings of the Fourteenth European Conference on Artificial Intelligence,* 2000, pp. 599–603.
21. M. L. Ginsberg, Dynamic Backtracking, *Journal of Artificial Intelligence Research* 1:25–46, 1993.
22. M. L. Ginsberg, D. McAllester. GSAT and Dynamic Backtracking. *Proceedings of the International Conference on Principles of Knowledge and Reasoning,* 1994, pp. 226–237.
23. M. K. Goldberg, R. D. Rivenburgh. Constructing Cliques Using Restricted Backtracking. D. S. Johnson, M. A. Trick (eds.), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol. 26, American Mathematical Society, 1996, pp. 89–102.
24. C. P. Gomes, B. Selman, H. Kautz. Boosting Combinatorial Search Through Randomization. *Proceedings of the Fifteenth National Conference on Artificial Intelligence,* AAAI Press, 1998, pp. 431–437.

25. Y. Hanatani, T. Horiyama, K. Iwama. Density Condensation of Boolean Formulas. *Proceedings of the Sixth International Conference on the Theory and Applications of Satisfiability Testing,* Santa Margherita Ligure, Italy, 2003, pp. 126–133.
26. W. D. Harvey. Nonsystematic Backtracking Search. PhD thesis, Stanford University, 1995.
27. W. D. Harvey, M. L. Ginsberg. Limited Discrepancy Search. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* Morgan Kaufmann, 1995, pp. 607–615.
28. M. Henz. Constraint Programming – An Oz Perspective. *Tutorial at the Fifth Pacific Rim International Conferences on Artificial Intelligence,* 1998, NUS, Singapore, November 1998.
29. E. A. Hirsch, A. Kojevnikov. Solving Boolean Satisfiability Using Local Search Guided by Unit Clause Elimination. *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2239, Springer-Verlag, 2001, pp. 605–609.
30. J. N. Hooker. A Search-Infer-and-Relax Framework for Integrating Solution Methods. Roman Bartk and Michela Milano, eds., *Proceedings of the Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 3524, Springer-Verlag, 2005, pp. 243–257.
31. H. H. Hoos, T. Stützle. Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco, CA, USA, 2004.
32. Y. Interian, G. Corvera, B. Selman, R. Williams. Finding Small Unsatisfiable Cores to Prove Unsatisfiability of QBFs. *Proceedings of the Ninth International Symposium on AI and Mathematics,* 2006.
33. A. K. Jonsson, M. L. Ginsberg. Experimenting with New Systematic and Nonsystematic Search Techniques. *Proceedings of the AAAI Spring Symposium on AI and NP-Hard Problems,* Stanford, California, 1993.
34. D. E. Joslin, D. P. Clements. Squeaky Wheel Optimization. *Journal of Artificial Intelligence Research* 10:353–373, 1999.
35. D. Joslin, A. Roy. Exploiting Symmetry in Lifted CSPs. *Proceedings of the Fourteenth National Conference on Artificial Intelligence,* American Association for Artificial Intelligence 1997, pp. 197–203.
36. N. Jussien and O. Lhomme. Local Search With Constraint Propagation and Conflict-Based Heuristics. *Artificial Intelligence* 139(1):21–45, 2002.
37. K. Kask, R. Dechter. GSAT and Local Consistency. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* Morgan Kaufmann 1995, pp. 616–622.
38. H. Kautz, D. McAllester, B. Selman. Exploiting Variable Dependency in Local Search. *Poster Sessions, Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence,* 1997.
39. P. Langley. Systematic and Nonsystematic Search Strategies. *Artificial Intelligence Planning Systems: Proceedings of the First International Conference,* 1992.
40. F. Lardeux, F. Saubion, J.-K. Hao. GASAT: a genetic local search algorithm for the satisfiability problem. *Evolutionary Computation* 14(2):223–253, 2006.
41. I. Lynce, J. P. Marques-Silva. Random Backtracking in Backtrack Search Algorithms for Satisfiability. *Discrete Applied Mathematics* 155(12):1604–1612, 2007.

42. P. Meseguer, C. Torras. Exploiting Symmetries Within Constraint Satisfaction Search. *Artificial Intelligence* 129(1–2):133–163, 2001.
43. B. Meyer, A. Ernst, Integrating ACO and Constraint Propagation. *Proceedings of the Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science* vol. 3172, Springer-Verlag, 2004, pp. 166–177.
44. Z. Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. *Proceedings of the Fourth Annual Conference on Evolutionary Programming* 1995, pp. 135–155.
45. M. Milano, A. Roli. On the Relation Between Complete and Incomplete Search: An Informal Discussion. *Proceedings of the Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems,* le Croisic, France, 2002, pp. 237–250.
46. S. Minton, M. D. Johnston, A. B. Philips, P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58(1–3):160–205, 1992.
47. C. Morgenstern, Distributed Coloration Neighborhood Search. D. S. Johnson, M. A. Trick (eds.), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol. 26, American Mathematical Society, 1996, pp. 335–357.
48. M. Pelikan and D. E. Goldberg. Genetic Algorithms, Clustering, and the Breaking of Symmetry. *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature,* 2000.
49. G. Pesant, M. Gendreau. A View of Local Search in Constraint Programming. *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1118, Springer-Verlag, 1996, pp. 353–366.
50. D. G. Pothos, E. B. Richards. An Empirical Study of Min-Conflict Hill Climbing and Weak Commitment Search. *CP'95 Workshop on Studying and Solving Really Hard Problems,* 1995, pp. 140–146.
51. C. N. Potts, S. L. van de Velde. Dynasearch – Iterative Local Improvement by Dynamic Programming: Part I, the Travelling Salesman Problem. Technical report, University of Twente, The Netherlands, 1995.
52. S. D. Prestwich. A Hybrid Search Architecture Applied to Hard Random 3-SAT and Low-Autocorrelation Binary Sequences. *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1894, Springer-Verlag, 2000, pp. 337–352.
53. S. D. Prestwich. Trading Completeness for Scalability: Hybrid Search for Cliques and Rulers. *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems,* 2001, pp. 159–174.
54. S. D. Prestwich. Local Search and Backtracking vs Non-Systematic Backtracking. *AAAI Fall Symposium on Using Uncertainty within Computation,* Technical report FS-01-04, AAAI Press, 2001, pp. 109–115.
55. S. D. Prestwich. SAT Problems With Chains of Dependent Variables. *Discrete Applied Mathematics* 3037:1–22, Elsevier, 2002.
56. S. D. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research* 118:137–150, Kluwer Academic Publishers, 2003.

57. S. D. Prestwich. Exploiting Relaxation in Local Search. *Proceedings of the First International Workshop on Local Search Techniques in Constraint Satisfaction,* Toronto, Canada, 2004.
58. S. D. Prestwich. Increasing Solution Density by Dominated Relaxation. *Proceedings of the Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems,* 2005, pp. 1–13.
59. S. D. Prestwich, Ines Lynce. Local Search for Unsatisfiability. *Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 4121, Springer-Verlag, 2006, pp. 283–296.
60. S. D. Prestwich, A. Roli. Symmetry Breaking and Local Search Spaces. *Proceedings of the Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 3524, Springer-Verlag, 2005, pp. 273–287.
61. A. Prügel-Bennett. Symmetry Breaking in Population-Based Optimization. *IEEE Transactions on Evolutionary Computation* 8(1):63–79, 2004.
62. J.-F. Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. J. Komorowski, Z. W. Ras (eds.), Methodologies for Intelligent Systems, *Proceedings of the International Symposium on Methodologies for Intelligent Systems, Lecture Notes in Computer Science* vol. 689, Springer-Verlag 1993, pp. 350–361.
63. J.-C. Régin. Minimization of the Number of Breaks in Sports League Scheduling Problems Using Constraint Programming. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol. 57, 2001.
64. E. T. Richards, B. Richards. Non-Systematic Search and No-Good Learning. *Journal of Automated Reasoning* 24(4):483–533, 2000.
65. A. Schaerf. Combining Local Search and Look-Ahead for Scheduling and Constraint Satisfaction Problems. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence,* Morgan Kaufmann, 1997, pp. 1254–1259.
66. M. Sellmann, C. Ansótegui. Disco-Novo-GoGo: Integrating Local Search and Complete Search with Restarts. *Proceedings of the Twenty First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference,* AAAI Press, 2006.
67. B. Selman, H. Levesque, D. Mitchell. A New Method for Solving Hard Satisfiability Problems. *Proceedings of the Tenth National Conference on Artificial Intelligence,* MIT Press, 1992, pp. 440–446.
68. P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. *Proceedings of the Principles and Practice of Constraint Programming, Proceedings of the Fourth International Conference, Lecture Notes in Computer Science* vol. 1520, Springer-Verlag, 1998, pp. 417–431.
69. G. Verfaillie, T. Schiex. Solution Reuse in Dynamic Constraint Satisfaction Problems. *Proceedings of the Twelfth National Conference on Artificial Intelligence,* AAAI Press 1994, pp. 307–312.
70. W. Wei, B. Selman. Accelerating Random Walks. *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer-Verlag, 2002, pp. 216–232.
71. D. H. Wolpert, W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation* 1(1):67–82, 1997.

72. M. Yokoo. Weak-Commitment Search for Solving Constraint Satisfaction Problems. *Proceedings of the Twelfth National Conference on Artificial Intelligence* 313–318, AAAI Press, 1994.
73. N. Yugami, Y. Ohta and H. Hara, Improving repair-based constraint satisfaction methods by value propagation, *Proceedings of the Twelfth National Conference on Artificial Intelligence* vol. 1, AAAI Press, 1994, pp. 344–349.
74. E. Zarpas. Back to the SAT05 Competition: an a Posteriori Analysis of Solver Performance on Industrial Benchmarks. *Journal on Satisfiability, Boolean Modeling and Computation* 2:229–237, 2006, research note.
75. W. Zhang. Depth-First Branch-and-Bound versus Local Search: A Case Study. *Proceedings of the Seventeenth National Conference on Artificial Intelligence,* 2002, pp. 930–935.
76. H. Zhang, M. E. Stickel. Implementing the Davis-Putnam Method. *Journal of Automated Reasoning* 24(1–2):77–296, 2000.
77. J. Zhang, H. Zhang. Combining Local Search and Backtracking Techniques for Constraint Satisfaction. *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Conference on Innovative Applications of Artificial Intelligence,* AAAI Press / The MIT Press, 1996, pp. 369–374.

Hybridizations of Metaheuristics With Branch & Bound Derivates

Christian Blum[1], Carlos Cotta[2], Antonio J. Fernández[2], José E. Gallardo[2], and Monaldo Mastrolilli[3]

[1] ALBCOM research group
Universitat Politècnica de Catalunya
cblum@lsi.upc.edu

[2] Dept. Lenguajes y Ciencias de la Computación
Universidad de Málaga
{ccottap,afdez,pepeg}@lcc.uma.es

[3] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
monaldo@idsia.ch

Summary. An important branch of hybrid metaheuristics concerns the hybridization with branch & bound derivatives. In this chapter we present examples for two different types of hybridization. The first one concerns the use of branch & bound features within construction-based metaheuristics in order to increase their efficiancy. The second example deals with the use of a metaheuristic, in our case a memetic algorithm, in order to increase the efficiancy of branch & bound, respectively branch & bound derivatives such as beam search. The quality of the resulting hybrid techniques is demonstrated by means of the application to classical string problems: the longest common subsequence problem and the shortest common supersequence problem.

1 Introduction

One of the basic ingredients of an optimization technique is a mechanism for exploring the search space, that is, the space of valid solutions to the considered optimization problem. Algorithms belonging to the important class of *constructive optimization techniques* tackle an optimization problem by exploring the search space in form of a tree, a so-called *search tree.* The search tree is generally defined by an underlying solution construction mechanism. Each path from the root node of the search tree to one of the leaves corresponds to the process of constructing a candidate solution. Inner nodes of the tree can be seen as partial solutions. The process of moving from an inner node to one of its child nodes is called a solution construction step, or extension of a partial solution.

C. Blum et al.: *Hybridizations of Metaheuristics With Branch & Bound Derivates*, Studies in Computational Intelligence (SCI) **114**, 85–116 (2008)
www.springerlink.com

The class of constructive optimization techniques comprises approximate methods as well as complete methods. Recall that complete algorithms are guaranteed to find for every finite size instance of a combinatorial optimization problem an optimal solution in bounded time. This is in contrast to incomplete methods such as heuristics and metaheuristics where we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. A prominent example of a deterministic constructive heuristic is a *greedy heuristic*. Greedy heuristics make use of a weighting function that gives weights to the child nodes of each inner node of the search tree. At each construction step the child node with the highest weight is chosen.

Apart from greedy heuristics, the class of constructive optimization techniques also includes metaheuristics such as ant colony optimization (ACO) [12] and greedy randomized adaptive search procedures (GRASP) [13].[1] They are iterative algorithms that employ repeated probabilistic (randomized) solution constructions at each iteration. For each child node of an inner node of the tree they compute the probability of performing the corresponding construction step. These probabilities may depend on weighting functions and/or the search history of the algorithm. They are sometimes called *transition probabilities* and define a probability distribution over the search space. In GRASP, this probability distribution does not change during run-time, while in ACO the probability distribution is changed during run-time with the aim of biasing the probabilistic construction of solutions towards areas of the search space containing high quality solutions.

In addition to the methods mentioned above, the class of constructive optimization techniques also includes complete algorithms such as *backtracking* [36] and *branch & bound* [25]. A common backtracking scheme is implemented in the depth-first search (DFS) algorithm. The un-informed version of DFS starts from the root node of the search tree and progresses by always moving to the best unexplored child of the current node, going deeper and deeper until a leaf node is reached. Then the search backtracks, returning to the most recently visited node of which remain unexplored children, and so on. This systematic search method explicitly visits all possible solutions exactly once.

Branch & bound algorithms belong to the class of implicit enumeration techniques. The branch & bound view on the search tree is slightly different to that exhibited by the algorithms mentioned before. More specifically, the subtree rooted at an inner node of the search tree is seen as a subspace of the search space. Accordingly, the subspaces represented by the subtrees rooted at the children of an inner node consitute a partition of the subspace that is

[1] See Chapter 1 of this book for a comprehensive introduction to metaheuristics.

represented by the inner node itself. The partitioning of the search space is called branching. A branch & bound algorithm produces for each inner node of the search tree an upper bound as well as a lower bound of the objective function values of the solutions contained by the corresponding subspace. These bounds are used to decide if the whole subspace can be discarded, or if it has to be further partitioned. As in backtracking, there are different schemes such as depth-first search or breadth-first search for traversing over the search tree.

An interesting heuristic version of a breadth-first branch & bound is *beam search* [33]. Instead of considering all nodes of a certain level of the search tree, beam search restricts the search to a certain number of nodes based on the bounding information (lower bounds for minimization, and upper bounds for maximization).

Each of the algorithms mentioned above has advantages as well as disadvantages. Greedy heuristics, for example, are usually easy to implement and fast in execution. However, the obtained solution quality is often not sufficient. Metaheuristics, on the other side, can find good solutions in a reasonable amount of time, without providing performance guarantees. However, metaheuristics can generally not avoid visiting the same solution more than once, which might lead to a waste of computation time. Finally, complete techniques guarantee to find an optimal solution. However, a user might not be prepared to accept overly large running times. In recent years it has been shown that a hybridization of concepts originating from different types of algorithms can often result in more efficient techniques. For example, the use of backtracking in metaheuristics is relatively wide-spread. Examples of their use in construction-based metaheuristics are [2, 3, 9]. Backtracking is also used in evolutionary algorithms (see, for example, [10, 24]), and even in tabu search settings [32]. The hybridization of metaheuristics with branch & bound (respectively, beam search) concepts is rather recent. We distinguish between two different lines of hybridization. On one side, it is possible to use branch & bound concepts within construction-based metaheuristics in order to increase the efficiency of the metaheuristics search process. On the other side, metaheuristics can be used within branch & bound in order to reduce the space and time consumption of branch & bound. This chapter is dedicated to outline representative examples of both types of hybrid algorithms. The reader interested in a broader discussion on the combination of metaheuristics and exact techniques is referred to [34].

2 Using Branch & Bound Concepts Within Construction-Based Metaheuristics

Recent hybrid metaheuristics include some important features that are inspired by deterministic branch & bound derivatives such as beam search:

1. Bounding information is used for evaluating partial solutions; sometimes also for choosing among different partial solutions, or discarding partial solutions.
2. The extension of partial solutions is allowed in more than one way. The number of nodes which can be selected at each search tree level is hereby limited from above by a parametric constraint, resulting in parallel and non-independent solution constructions.

This type of hybrid algorithm includes probabilistic beam search (PBS) [6], Beam-ACO algorithms [4, 5], and approximate and non-deterministic tree search (ANTS) procedures [27, 28, 29].[2] These works give empiricial evidence of the usefulness of including the two features mentioned above in the construction process of construction-based metaheuristics.[3]

In the following we first give a motivation of why the above mentioned branch & bound features should be incorporated in construction-based metaheuristics. Afterwards, we present some representative approaches.

2.1 A Tree Search Model

The following tree search model captures the essential elements common to all constructive procedures. In general, we are given an optimization problem $\mathcal{P}$ and an instance x of $\mathcal{P}$. Typically, the search space S_x is exponentially large in the size of the input x. Without loss of generality we intend to maximize the objective function $f : S_x \mapsto \mathbb{R}^+$. The optimization goal is to find a solution $y \in S_x$ to x with $f(y)$ as great as possible. Assume that each element $y \in S_x$ can be viewed as a composition of $l_{y,x} \in \mathbb{N}$ elements from a set Σ. From this point of view, S_x can be seen as a set of strings over an alphabet Σ. Any element $y \in S_x$ can be constructed by concatenating $l_{y,x}$ elements of Σ.

The following method for constructing elements of S_x is instructive: A solution construction starts with the empty string ϵ. The construction process consists of a sequence of construction steps. At each construction step, we select an element of Σ and append it to the current string t. The solution construction may end for two reasons. First, it may end in case t has no feasible extensions. This happens in case t is already a complete solution, or when no solution of S_x has prefix t. Second, a solution construction ends in case of available upper bound information that indicates that each solution with prefix t is worse than any solution that is already known. Henceforth we denote the upper bound value of a partial solution t by $\mathrm{UB}(t)$.

[2] Note that the algorithms presented in [27, 28] only use the first one of the features mentioned above.

[3] Related work that is not subject of this chapter can be found, for example, in [20].

Algorithm 7 Solution construction: SC($\hat{f}$)

```
input: the best known objective function value $\hat{f}$ (which might be 0)
initialization: $v := v_0$
while $|C(v)| > 0$ and $v \neq$ NULL do
  $w :=$ ChooseFrom($C(v)$)
  if UB($w$) $< \hat{f}$ then
    $v :=$ NULL
  else
    $v := w$
  end if
end while
output: $v$ (which is either a complete solution, or NULL)
```

The application of such an algorithm can be equivalently described as a walk from the root v_0 of the corresponding search tree to a node at level $l_{y,x}$. The search tree has nodes for all $y \in S_x$ and for all prefixes of elements of S_x. The root of the tree is the empty string, that is, v_0 corresponds to ϵ. There is a directed arc from node v to node w if w can be obtained by appending an element of Σ to v. Note that henceforth we identify a node v of the search tree with its corresponding string t. We will use both notations interchangeably. The set of nodes that can be reached from a node v via directed arcs are called the children of v, denoted by $C(v)$. Note, that the nodes at level i correspond to strings of length i. If w is a node corresponding to a string of length $l > 0$ then the length $l-1$ prefix v of w is also a node, called the father of w denoted by $\mathcal{F}(w)$. Thus, every $y \in S_x$ corresponds to exactly one path of length $l_{y,x}$ from the root node of the search tree to a specific leaf. The above described solution construction process is pseudo-coded in Algorithm 7. In the following we assume function ChooseFrom($C(v)$) of this algorithm to be implemented as a probabilistic choice function.

2.2 Primal and Dual Problem Knowledge

The analysis provided in the following assumes that there is a unique optimal solution, represented by leaf node v_d of the search tree, also referred to as the target node. Let us assume that – without loss of generality – the target node v_d is at the maximum level $d \geq 1$ of the search tree. A probabilistic constructive optimization algorithm is said to be *successful*, if it can find the target node v_d with high probability.

In the following let us examine the success probability of repeated applications of Algorithm 7 in which function ChooseFrom($C(v)$) is implemented as a probabilisitc choice function. Such solution constructions are employed, for example, within the ACO metaheuristic. The value of the input $\hat{f}$ is not important for the following analysis. Given any node v_i at level i of the search tree, let $\mathbf{p}(v_i)$ be the probability that a solution construction process includes node v_i. Note that there is a single path from v_0, the root node, to v_i. We denote the corresponding sequence of nodes by $(v_0, v_1, v_2, ..., v_i)$. Clearly, $\mathbf{p}(v_0) = 1$ and $\mathbf{p}(v_i) = \prod_{j=0}^{i-1} \mathbf{p}(v_{j+1}|v_j)$. Let $Success(\rho)$ denote the event of finding the target node v_d within ρ applications of Algorithm 7. Note that the probability of $Success(\rho)$ is equal to $1 - (1 - \mathbf{p}(v_d))^\rho$, and it is easy to check that the following inequalities hold:

$$1 - e^{-\rho\mathbf{p}(v_d)} \leq 1 - (1 - \mathbf{p}(v_d))^\rho \leq \rho\mathbf{p}(v_d) \tag{1}$$

By (1), it immediately follows that the chance of finding node v_d is *large* if and only if $\rho\mathbf{p}(v_d)$ is *large*, namely as soon as

$$\rho = O\left(1/\mathbf{p}(v_d)\right) \tag{2}$$

In the following, we will not assume anything about the exact form of the given probability distribution. However, let us assume that the transition probabilities are heuristically related to the *attractiveness* of child nodes. In other words, we assume that in a case in which a node v has two children, say w and q, and w is known (or believed) to be *more promising*, then $\mathbf{p}(w|v) > \mathbf{p}(q|v)$. This can be achieved, for example, by defining the transition probabilities proportional to the weights assigned by greedy functions.

Clearly, the probability distribution reflects the available knowledge on the problem, and it is composed of two types of knowledge. If the probability $\mathbf{p}(v_d)$ of reaching the target node v_d is "high", then we have a "good" problem knowledge. Let us call the knowledge that is responsible for the value of $\mathbf{p}(v_d)$ the **primal problem knowledge** (or just primal knowledge). From the dual point of view, we still have a "good" knowledge of the problem if for "most" of the wrong nodes (i.e. those that are not on the path from v_0 to v_d) the probability that they are reached is "low". We call this knowledge the **dual problem knowledge** (or just dual knowledge). Note that the quality of the dual knowledge grows with the value $\hat{f}$ that is provided as input to Algorithm 7. This means, the better the solution that we already know, the higher is the quality of the dual knowledge. Observe that the two types of problem knowledge outlined above are complementary, but not the same. Let us make an example to clarify these two concepts. Consider the search tree of Figure 1, where the target node is v_5. Let us analyze two different probability distributions:

Case (a) For each v and $w \in \mathcal{C}(v)$ let $\mathbf{p}(w|v) = 0.5$. Moreover, let us assume that no upper bound information is available. This means that each solution construction is performed until a leaf node is reached. When probabilistically constructing a solution the probability of each child is therefore the same at each construction step.

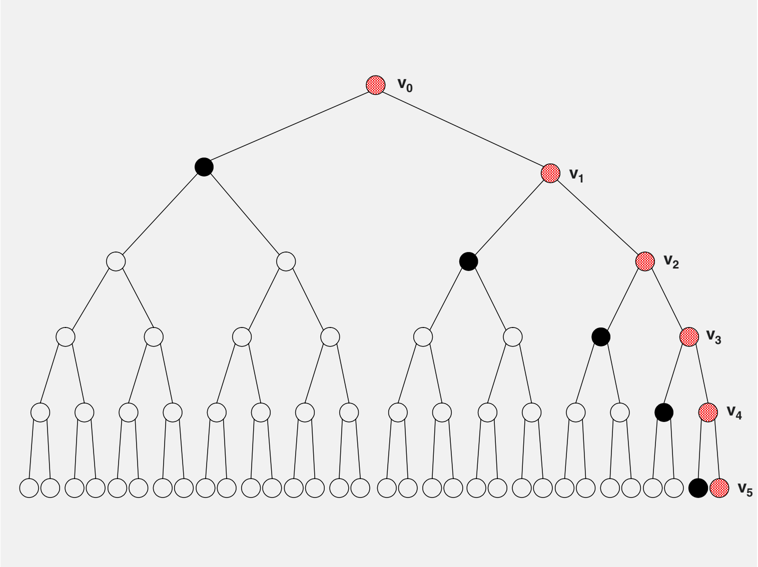


Fig. 1. Example of a search tree. v_5 is the unique optimal solution.

Case (b) In general, the transition probabilities are defined as in case (a), with one exception. Let us assume that the available upper bound indicates that the subspaces represented by the black nodes do not contain any better solutions than the ones we already know, that is, $\mathrm{UB}(v) \leq \hat{f}$, where v is a black node. Accordingly, the white children of the black nodes have probability 0 to be reached.

Note that in both cases the primal knowledge is "scarce", since the probability that the target node v_d is reached by a probabilistic solution construction decreases exponentially with d, that is, $\mathbf{p}(v_d) = 2^{-d}$. However, in **case (b)** the dual knowledge is "excellent", since for most of the wrong nodes (i.e. the white nodes), the probability that any of them is reached is zero. Viceversa, in **case (a)** the dual knowledge is "scarce", because there is a relatively "high" probability that a white node is reached.

By using the intuition given by the provided example, let us try to better quantify the quality of the available problem knowledge. Let V_i be the set of nodes at level i, and let

$$\ell(i) = \sum_{v \in V_i} \mathbf{p}(v), \qquad i = 1, \ldots, d \ . \tag{3}$$

Note that $\ell(i)$ is equal to the probability that the solution construction process reaches level i of the search tree. Observe that the use of the upper bound information makes the probabilities $\ell(i)$ smaller than one. **Case (b)** was obtained from **case (a)** by decreasing $\ell(i)$ (for $i = 1, \ldots, d$) down to 2^{i-1} (and without changing the probability $\mathbf{p}(v_i)$ of reaching the ancestor v_i of the target node at level i), whereas in **case (a)** it holds that $\ell(i) = 1$ (for $i = 1, \ldots, d$). In general, good dual knowledge is supposed to decrease $\ell(i)$ without decreasing the probability of reaching the ancestor v_i of the target node v_d. This discussion may suggest that a characterization of the available problem knowledge can be given by the following *knowledge ratio*:

$$K_{v_d} = \min_{1 \leq i \leq d} \frac{\mathbf{p}(v_i)}{\ell(i)} \tag{4}$$

The larger this ratio the better the knowledge we have on the target node v_d. In **case (a)** it is $K_{v_d} = 1/2^d$, whereas the knowledge ratio of **case (b)** is $K_{v_d} = 1/2$, which is exponentially larger.

Finally, it is important to observe that the way of (repeatedly) constructing solutions in a probabilistic way does not exploit the dual problem knowledge. For example in **case (b)**, although the available knowledge is "excellent", the target node v_d is found after an expected number of runs that is proportional to $1/\mathbf{p}(v_d) = 2^d$ (see Equation (2)), which is the same as in **case (a)**. In other words, the number of necessary probabilistic solution constructions only depends on the primal knowledge.

2.3 How to Exploit the Dual Knowledge?

The problem of Algorithm 7 is clearly the following one: When encountering a partial solution whose upper bound is less or equal to the value of the best solution found so far, the construction process is aborted, and the computation time invested in this construction is lost. Generally, this situation may occur very often. In fact, the probability for the abortion of a solution construction is $1 - \mathbf{p}(v_d)$ in the example outlined in the previous section, which is quite high.

In the following let us examine a first simple extension of Algorithm 7. The corresponding algorithm – henceforth denoted by PSC(α,$\hat{f}$) – is pseudocoded in Algorithm 8. Hereby, α denotes the maximum number of allowed extensions of partial solutions at each construction step; in other words, α is the maximum number of solutions to be constructed in parallel. We use the following additional notation: For any given set S of search tree nodes let $\mathcal{C}(S)$ be the set of children of the nodes in S. Morever, B_i denotes the set of reached nodes of tree level i. Recall that the root node v_0 is the only node at level 0.

The algorithm works as follows. Given the selected nodes B_i of level i (with $|B_i| \leq \alpha$)), the algorithm probabilistically chooses at most α solutions from $C := \mathcal{C}(B_i)$, the children of the nodes in B_i. The probabilistic choice of a

Algorithm 8 Parallel solution construction: PSC(α,$\hat{f}$)

```
input: $\alpha \in \mathbb{Z}^+$, the best known objective function value $\hat{f}$
initialization: $i := 0$, $B_i := \{v_0\}$, $z :=$ NULL
while $B_i \neq \emptyset$ do
  $B_{i+1} := \emptyset$
  $C := \mathcal{C}(B_i)$
  for $k = 1, \ldots, \min\{\alpha, |\mathcal{C}(B_i)|\}$ do
    $w :=$ ChooseFrom($C$)
    if $|\mathcal{C}(w)| > 0$ then
      if UB($w$) $> \hat{f}$ then $B_{i+1} := B_{i+1} \cup \{w\}$ end if
    else
      if $f(w) > \hat{f}$ then $z := w$, $\hat{f} := f(z)$ end if
    end if
    $C := C \setminus \{w\}$
  end for
  $i := i + 1$
end while
output: $z$ (which might be NULL)
```

child is performed in function ChooseFrom(C) proportionally to the following probabilities:

$$\mathbf{p}(w|C) = \frac{\mathbf{p}(w|\mathcal{F}(w))}{\sum\limits_{v \in C} \mathbf{p}(v|\mathcal{F}(v))}, \forall\, w \in C \tag{5}$$

Remember that $\mathcal{F}(w)$ denotes the father of node w. After choosing a node w it is first checked if w is a complete solution, or not. In case it is not a complete solution, it is checked if the available bounding information allows the further extension of this partial solution, in which case w is added to B_{i+1}. However, if w is already a complete solution, it is checked if its value is better than the value of the best solution found so far. The algorithm returns the best solution found, in case it is better than the $\hat{f}$ value that was provided as input. Otherwise the algorithm returns NULL.

Observe that when $\alpha = 1$, PSC(α,$\hat{f}$) is equivalent to SC($\hat{f}$). In contrast, when $\alpha > 1$ the algorithm constructs (maximally) α solutions non-independently in parallel. Concerning the example outlined in the previous section with the probability distribution as defined in **case(b)**, we can observe that algorithm PSC(α,$\hat{f}$) with $\alpha > 1$ solves this problem even within one application. At each step i, B_i will only contain the brother of the corresponding black node, because the upper bound information does not allow the inclusion of the black nodes in B_i. This shows that algorithm PSC(α,$\hat{f}$), in contrast to algorithm SC($\hat{f}$), benefically uses the dual problem knowledge.

2.4 Probabilistic Beam Search

For practical optimization, algorithm PSC(α,$\hat{f}$) has some drawbacks. First, in most cases algorithms for optimization are applied with the goal of finding a solution as good as possible, without having a clue beforehand about the value of good solutions. Second, the available upper bound function might not be very tight. For both reasons, solution constructions that lead to unsatisfying solutions are discarded only at very low levels of the search tree, that is, close to the leaves. Referring to the example of Section 2.2, this means that black nodes will only appear close to the leaves. In those cases, algorithm PSC(α,$\hat{f}$) will have practically no advantage over algorithm SC($\hat{f}$). It might even have a disadvantage due to the amount of computation time invested in choosing children from bigger sets.

The following simple extension can help in overcoming the drawbacks of algorithm PSC(α,$\hat{f}$). At each algorithm iteration we allow the choice of $\mu \cdot \alpha$ nodes from B_i, instead of α nodes. $\mu \geq 1$ is a parameter of the algorithm. Moreover, after the choice of the child nodes we restrict set B_{i+1} to the (maximally) α best solutions with respect to the upper bound information. This results in a so-called (probabilistic) beam search algorithm – henceforth denoted by PBS(α,μ,$\hat{f}$) – pseudo-coded in Algorithm 9. Note that algorithm PBS(α,μ,$\hat{f}$) is a generalization of algorithm PSC(α,$\hat{f}$), that is, when $\mu = 1$ both algorithms are equivalent. Algorithm PBS(α,μ,$\hat{f}$) is also a generalization of algorithm SC($\hat{f}$), which is obtained by $\alpha = \mu = 1$.

2.5 Adding a Learning Component: Beam-ACO

In general, algorithm PBS(α,μ,$\hat{f}$) can be expected to produce good solutions if (at least) two conditions are fullfilled: Neither the greedy function nor the upper bound function are misleading. If at least one of these two functions is misleading, the algorithm might not be able to find solutions above a certain threshold. One possibility of avoiding this drawback is to add a learning component to algorithm PBS(α,μ,$\hat{f}$), that is, adding a mechanism that is supposed to adapt the primal knowledge, the dual knowledge, or both, over time, based on accumulated search experience.

Ant colony optimization (ACO) [12] is the most prominent construction-based metaheuristics that attempts to learn the primal problem knowledge during run-time. ACO is inspired by the foraging behavior of ant colonies. At the core of this behavior is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, for example, combinatorial optimization problems. In general, the ACO approach attempts to solve an optimization problem by iterating the following two steps:

Algorithm 9 Probabilistic beam search: PBS(α,$\mu\hat{f}$)

```
input: $\alpha, \mu \in \mathbb{Z}^+$, the best known objective function value $\hat{f}$
initialization: $i := 0$, $B_i := \{v_0\}$, $z :=$ NULL
while $B_i \neq \emptyset$ do
   $B_{i+1} := \emptyset$
   $C := \mathcal{C}(B_i)$
   for $k = 1, \ldots, \min\{\mu \cdot \alpha, |\mathcal{C}(B_i)|\}$ do
      $w :=$ ChooseFrom($C$)
      if $|\mathcal{C}(w)| > 0$ then
         if UB($w$) $> \hat{f}$ then $B_{i+1} := B_{i+1} \cup \{w\}$ end if
      else
         if $f(w) > \hat{f}$ then $z := w$, $\hat{f} := f(z)$ end if
      end if
      $C := C \setminus \{w\}$
   end for
   Restrict $B_{i+1}$ to the (maximally) $\alpha$ best nodes w.r.t. their upper bound
   $i := i + 1$
end while
output: $z$ (which might be NULL)
```

- At each iteration, a certain number of α candidate solutions is probabilistically constructed. The respective probability distribution is derived from an available greedy function and from the values of so-called pheromone trail parameters, the *pheromone values*. The set of pheromone trail parameters is denoted by $\mathcal{T}$.
- The constructed candidate solutions are used to modify the pheromone values in a way that is deemed to bias future solution constructions towards areas of the search space containing high quality solutions. Hereby, the greedy function can be seen as the a priori available primal knowledge, whereas the pheromone values are used to modify (ideally, to improve) this a priori given primal knowledge over time.

While standard ACO algorithms use α applications of algorithm SC($\hat{f}$) at each iteration for the probabilistic construction of solutions, the idea of Beam-ACO [4, 5] is to use one application of probabilistic beam search PBS(α,μ,$\hat{f}$) instead.

A related ACO approach is labelled ANTS (see [27, 28, 29]). The characterizing feature of ANTS is the use of upper bound information for defining the primal knowledge. The latest version of ANTS [29] uses at each iteration algorithm PSC(α,$\hat{f}$) to construct candidate solutions.

2.6 Example: Longest Common Subsequence Problem

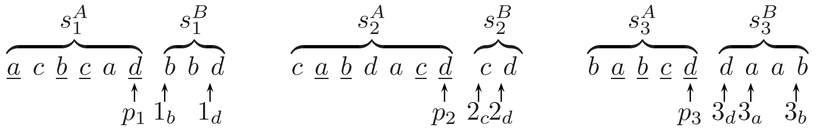
The longest common subsequence (LCS) problem is one of the classical string problems. Given a problem instance $(\mathcal{S}, \Sigma)$, where $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ is a set of n strings over a finite alphabet Σ, the problem consists in finding a longest string t^* that is a subsequence of all the strings in $\mathcal{S}$. Such a string t^* is called a *longest common subsequence* of the strings in $\mathcal{S}$. Note that a string t is called a subsequence of a string s, if t can be produced from s by deleting characters. For example, *dga* is a subsequence of *adagtta*. If $n = 2$ the problem is polynomially solvable, for example, by dynamic programming [17]. However, when $n > 2$ the problem is in general NP-hard [26]. Traditional applications of this problem are in data compression, syntactic pattern recognition, and file comparison [1], whereas more recent applications also include computational biology [38].

In order to apply algorithm PBS(α,μ,$\hat{f}$) to the LCS problem, we have to define the solution construction mechanism, the greedy function that defines the primal knowledge, and the upper bound function that defines the dual knowledge. We use the construction mechanism of the so-called Best-Next heuristic [15, 22] for our algorithm. Given a problem instance $(\mathcal{S}, \Sigma)$, this heuristic produces a common subsequence t sequentially by appending at each construction step a letter to t such that t maintains the property of being a common subsequence of all strings in $\mathcal{S}$. Given a common subsequence t of the strings in $\mathcal{S}$, we explain in the following how to derive the children of t. For that purpose we introduce the following notations:

1. Let $s_i = s_i^A \cdot s_i^B$ be the partition of s_i into substrings s_i^A and s_i^B such that t is a subsequence of s_i^A and s_i^B has maximal length. Given this partition, which is well-defined, we introduce position pointers $p_i := |s_i^A|$ for $i = 1, \ldots, n$ (see Figure 2 for an example).
2. The position of the first appearance of a letter $a \in \Sigma$ in a string $s_i \in \mathcal{S}$ after the position pointer p_i is well-defined and denoted by i_a. In case a letter $a \in \Sigma$ does not appear in s_i^B, i_a is set to ∞ (see Figure 2).
3. A letter $a \in \Sigma$ is called *dominated*, if exists at least one letter $b \in \Sigma$ such that $i_b < i_a$ for $i = 1, \ldots, n$;
4. $\Sigma_t^{\mathrm{nd}} \subseteq \Sigma$ henceforth denotes the set of non-dominated letters of the alphabet Σ with respect to a given t. Moreover, for all $a \in \Sigma_t^{\mathrm{nd}}$ it is required that $i_a < \infty$, $i = 1, \ldots, n$. Hence, we require that in each string s_i a letter $a \in \Sigma_t^{\mathrm{nd}}$ appears at least once after position pointer p_i.

The children $\mathcal{C}(t)$ of a node t are then determined as follows: $\mathcal{C}(t) := \{v = ta \mid a \in \Sigma_t^{\mathrm{nd}}\}$. The primal problem knowledge is derived from the greedy function $\eta(\cdot)$ that assigns to each child $v = ta \in \mathcal{C}(t)$ the following greedy weight:

$$\eta(v) = \min\{|s_i| - i_a \mid i = 1, \ldots, n\} \tag{6}$$



(a) String s_1 (b) String s_2 (c) String s_3

Fig. 2. Given is the problem instance ($\mathcal{S} = \{s_1, s_2, s_3\}, \Sigma = \{a, b, c, d\}$) where $s_1 = acbcadbbd$, $s_2 = cabdacdcd$, and $s_3 = babcddaab$. Let us assume that $t = abcd$. (a), (b), and (c) show the corresponding division of s_i into s_i^A and s_i^B, as well as the setting of the pointers p_i and the next positions of the 4 letters in s_i^B. Note that in case a letter does not appear in s_i^B (for example, letter a does not appear in s_1^B), the corresponding pointer is set to ∞. For example, as letter a does not appear in s_1^B, we set $1_a := \infty$.

The child with the highest greedy weight is considered the *most promising one*. Instead of the greedy weights themselves, we will use the corresponding ranks. More in detail, the child $v = ta$ with the highest greedy weight will be assigned rank 1, denoted by $r(v) = 1$, the child $w = tb$ with the second-highest greedy weight will be assigned rank 2 (that is, $r(w) = 2$), and so on.

In the following we explain the implementation of function $\mathsf{ChooseFrom}(C)$ of algorithm PBS(α,μ,$\hat{f}$). Remember that C denotes the set of children obtained from the nodes that are contained in the beam B_i (that is, $C := \mathcal{C}(B_i)$). For evaluating a child $v \in C$ we use the sum of the ranks of the greedy weights that correspond to the construction steps performed to construct string v. Let us assume that v is on the i-th level of the search tree, and let us denote the sequence of characters that forms string v by $v_1 \dots v_i$, that is, $v = v_1 \dots v_i$. Then,

$$\nu(v) := \sum_{j=1}^{i} r(v_1 \dots v_j), \tag{7}$$

where $v_1 \dots v_j$ denotes the substring of v from position 1 to position j. With this definition, Equation 5 can be defined for the LCS problem as follows:

$$\mathbf{p}(v|C) = \frac{\nu(v)^{-1}}{\sum_{w \in C} \nu(w)^{-1}}, \qquad \forall\, v \in C \tag{8}$$

Finally, we outline the upper bound function UB($\cdot$) that the PBS(α,μ,$\hat{f}$) algorithm requires. Remember that a given subsequence t splits each string $s_i \in \mathcal{S}$ into a first part s_i^A and into a second part s_i^B, that is, $s_i = s_i^A \cdot s_i^B$. Henceforth, $|s_i^B|_a$ denotes the number of occurrences of letter a in s_i^B for all $a \in \Sigma$. Then,

$$\mathrm{UB}(t) := |t| + \sum_{a \in \Sigma} \min\{|s_i^B|_a \mid i = 1, \dots, n\}, \tag{9}$$

that is, for each letter $a \in \Sigma$ we take the minimum of the occurrences of a in s_i^B, $i = 1, \dots, n$. Summing up these minima and adding the result to the length of t results in the upper bound. This completes the description of the implementation of the PBS(α,μ,$\hat{f}$) algorithm for the LCS problem.

In the following, we use algorithm PBS(α,μ,$\hat{f}$) in two different ways: First, we use PBS(α,μ,$\hat{f}$) in a multi-start fashion as shown in Algorithm 10, denoted by MS-PBS(α,μ). Second, we use PBS(α,μ,$\hat{f}$) within a Beam-ACO algorithm as explained in the following.

Algorithm 10 Multi-start probabilistic beam search: MS-PBS(α,μ)

```
input: $\alpha, \mu \in \mathbb{Z}^+$
$z :=$ NULL
$\hat{f} := 0$
while CPU time limit not reached do
    $v :=$ PBS($\alpha$,$\mu$,$\hat{f}$) {see Algorithm 9}
    if $v \neq$ NULL then $z := v$, $\hat{f} := |z|$
end while
output: $z$
```

The first step of defining a Beam-ACO approach – and, in general, any ACO algorithm – consists in the specification of the set of pheromone values $\mathcal{T}$. In the case of the LCS problem $\mathcal{T}$ contains for each position j of a string $s_i \in S$ a pheromone value $0 \leq \tau_{ij} \leq 1$, that is, $\mathcal{T} = \{\tau_{ij} \mid i = 1, \ldots, n, \ j = 1, \ldots, |s_i|\}$. A value $\tau_{ij} \in \mathcal{T}$ indicates the desirability of adding the letter at position j of string i to a solution: the greater τ_{ij}, the greater is the desirability of adding the corresponding letter. In addition to the definition of the pheromone values, we also introduce a solution representation that is suitable for ACO. Any common subsequence t of the strings in S can be translated into an ACO-solution $T = \{T_{ij} \in \{0, 1\} \mid i = 1, \ldots, n, \ j = 1, \ldots, |s_i|\}$ where $T_{ij} = 0$ if the letter at position j of string i was *not* added to t during the solution construction, and $T_{ij} = 1$ otherwise. Note that the translation of t into T is well-defined due to the construction mechanism. For example, given solution $t = abcdd$ for the problem instance of Figure 2, the corresponding ACO-solution is $T_1 = 101101001$, $T_2 = 011001101$, and $T_3 = 011111000$, where T_i refers to the sequence $T_{i1} \ldots T_{i|s_i|}$. In the following, for each given solution, the lower case notation refers to its string representation, and the upper case notation refers to its binary ACO representation.

The particular ACO framework that we used for our algorithm is the so-called $\mathcal{MAX} - \mathcal{MIN}$ ant system ($\mathcal{MM}$AS) algorithm implemented in the hyper-cube framework (HCF), an ACO variant that generally performs very well. One of the particularities of this algorithm is the use of an upper bound for the pheromone values with the aim of preventing convergence to a solution, see [7]. A high level description of the algorithm is given in Algorithm 11. The data structures used, in addition to counters and to the pheromone values, are: (1) the *best-so-far* solution T^{bs}, i.e., the best solution generated since the start of the algorithm; (2) the *restart-best* solution T^{rb}, that is, the best solution generated since the last restart of the algorithm; (3) the *convergence factor cf*, $0 \leq cf \leq 1$, which is a measure of how far the algorithm is from convergence; and (4) the Boolean variable *bs_update*, which becomes true when the algorithm reaches convergence.

Roughly, the algorithm works as follows. First, all the variables are initialized. In particular, the pheromone values are set to their initial value 0.5. Each algorithm iteration consists of the following steps. First, algorithm PBS(α,μ,$\hat{f}$) is applied with $\hat{f} = 0$ to generate a solution T^{pbs}. The setting of $\hat{f} = 0$ is chosen, because in ACO algorithms it is generally useful to learn also from solutions that are worse than the best solution found so far. The only change in algorithm PBS(α,μ,$\hat{f}$) occurs in the definition of the choice probabilities. Instead of using Equation 8, these probabilities are now defined as follows:

$$\mathbf{p}(v = ta|C) = \frac{\left(\min\limits_{i=1,\ldots,n}\{\tau_{ii_a}\} \cdot \nu(v)^{-1}\right)}{\sum\limits_{w=tb\in C}\left(\min\limits_{i=1,\ldots,n}\{\tau_{ii_b}\} \cdot \nu(w)^{-1}\right)}, \quad \forall\, v = ta \in C \qquad (10)$$

Remember in this context, that i_a was defined as the next position of letter a after position pointer p_i in string s_i. The intuition of choosing the minimum of the pheromone values corresponding to the next positions of a letter in the n given strings is as follows: If at least one of these pheromone values is low, the corresponding letter should not yet be appended to the string, because apparently there is another letter that should be appended first.

The second action at each iteration concerns the pheromone update conducted in the ApplyPheromoneUpdate(cf, bs_update, $\mathcal{T}$, T^{pbs}, T^{rb}, T^{bs}) procedure. Third, a new value for the convergence factor cf is computed. Depending on this value, as well as on the value of the Boolean variable *bs_update*, a decision on whether to restart the algorithm or not is made. If the algorithm is restarted, all the pheromone values are reset to their initial value (that is, 0.5). The algorithm is iterated until the CPU time limit is reached. Once terminated, the algorithm returns the string version t^{bs} of the best-so-far ACO-solution T^{bs}. In the following we describe the two remaining procedures of Algorithm 11 in more detail.

ApplyPheromoneUpdate(cf,bs_update,$\mathcal{T}$,T^{pbs},T^{rb},T^{bs}): In general, three solutions are used for updating the pheromone values. These are the solution T^{pbs} generated by the PBS algorithm, the restart-best solution T^{rb}, and the best-so-far solution T^{bs}. The influence of each solution on the pheromone update depends on the state of convergence of the algorithm as measured by the

Algorithm 11 Beam-ACO for the LCS problem

```
input: $\alpha, \mu \in \mathbb{Z}^+$
$T^{bs} :=$ NULL, $T^{rb} :=$ NULL, $cf := 0$, $bs\_update :=$ FALSE
$\tau_{ij} := 0.5$, $i = 1, \ldots, n$, $j = 1, \ldots, |s_i|$
while CPU time limit not reached do
   $T^{pbs} :=$ PBS($\alpha$,$\mu$,0) {see Algorithm 9}
   if $|t^{pbs}| > |t^{rb}|$ then $T^{rb} := T^{pbs}$
   if $|t^{pbs}| > |t^{bs}|$ then $T^{bs} := T^{pbs}$
   ApplyPheromoneUpdate($cf$,$bs\_update$,$\mathcal{T}$,$T^{pbs}$,$T^{rb}$,$T^{bs}$)
   $cf :=$ ComputeConvergenceFactor($\mathcal{T}$)
   if $cf > 0.99$ then
      if $bs\_update =$ TRUE then
         $\tau_{ij} := 0.5$, $i = 1, \ldots, n$, $j = 1, \ldots, |s_i|$
         $T^{rb} :=$ NULL
         $bs\_update :=$ FALSE
      else
         $bs\_update :=$ TRUE
      end if
   end if
end while
output: $t^{bs}$ (that is, the string version of ACO-solution $T^{bs}$)
```

convergence factor *cf*. Each pheromone value $\tau_{ij} \in \mathcal{T}$ is updated as follows:

$$\tau_{ij} := \tau_{ij} + \rho \cdot (\xi_{ij} - \tau_{ij}) \ , \tag{11}$$

where

$$\xi_{ij} := \kappa_{pbs} \cdot T_{ij}^{pbs} + \kappa_{rb} \cdot T_{ij}^{rb} + \kappa_{bs} \cdot T_{ij}^{bs} \ , \tag{12}$$

where κ_{pbs} is the weight (that is, the influence) of solution T^{pbs}, κ_{rb} is the weight of solution T^{rb}, κ_{bs} is the weight of solution T^{bs}, and $\kappa_{pbs} + \kappa_{rb} + \kappa_{bs} = 1$. After the application of the pheromone update rule (Equation 11), pheromone values that exceed $\tau_{\max} = 0.999$ are set back to $\tau_{\max}$ (similarly for $\tau_{\min} = 0.001$). This is done in order to avoid a complete convergence of the algorithm, which is a situation that should be avoided. Equation 12 allows to choose how to schedule the relative influence of the three solutions used for updating the pheromone values. For our application we used a standard update schedule as shown in Table 1.

ComputeConvergenceFactor($\mathcal{T}$): The convergence factor cf, which is a function of the current pheromone values, is computed as follows:

$$cf := 2 \left(\left(\frac{\sum_{\tau_{ij} \in \mathcal{T}} \max\{\tau_{\max} - \tau_{ij}, \tau_{ij} - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right)$$

Table 1. Setting of κ_{pbs}, κ_{rb}, κ_{bs}, and ρ depending on the convergence factor *cf* and the Boolean control variable *bs_update*

	bs_update = FALSE				*bs_update*
	$cf < 0.4$	$cf \in [0.4, 0.6)$	$cf \in [0.6, 0.8)$	$cf \geq 0.8$	= TRUE
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1
ρ	0.2	0.2	0.2	0.15	0.15

In this way, $cf = 0$ when the algorithm is initialized (or reset), that is, when all pheromone values are set to 0.5. On the other side, when the algorithm has converged, then $cf = 1$. In all other cases, *cf* has a value in $(0, 1)$. This completes the description of our Beam-ACO approach for the LCS problem.

Experimental Results

We implemented algorithms MS-PBS(α,μ) and Beam-ACO in ANSI C++ using GCC 3.2.2 for compiling the software. The experimental results that we outline in the following were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory. We applied algorithm MS-PBS(α,μ) with three different settings:

1. $\alpha = \mu = 1$: The resulting algorithm corresponds to a multi-start version of algorithm SC($\hat{f}$); see Algorithm 7. In the following we refer to this algorithm by **MS-SC**.
2. $\alpha = 10$, $\mu = 1$: This setting corresponds to a multi-start version of algorithm PSC(α,$\hat{f}$); see Algorithm 8. We refer henceforth to this algorithm by **MS-PSC**.
3. $\alpha = 10$, $\mu > 1$: These settings generate a multi-start version of algorithm PBS(α,μ,$\hat{f}$); see Algorithm 9. This algorithm version is referred to simply by **MS-PBS**. Note that the setting of μ will depend on the alphabet size, that is, the number of expected children of a partial solution.

In addition we applied Beam-ACO with $\alpha = 10$ and with the same settings for μ as chosen for MS-PBS.

For the experimentation we used a set of benchmark instances that was generated as explained in the following. Given $h \in \{100, 200, \ldots, 1000\}$ and Σ (where $|\Sigma| \in \{2, 4, 8, 24\}$), an instance is produced as follows. First, a string s of length h is produced randomly from the alphabet Σ. String s is in the following called *base string*. Each instance contains 10 strings. Each of these strings is produced from the base string s by traversing s and by deciding for each letter with a probabilitiy of 0.1 whether to remove it, or not. Note that the 10 strings of such an instance are not necessarily of the same length. As we produced 10 instances for each combination of h and $|\Sigma|$, 400 instances were generated in total. Note that the values of optimal solutions of these instances are unknown. However, a lower bound is obtained as follows. While producing the 10 strings of an instance, we record for each position of the base string s, whether the letter at that position was removed for the generation of at least one of the 10 strings. The number of positions in s that were never removed constitutes the lower bound value henceforth denoted by LB_I with respect to an instance I.

We applied each of the 4 algorithms exactly once for $h/10$ seconds to each problem instance. We present the results averaged over the 10 instances for each combination of h (the length of the base string that was used to produce an instance) and the alphabet size $|\Sigma|$. Two measures are presented:

1. The (average) length of the solutions expressed in deviation (percentage) from the respective lower bounds, which is computed as follows:

$$\left(\frac{f}{\mathrm{LB}_I} - 1\right) \cdot 100, \tag{13}$$

   where f is the length of the solution achieved by the respective algorithm.
2. The computation time of the algorithms, which refers to the time the best solution was found within the given CPU time (averaged over the 10 instances of each type).

The results are shown graphically in Figure 3. The graphics on the left hand side show the algorithm performance (in percentage of deviation from the lower bound), and the graphics on the right hand side show the computation times. The following observations are of interest. First, while having a comparable computation time, algorithm MS-PBS is always clearly better than algorithms MS-PSC and MS-SC. Second, algorithm Beam-ACO is consistently the best algorithm of the comparison. This shows that it can pay off adding a learning component to algorithm (MS-)PBS. The advantage of Beam-ACO over MS-PBS grows with growing alphabet size, that is, with growing problem complexity. This advantage of Beam-ACO comes with a slight increase in computational cost. However, this is natural: due to the learning component, Beam-ACO has a higher probability than MS-PBS of improving on the best solution found even at late stages of a run. Finally, a last interesting remark concerns the comparison of MS-PSC with MS-SC. Despite of the construction of solutions in parallel, MS-PSC is always slightly beaten by MS-SC. This is due to fact that the used upper bound function is not tight at all, which results in the fact that constructing solutions in parallel in the way of algorithm (MS-)PSC is rather a waste of computation time.

Algorithm 12 Pseudocode of a memetic algorithm

```
for $i := 1$ to popsize do
    pop[i] := Heuristic Solution(ProblemData)
    pop[i] := Local Search(pop[i])
    Evaluate(pop[i])
end for
while allowed runtime not exceeded do
    for $i := 1$ to offsize do
        if recombination is performed then
            $parent_1$ := Select(pop)
            $parent_2$ := Select(pop)
            offspring[i] := Recombine($parent_1$,$parent_2$)
        else
            offspring[i] := Select(pop)
        end if
        if mutation is performed then
            offspring[i] := Mutate(offspring[i])
        end if
        offspring[i] := Local Search(offspring[i])
        Evaluate(offspring[i])
    end for
    pop := Replace(pop, offspring)
end while
```

3 Using Metaheuristics Concepts Within Branch & Bound

In this section, we present a collaborative technique that integrates a population based metaheuristics, a memetic algorithm (MA) [31, 19, 23], with the beam search variant of the branch & bound procedure. MAs are based on the systematic exploitation of knowledge about the problem being solved, and the synergistic combination of ideas taken from both population-based techniques and trajectory-based metaheuristics. A very common way to achieve this combination is using the template of an evolutionary algorithm, endowing it with local search add-ons. A general sketch of this kind of MA is shown in Algorithm 12. Several things must be noted: firstly, initialization is very often done by means of problem-dependent constructive heuristics, thus ensuring that a good starting point is used for the evolutionary search. Local search can be also used in this initialization stage (to supplement the lack of an adequate constructive heuristic, or to complement the latter). The remaining components of the algorithm are typically chosen so that they incorporate problem-knowledge (if possible) as well.

According to the previous description, it is clear that MAs are specifically concerned with exploiting as much problem-knowledge as available. This makes MAs specifically suited for taking part in hybrid approaches, either integrative or collaborative [34]. In this case, we have considered an approach in which the control flows of BS and MA are intertwined: phases of BS and MA alternate, and both processes share the best known solution. The technique provides the following benefits:

- The best known solution can be used by the beam search part to purge its problem queue, by not expanding partial nodes whose upper bound is worse than the one obtained by the MA.
- The beam search can guide the search of the MA by injecting information about more promising regions of the search space into the MA population.

The resulting algorithm for a minimization problem is pseudo-coded in Algorithm 13. The procedure performs a standard beam search procedure (B_i is used to maintain the *beam* at level i of the search tree and α is the *beam width*, i.e., the maximum number of partial solutions to be expanded at each level). After spreading out each level, if a level dependent problem specific condition is fulfilled (represented in the pseudocode by the *runMA* variable), the MA is run with a population that is initialized using the best nodes (w.r.t some criteria) in the current beam. Note that nodes in the beam are partial solutions, whereas the MA population consists of complete solutions, so a problem specific procedure must be used to complete them. After the MA stabilizes, if the solution it provides improves the incumbent one, this one is updated.

Example: Shortest Common Supersequence Problem

The Shortest Common Supersequence Problem (SCSP) is a well-known problem in the area of string analysis. Essentially, given a certain alphabet Σ and a set L of strings from Σ, the aim is to find a minimal-length sequence s, such that all strings in the given set L can be *embedded* in s. The SCSP can be shown to be NP$-$hard, even if strong constraints are posed on L, or on Σ. For example, it is NP$-$hard in general when all s_i have length two [39], or when the alphabet size $|\Sigma|$ is two [30]. This combinatorial problem is interesting as it constitutes a formalization of different real-world problems. For example, it has many implications in bioinformatics [18]: it is a problem with a close relationship to multiple sequence alignment [37], and to probe synthesis during microarray production [35]. Besides this, it also has applications in planning [14] and data compression [39], among other fields.

Formally, the notion of embedding can be described as follows. Let s and r be two strings of symbols taken from Σ. String s can be said to embed string r (denoted as $s \succ r$) using the following recursive definition:

$$\begin{aligned} s \succ \epsilon &= \text{True} \\ \epsilon \succ r &= \text{False}, \quad \text{if } r \neq \epsilon \\ \alpha s \succ \alpha r &= s \succ r \\ \alpha s \succ \beta r &= s \succ \beta r, \ \text{if } \alpha \neq \beta \end{aligned} \tag{14}$$

Algorithm 13 Beam Search and MA Hybrid: Hybrid(α,$\hat{f}$)

```
input: $\alpha \in \mathbb{Z}^+$, the best known objective function value $\hat{f}$
initialization: $i := 0$, $B_i := \{\epsilon\}$, $z :=$ NULL
while $B_i \neq \emptyset$ do
  $B_{i+1} := \emptyset$
  for $w \in \mathcal{C}(B_i)$ do
    if $|\mathcal{C}(w)| > 0$ then
      if $\mathrm{LB}(w) < \hat{f}$ then $B_{i+1} := B_{i+1} \cup \{w\}$ end if
    else
      if $f(w) < \hat{f}$ then $z := w$, $\hat{f} := f(z)$ end if
    end if
  end for
  Restrict $B_{i+1}$ to the (maximally) $\alpha$ best nodes
  if $runMA$ then
    $pop$ := select $popsize$ best nodes from $B_{i+1}$
    for $j = 1, \ldots, popsize$ do
      complete partial solution $pop_j$
    end for
    $sol$ := run $\mathrm{MA}_{pop}$
    if $f(sol) < \hat{f}$ then $z := sol$, $\hat{f} := f(z)$ end if
  end if
  $i := i + 1$
end while
output: $z$ (which might be NULL)
```

Plainly, $s \succ r$ means that all symbols in r are present in s in the very same order (although not necessarily consecutive).

Formally, an instance $I = (\Sigma, L)$ for the SCSP is given by a finite alphabet Σ and a set L of m strings $\{s_1, \cdots, s_m\}$, $s_i \in \Sigma^*$. The problem consists of finding a string s of minimal length that embeds each string in L ($s \succ s_i, \forall s_i \in L$ and $|s|$ is minimal).

A branch & bound algorithm to solve an instance $I = (\Sigma, L)$ of the SCSP can start from a single node containing as tentative solution ϵ. In order to implement function $\mathcal{C}(w)$, $|\Sigma|$ subproblems are generated, each of them obtained by appending a symbol from Σ to the partial solution w. Nodes with unproductive characters (i.e., not contributing to embedding any string in L) are pruned from the search tree. To obtain a lower bound for a node s^t, the set of remaining strings in L not embedded by s^t must first be calculated as follows:

$$R = \{r_i \mid (s_i^e, r_i) = s^t \gg s_i, s_i \in L\} \tag{15}$$

where $s \gg r = (r^e, r^r)$ if r^e is the longest initial segment of string r embedded by s and r^r is the remaining part of r not embedded by s. Let $M(\alpha, R)$ be the maximum number of occurrences of symbol α in any string in R:[4]

$$M(\alpha, R) = \max\{|r_i|_\alpha \mid r_i \in R\} \tag{16}$$

Clearly, every common supersequence for the remaining strings must contain at least $M(\alpha, R)$ copies of the symbol α. Thus a lower bound can be obtained by summing the length of the tentative solution and the maximum number of occurrences in any string in R of each symbol of the alphabet:

$$LB(s^t) = |s^t| + \sum_{\alpha \in \Sigma} M(\alpha, R) \tag{17}$$

In order to rank nodes in the branch & bound queue, the following quality function was used for each node:

$$\textit{quality}\ (s^t, L) \;=\; \sum_{s_i \in L} \{\, |s_i^e| \mid (s_i^e, r_i) = s^t \gg s_i\} \tag{18}$$

so that tentative solutions embedding more symbols in L are selected. As all tentative solutions in the same level of the search tree have the same length, the algorithm selects nodes that provide good initial segments for constructing a short supersequence. Before being injected into the MA population, solutions were randomly completed and repaired using the following function:

$$\begin{array}{lcll} \rho\,(s, L) & = & s, & \text{if } \forall i : s_i = \epsilon \\ \rho\,(\alpha s', L) & = & \rho(s', L), & \text{if } \nexists i : s_i = \alpha s'_i \\ \rho\,(\alpha s', L) & = & \alpha\rho(s', L|_\alpha), & \text{if } \exists i : s_i = \alpha s'_i \\ \rho\,(\epsilon, L) & = & \mathrm{MM}(L), & \text{if } \exists i : s_i \neq \epsilon \end{array} \tag{19}$$

where MM is the *Majority Merge* algorithm (see Algorithm 14) described in [8]. This is a greedy algorithm that constructs a supersequence incrementally by adding the symbol most frequently found at the front of the strings in L, and removing these symbols from the corresponding strings.

Note that, apart from completing a string in order to have a valid supersequence, this function also removes unproductive steps from the repaired string, acting thus as a local searcher.

Preliminary tests show that partial good solutions were only obtained after descending a substantial number of levels in the beam search tree. This led us to the following strategy for interleaving the MA and the branch & bound in the hybrid algorithm: start by running in isolation the branch & bound part of the algorithm for a initial number of levels, and then periodically interleave both algorithms afterwards. To be precise, an estimation for the SCSP solution s_0 was calculated using the *Weighted Majority Merge* (WMM) algorithm [8] and its length was used to set $l_0 = 0.7 \cdot |s_0|$. The condition for running the MA was $(i > l_0)$ **and** $(i$ **mod** $l = 0)$, where variable i (see Algorithm 13) is the current level explored by the beam search part of the

[4] As in Section 2.6, $|r_i|_\alpha$ denotes the number of occurrences of symbol α in r_i.

Algorithm 14 Majority Merge algorithm

```
input: $L = \{s_1, \cdots, s_m\}$
$s := \epsilon$
repeat
    for $\alpha \in \Sigma$ do
        $\nu(\alpha) := \sum_{s_i \in L, s_i = \alpha s'_i} 1$
    end for
    $\beta \leftarrow \max^{-1}\{\nu(\alpha) \mid \alpha \in \Sigma\}$
    for $s_i \in L, s_i = \beta s'_i$ do
        $s_i := s'_i$
    end for
    $s := s\beta$
until $\sum_{s_i \in L} |s_i| = 0$
output: $s$
```

algorithm, and parameter l controls the balance between the MA and beam search, i.e., an execution of the MA is performed every l iterations of the beam search. A sensitivity analysis of the parameters was done in a similar way to that described in [16] and, based on it, the following values were used for the different parameters of the algorithm: $\alpha = 10000$ and $l = 10$.

As to the MA used, it evolves sequences in $|\Sigma|^\lambda$, where $\lambda = \sum_{s_i \in L} |s_i|$. Before being evaluated, sequences in the population are repaired using the ρ function. After this repairing, raw fitness (to be minimized) is simply computed as:

$$\begin{array}{lcll} \mathit{fit}\,(s, L) & = & 0, & \text{if } \forall i : s_i = \epsilon \\ \mathit{fit}\,(\alpha s', L) & = & 1 + \mathit{fit}(s', L|_\alpha), & \text{if } \exists i : s_i \neq \epsilon \end{array} \tag{20}$$

Selection in the MA was performed by binary tournament selection, the mutation operator randomly flips a character in the sequence, and recombination is carried out using an standard uniform crossover operator. The local search technique used is based on the neighborhood defined by the $\textsc{Del}_k : \Sigma^* \times (\Sigma^*)^m \rightarrow \Sigma^*$ operation [35]. The functioning of this procedure is as follows:

$$\begin{array}{lcll} \textsc{Del}_k\,(\alpha s, L) & = & \rho(s, L), & \text{if } k = 1 \\ \textsc{Del}_k\,(\alpha s, L) & = & \alpha \textsc{Del}_{k-1}(s, L|_\alpha), & \text{if } k > 1 \end{array} \tag{21}$$

This operation thus removes the k-th symbol from a string, and then submits it to the repair function so that all strings in L can be embedded. Notice that the repairing function can actually find that the sequence is feasible, hence resulting in a reduction of length by one symbol. A full local-search scheme is defined by iterating this operation until no single deletion results in length reduction (see Algorithm 15). The improvement in solution quality attainable via the application of this LS operator comes obviously at the expenses of an increased computational cost. This additional cost might be too high if

Algorithm 15 Local search for MA(s, L)

```
input: $s \in \Sigma^*$, $L = \{s_1, \cdots, s_m\}$
initialization: $k := 1$
while $k < |s|$ do
    $r := \textsc{Del}_k(s, L)$
    if $fit(r, L) < fit(s, L)$ then
        $s := r$
        $k := 1$
    else
        $k := k + 1$
    end if
end while
output: $s$
```

LS were massively applied. On the other hand, the extreme option of simply removing LS handicaps the search capabilities of the algorithm. A pragmatic solution can be found in the use of partial lamarckism [21], namely using LS but with some intermediate probability. Preliminary experiments were conducted with probabilities to apply local search in $\{0, 0.01, 0.1, 0.5, 1\}$ (see [11]), and setting this parameter to 0.01 provided a better tradeoff between the attainable improvement, and the additional computational cost implied.

Experimental Results

In this section, we do a experimental comparison of the beam search and MA hybrid algorithm with respect to the probabilistic beam search (PBS) algorithm for the SCSP described in [6]. For this purpose, two sets of benchmark instances have been used:

The first one – henceforth referred to as RandomSet – consists of random strings with different alphabet lengths. To be precise, each instance is composed of eight strings, four of them of length 40, and the other four of length 80. Each of these strings is randomly generated, using an alphabet Σ. The benchmark set consists of 5 classes of each 5 instances characterized by different alphabet sizes, namely $|\Sigma| = 2$, 4, 8, 16, and 24. Thus, the benchmark set consists of 25 different problem instances.

A second set of instances – henceforth referred to as RealSet – is composed of strings obtained from molecular sequences, comprising both DNA sequences ($|\Sigma| = 4$) and protein sequences ($|\Sigma| = 20$). In the first case, we have taken two DNA sequences of the SARS coronavirus from a genomic database[5]; these sequences are 158 and 1269 nucleotides long. As to the protein sequences, we have considered four of them, extracted from Swiss-Prot[6]:

[5] `http://gel.ym.edu.tw/sars/genomes.html`

[6] `http://www.expasy.org/sprot/`

- *Oxytocin*: quite important in pregnant women, this protein causes contraction of the smooth muscle of the uterus and of the mammary gland. The sequence is 125-aminoacid long.
- *p53*: this protein is involved in the cell cycle, and acts as tumor suppressor in many tumor types; the sequence is 393-aminoacid long.
- *Estrogen*: involved in the regulation of eukaryotic gene expression, this protein affects cellular proliferation and differentiation; the sequence is 595-aminoacid long.
- *Myelin*: this sequence correspond to a transcription factor of myelin, and is associated with neuronal differentiation. The sequence is 1186-aminoacid long.

Problem instances in RealSet are obtained from the target sequence by removing symbols from the latter with a certain probability p % (p ∈{10 %,15 %,20 %} in our experiments).

Figure 4 shows results for RandomSet. Results are averaged over 5 independent runs for each problem instance and further averaged over 5 different problem instances with the same alphabet length. For the beam search and MA hybrid, executions were performed on a Pentium IV PC (2400MHz and 512MB of main memory), and a time limit of 600 seconds per execution was imposed. As to the PBS, tests were performed on a AMD64X2 4400 processor and 4 Gb of memory. The time limit was set to 350 seconds, that roughly corresponds to the time given to other algorithm on a different machine. Results show that PBS performs better for this instance set, as it finds better solutions except for $|\Sigma| = 2$. Note that PBS performs several iterations of the beam search part of the algorithm, and thus exhausts the allowed time, whereas the beam search and MA hybrid only performs a beam search execution, and does not necessarily use all the permitted time. We also studied the performance of a variation of the beam search and MA hybrid (labelled MA-BS 2 in Figure 4) that exhausts the allowed time by performing several iterations of the beam search. In order to introduce more randomness in the algorithm, each time the MA was executed, its population was initialized by selecting nodes from the beam using binary tournament selection. Results show that MA-BS 2 outperforms PBS for $|\Sigma| \in \{2, 24\}$, and is slightly worse for $|\Sigma| = 4$.

Figure 5 shows results for RealSet. In this case, the beam search and MA hybrid performs better, as it always finds the presumed optimal solution in all runs (except for the MYELIN instance with p %=20 %). Note that in this latter instance, PBS finds a non-optimal better result. We also make note that the second version of the MA-BS hybrid does not improve these results because the allowed time is exhausted in the first iteration of the beam search; for this reason the results obtained by MA-BS 2 are not shown in Figure 5.

4 Conclusions

In this chapter we have dealt with hybridizations of branch & bound derivatives (i.e., beam search) and metaheuristic techniques and have shown that the resulting hybrid algorithms provide better results than their counterparts working alone. Particularly, we have highlighted two different proposals: in the first one, a construction-based metaheuristic is enriched with branch & bound features. Here, we start from a (parallel) solution construction method with a probabilistic component in the election of the next step to execute (i.e., the set of nodes that can be reached from the current state). Then we improve it by incorporating first a beam search component in the election, resulting in a probabilistic beam search algorithm, and second adding a learning component to adjust the knowledge acquired from the accumulated experience.

Our second proposal consists of a branch & bound technique that collaborates in an interleaved way with a metaheuristic, namely a memetic algorithm. Here, the branch & bound technique is used to identify the promising regions of the search space in which the optimal solution can be found. The metaheuristics is then used to exploit this knowledge in order to improve the bounds employed by the branch & bound technique to force further branch pruning.

Our hybrid algorithms have been first described in detail and then applied on practical problems to show their effectiveness. This paper clearly shows that both exact techniques such as branch & bound (including non-complete derivatives such as beam search) and metaheuristics can clearly benefit one from each other.

Acknowledgements

This work was partially supported by the Spanish Ministry of Science and Technology (MCyT) under contracts TIC2002-04498-C05-02, TIN2004-7943-C04-01, and TIN-2005-08818-C04-01; and by the *Ramón y Cajal* program of which Christian Blum is a research fellow. Monaldo Mastrolilli acknowledges support from the Swiss National Science Foundation projects 200021-104017/1 (Power Aware Computing) and 200021-100539/1 (Approximation Algorithms for Machine scheduling Through Theory and Experiments).

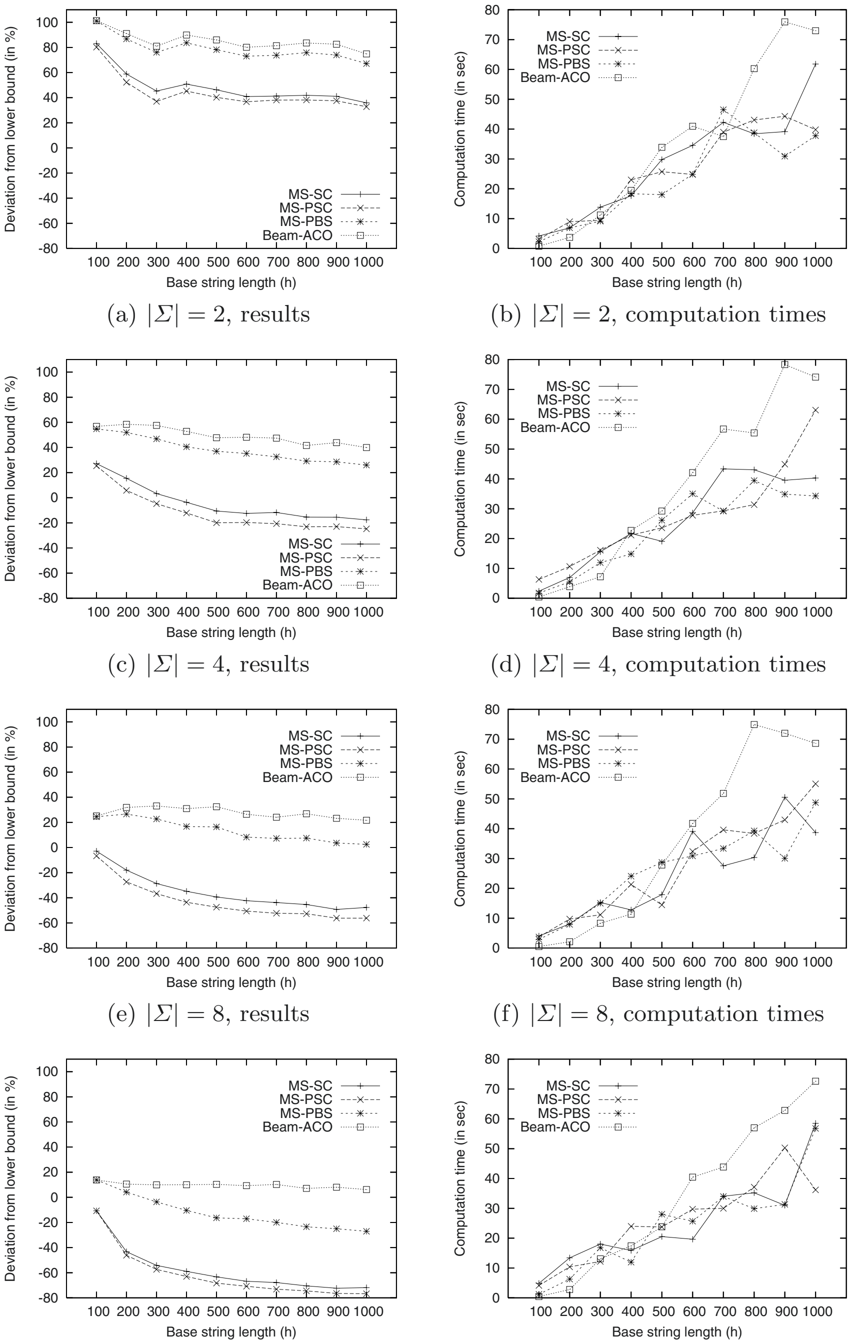
References

1. A. Aho, J. Hopcroft, and J. Ullman. *Data structures and algorithms.* Addison-Wesley, Reading, MA, 1983.
2. S. Al-Shihabi. Backtracking ant system for the traveling salesman problem. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Proceedings of ANTS 2004 – 4th International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 318–325. Springer-Verlag, Berlin, 2004.

3. M. J. Blesa and C. Blum. Ant colony optimization for the maximum edge-disjoint paths problem. In G. R. Raidl et al., editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2004*, volume 3005 of *Lecture Notes in Computer Science*, pages 160–169. Springer-Verlag, Berlin, 2004.
4. C. Blum. Beam-ACO–hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers and Operations Research*, 32:1565–1591, 2005.
5. C. Blum, J. Bautista, and J. Pereira. Beam-ACO applied to assembly line balancing. In M. Dorigo, L. M. Gambardella, A. Martinoli, R. Poli, and T. Stützle, editors, *Proceedings of ANTS 2006 – Fifth International Workshop on Swarm Intelligence and Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 14–27. Springer-Verlag, Berlin, Germany, 2006.
6. C. Blum, C. Cotta, A. J. Fernández, and J. E. Gallardo. A probabilistic beam search algorithm for the shortest common supersequence problem. In C. Cotta et al., editor, *Proceedings of EvoCOP 2007 – Seventh European Conference on Evolutionary Computation in Combinatorial Optimisation*, volume 4446 of *Lecture Notes in Computer Science*, pages 36–47. Springer-Verlag, Berlin, Germany, 2007.
7. C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 34(2):1161–1172, 2004.
8. J. Branke, M. Middendorf, and F. Schneider. Improved heuristics and a genetic algorithm for finding short supersequences. *OR-Spektrum*, 20:39–45, 1998.
9. S. Casey and J. Thompson. GRASPing the examination scheduling problem. In E. K. Burke and P. De Causmaecker, editors, *Proceedings of PATAT 2002 – 4th International Conference on Practice and Theory of Automated Timetabling*, volume 2740 of *Lecture Notes in Computer Science*, pages 232–246. Springer-Verlag, Berlin, 2003.
10. C. Cotta. Protein structure prediction using evolutionary algorithms hybridized with backtracking. In J. Mira and J. R. Álvarez, editors, *Proceedings of the 7th International Work-Conference on Artificial and Natural Neural Networks (IWANN 2003)*, volume 2687 of *Lecture Notes in Computer Science*, pages 321–328. Springer-Verlag, Berlin, 2003.
11. C. Cotta. Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In J. Mira and J. R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, number 3562 in Lecture Notes in Computer Science, pages 84–91, Berlin Heidelberg, 2005. Springer-Verlag.
12. M. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.
13. T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
14. D. E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, 1992.
15. C. B. Fraser. *Subsequences and supersequences of strings*. PhD thesis, University of Glasgow, 1995.
16. J. E. Gallardo, C. Cotta, and A. J. Fernández. On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):77–83, 2007.
17. D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Computer Science and Computational Biology. Cambridge University Press, Cambridge, 1997.

18. M. T. Hallet. *An integrated complexity analysis of problems from computational biology.* PhD thesis, University of Victoria, 1996.
19. W. E. Hart, N. Krasnogor, and J. E. Smith. *Recent Advances in Memetic Algorithms.* Springer-Verlag, Berlin Heidelberg, 2005.
20. J. N. Hooker. Unifying local and exhaustive search. In L. Villaseñor and A. I. Martinez, editors, *Proceedings of ENC 2005 – Sixth Mexican International Conference on Computer Science*, pages 237–243. IEEE press, 2005.
21. C. Houck, J. A. Joines, M. G. Kay, and J. R. Wilson. Empirical investigation of the benefits of partial lamarckianism. *Evolutionary Computation*, 5(1):31–60, 1997.
22. K. Huang, C. Yang, and K. Tseng. Fast algorithms for finding the common subsequences of multiple sequences. In *Proceedings of the International Computer Symposium*, pages 1006–1011. IEEE press, 2004.
23. N. Krasnogor and J. E. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.
24. G. B. Lamont, S. M. Brown, and G. H. Gates Jr. Evolutionary algorithms combined with deterministic search. In V. W. Porto, N. Saravanan, D. E. Waagen, and A. E. Eiben, editors, *Proceedings of Evolutionary Programming VII, 7th International Conference*, volume 1447 of *Lecture Notes in Computer Science*, pages 517–526. Springer-Verlag, Berlin, 1998.
25. E. Lawler and D. Wood. Branch and bound methods: A survey. *Operations Research*, 4(4):669–719, 1966.
26. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.
27. V. Maniezzo. Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
28. V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16:927–935, 2000.
29. V. Maniezzo and M. Milandri. An ant-based framework for very strongly constrained problems. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of ANTS 2002: 3rd International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 222–227. Springer-Verlag, Berlin, 2002.
30. M. Middendorf. More on the complexity of common superstring and supersequence problems. *Theoretical Computer Science*, 125:205–228, 1994.
31. P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Press, Boston, Massachusetts, USA, 2003.
32. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.
33. P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.
34. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J. R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, number 3562 in Lecture Notes in Computer Science, pages 41–53, Berlin Heidelberg, 2005. Springer-Verlag.

35. S. Rahmann. The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, 19(Suppl. 2):ii156–ii161, 2003.
36. S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.
37. J. S. Sim and K. Park. The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms*, 1(1):111–117, 2003.
38. T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.
39. V. G. Timkovsky. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25:565–580, 1990.



(a) $|\Sigma| = 2$, results

(b) $|\Sigma| = 2$, computation times

(c) $|\Sigma| = 4$, results

(d) $|\Sigma| = 4$, computation times

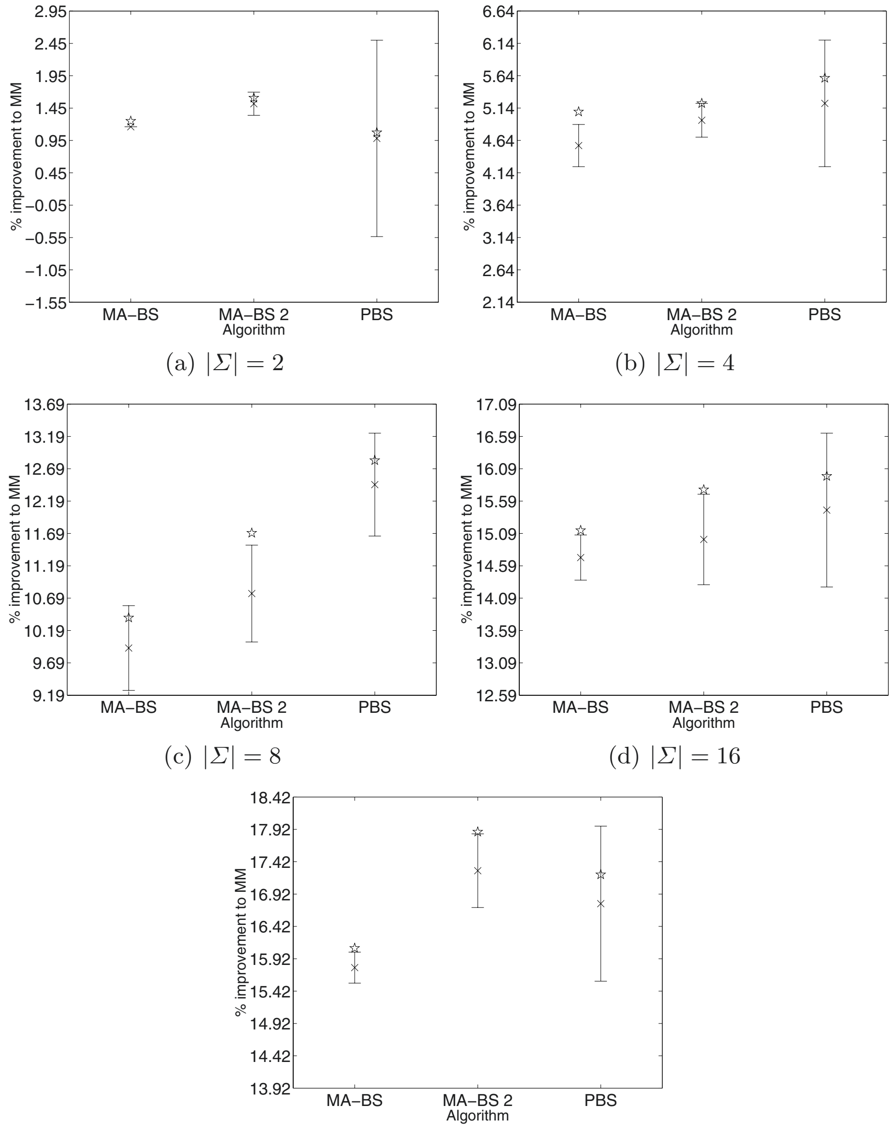
(e) $|\Sigma| = 8$, results

(f) $|\Sigma| = 8$, computation times

(g) $|\Sigma| = 24$, results

(h) $|\Sigma| = 24$, computation times

Fig. 3. Results and computation times of algorithms MS-SC, MS-PSC, MS-PBS, and Beam-ACO



(a) $|\Sigma| = 2$

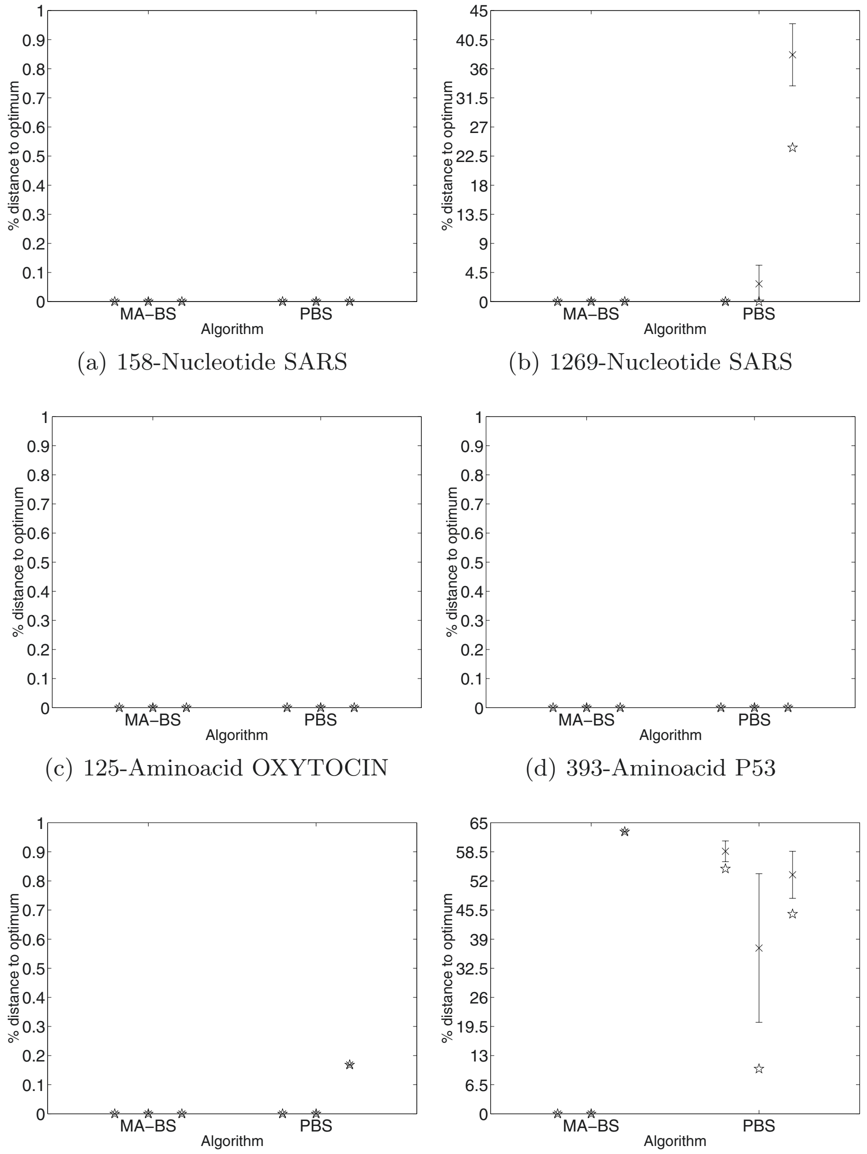
(b) $|\Sigma| = 4$

(c) $|\Sigma| = 8$

(d) $|\Sigma| = 16$

(e) $|\Sigma| = 24$

Fig. 4. Comparison of two versions of MA-BS Hybrid Algorithm and PBS on random instances for different alphabet sizes. Figures show relative improvements with respect to solutions provided by MM. A × sign indicates the mean solution, whereas a $\star$ marks the best solution. Standard deviations of distribution are also depicted.



(a) 158-Nucleotide SARS

(b) 1269-Nucleotide SARS

(c) 125-Aminoacid OXYTOCIN

(d) 393-Aminoacid P53

(e) 595-Aminoacid ESTROGEN

(f) 1186-Aminoacid MYELIN

Fig. 5. Comparison of MA-BS Hybrid Algorithm and PBS on different real instances and gap ∈ {10 %, 15 %, 20 %} (from left to right for each algorithm). Figures show relative distances to optimal solutions. A × sign indicates the mean solution, whereas a $\star$ marks the best solution. Standard deviations of distribution are also depicted.

Very Large-Scale Neighborhood Search: Overview and Case Studies on Coloring Problems

Marco Chiarandini[1], Irina Dumitrescu[2], and Thomas Stützle[3]

[1] Department of Mathematics and Computer Science
University of Southern Denmark, Odense, Denmark
marco@imada.sdu.dk

[2] School of Mathematics and Statistics
University of New South Wales, Sydney, Australia
irina.dumitrescu@unsw.edu.au

[3] Computer & Decision Engineering (CoDE) Department, IRIDIA
Université Libre de Bruxelles, Brussels, Belgium
stuetzle@ulb.ac.be

Summary. Two key issues in local search algorithms are the definition of a neighborhood and the way to examine it. In this chapter we consider techniques for examining very large neighborhoods, in particular, ways for exactly searching them. We first illustrate such techniques using three paradigmatic examples. In the largest part of the chapter, we focus on the development and experimental study of very large-scale neighborhood search algorithms for two coloring problems. The first example concerns the well-known (vertex) graph coloring problem. Despite initial promising results on the use of very large-scale neighborhoods, our final conclusion was negative: the usage of the proposed very large-scale neighborhoods did not help to improve the performance of effective stochastic local search algorithms. The second example, the graph set T-coloring problem, yielded more positive results. In this case, a very large-scale neighborhood that was specially tailored for this problem and that can be efficiently searched, resulted to be an essential component of a new state-of-the-art algorithm for various instance classes.

1 Introduction

Many efficient algorithms for combinatorial optimization problems rely on (perturbative) local search techniques [1, 37]. These start from some initial candidate solution for the problem to be solved and then iteratively replace the current candidate solution by some neighboring one. The neighborhood of a solution comprises all those candidate solutions that are reachable in one logical step of the algorithm. More formally, a neighborhood N can be defined as a function $N : S \rightarrow 2^S$ that assigns to every candidate solution s

M. Chiarandini et al.: *Very Large-Scale Neighborhood Search: Overview and Case Studies on Coloring Problems*, Studies in Computational Intelligence (SCI) **114**, 117–150 (2008)
www.springerlink.com

in the search space S a set of neighbors $N(s) \subseteq S$. The simplest local search algorithm only accepts better neighbors and it is therefore called iterative improvement. It terminates once no improving neighbor is available anymore. In this case, a local optimum has been reached.

Often, the neighborhood of a solution s is defined by considering all possible modifications that can be applied to a candidate solution in order to yield a new, different solution. In this case, the neighborhood of s is the set of candidate solutions that can be generated this way. Frequently, modifications introduce rather small changes. A classical example is the two-exchange neighborhood for the traveling salesman problem (TSP), where $N(s)$ consists of all those solutions that can be obtained from s by removing a pair of edges and replacing it by the distinct pair of edges that maintains a tour. Another example for the graph coloring problem is the one-exchange neighborhood of a solution s, which consists of all solutions that can be reached by changing the color of exactly one vertex.

Small modifications have the advantage that their effect is very fast to evaluate and, because of the relatively limited number of neighbors, the neighborhood can be searched rather efficiently. However, the algorithms embedding such neighborhoods have a tendency towards being rather "short-sighted" and they require a large number of search steps to traverse the search space. Therefore sometimes solutions need to undergo some larger, structural changes to improve over the current solution. Such changes are often rather difficult to be managed or found by local search algorithms using small neighborhoods.

Very large-scale neighborhood search techniques try to avoid some of the disadvantages incurred by using small neighborhoods. They allow relatively large modifications of a candidate solution. Therefore, they accept in one search step solutions that are of better quality than those reached within small neighborhoods. Due to the size of very large-scale neighborhoods, often the quality of local minima is much better than that reached in small neighborhoods. Additionally, the ability of making larger modifications to candidate solutions allows for a traversal of the search space in less steps. However, these advantages come at a typically higher computational cost for evaluating the search steps and searching the neighborhood. For many such very large-scale neighborhoods their size increases exponentially with the number of solution components that are allowed to be changed by one single search step. Given this tradeoff between the neighborhood size and the efficiency of searching it, it is difficult to predict the final impact large neighborhoods will have on the performance of more complex stochastic local search algorithms.

The design of algorithms that use very large-scale neighborhoods is based on one of the two main ways of searching improving solutions: exact or heuristic. The exact search is similar to a best-improvement algorithm, i.e. an iterative algorithm that, at each step, tries to move to the best neighboring solution possible. In this case, all neighboring solutions need to be evaluated, at least implicitly. The large size neighborhood can also be examined by some approximate algorithm that does not necessarily identify the best possible neighbor. An important ingredient for such techniques are heuristic ways of restricting a complete enumeration of the neighborhood. In this latter case, improving moves may be missed.

The next section gives a short review of techniques used to define and search very large-scale neighborhoods (VLSN). We will give concise examples for various such techniques. The largest part of this chapter focuses on the experimental analysis of two examples for the usage of very large-scale neighborhoods. The first example studies algorithms for the *(vertex) graph coloring problem* (GCP); see Sect. 3. Initial experimental results show the desirability of an exact search of the neighborhood with respect to solution quality. However, due to its high computational cost we also study various heuristic schemes for examining the proposed neighborhoods. Finally, we show that despite our efforts, once the very large-scale neighborhood search is integrated into an effective stochastic local search (SLS) algorithm [37] for the GCP, no performance improvements could be observed. The situation is different for the second example, (described in Sect. 4), a very large-scale neighborhood scheme used for tackling the *graph set T-coloring problem* (GSTCP). The GSTCP is an extension of the GCP where sets of colors need to be assigned to each vertex under certain types of color separation constraints. In this case, an exact re-assignment neighborhood, which considers for each vertex a complete color reassignment, was shown to contribute for various GSTCP instance classes towards a new state-of-the-art algorithm. The chapter ends with some concluding remarks and avenues for future research on very large-scale neighborhood search techniques.

2 Searching Large Neighborhoods

Using large size neighborhoods is certainly very appealing. However, these neighborhoods may not be very useful if their complete search requires a very high computational effort. Hence, for making the use of large neighborhoods practical, the neighborhood search problem (NSP), i.e. the problem of finding the best solution in a defined neighborhood set, needs to be efficiently solved. From an abstract perspective, one may distinguish exact algorithms for the NSP and heuristic approaches towards solving the NSP.

As previously mentioned, solving the NSP exactly is akin to best-improvement algorithms. The exact solution of the NSP requires to define neighborhoods that, although exponential in size, admit either efficient, that is, polynomial time algorithms for their exploration, or the exploitation of problem-specific knowledge to speed up the exact search as much as possible. One example of polynomial search is given by dynamic programming approaches like in the *dynasearch* algorithm [18, 19, 49]; we will describe the example application of dynasearch to the TSP in Sect. 2.1. Other examples are network flow based improvement algorithms where the examination of the neighborhood is carried out by solving shortest path or matching problems [30, 34, 56].

Many neighborhood definitions result in NSPs that are themselves $\mathcal{NP}$-hard; this is the case for the cyclic exchange neighborhoods [4, 53, 54] that we discuss in Sect. 2.2. Other examples are local branching for 0–1 integer programming problems where a k-flip neighborhood is defined and searched exactly by solving smaller and more constrained integer programming problems [24], or destruction-reconstruction neighborhoods where the reconstruction is carried out by means of constrained-based tree search [51]. In these cases, the search algorithms might have a significant computational cost, which make it advisable to apply pruning techniques or heuristics to truncate somehow the exact search. This results in a heuristic neighborhood exploration, which does not necessarily identify the best neighboring solution. Hence, there is a trade-off between the scrutiny of the neighborhood search and the solution quality reached.

An important class of very large-scale neighborhood search algorithms are variable depth search (VDS) methods. These work on exponentially sized neighborhoods that are obtained by considering concatenations of simple moves. The number of simple moves that are concatenated to obtain one large move is determined while exploring the neighborhood. VDS methods search the neighborhood in a heuristic way without aiming for the exploration of the full neighborhood. In fact, some of the first very large-scale neighborhood search algorithms were VDS methods. The most famous of these are certainly the Kernighan-Lin heuristic for the graph partitioning problem [41] and the Lin-Kernighan heuristic for the TSP [45]. However, additional VDS algorithms have been proposed rather recently and it should be mentioned that the ejection-chain methods for defining very large-scale neighborhoods [31] share many similarities to VDS. In Sect. 2.3 we will give an overview of the main features of the Lin-Kernighan heuristic for the TSP.

The examples we include below can only give a rough impression of the type of techniques that are available for exploring very large-scale neighborhoods. For a more extensive review of the available techniques, we refer to the overview papers by Ahuja et al. [2, 3].

2.1 Dynasearch

This first example illustrates how an exact algorithm, in this case a dynamic programming algorithm, can be used to search efficiently an appropriately defined, exponentially large neighborhood. In fact, dynamic programming can be used to search neighborhoods of a number of different problems and the resulting class of local search algorithms received the name *dynasearch*. Dynasearch searches neighborhoods that consist of all possible combinations of mutually independent simple search moves. Independence in the context of dynasearch means that the individual simple search moves do not interfere with each other with respect to the objective function and to the constraints of the problem. In particular, the gain incurred by a dynasearch move must be decomposable into the sum of the gains of the simple search moves.

So far, dynasearch has only been applied to problems where solutions can be represented as permutations. Independence between simple search moves is given if the last element involved in one move occurs before the first element of the next move. If $\pi = (\pi(1), \ldots, \pi(n))$ is the current permutation, two moves involving elements from $\pi(i)$ to $\pi(j)$ and from $\pi(k)$ to $\pi(l)$, with $1 \le i < j \le n$ and $1 \le k < l \le n$ are independent, if either $j < k$ or $l < i$. If, in addition the contributions of the two moves to the solution cost can be computed independently of each other, the best combination of independent moves can be found by a dynamic programming algorithm. (Without loss of generality, we focus here in this chapter on minimization problems.)

Let $\Delta(j)$ be the maximum total cost reduction incurred by independent moves involving only elements from position 1 to j of the current permutation and $\delta(i,j)$ be the cost reduction resulting from a move involving positions between i and j, including i and j. The maximum total cost reduction is obtained either by appending $\pi(j)$ to the current partial permutation or by appending element $\pi(j)$ and applying a move involving element $\pi(j)$ and an element $\pi(i)$ (and possibly elements from $\pi(i)$ onwards).

We assume that the current solution is given by π and set $\Delta(0) = 0$ and $\Delta(1) = 0$. Then, $\Delta(j+1),\ j = 1, \ldots, n-1$ can, in general, be computed in a forward evaluation using the recursive formula

$$\Delta(j+1) = \max\left\{ \max_{1 \le i \le j} \{\Delta(i-1) + \delta(i, j+1)\}, \Delta(j) \right\} . \qquad (1)$$

The largest reduction in solution cost is then given by $\Delta(n)$ and the single moves to be performed can be found by tracing back the computation steps. In order to apply dynasearch using the forward evaluation, the value of $\Delta(j)$ should not depend on positions $k > j$. If the evaluation of moves depends only on elements $k > j$, a backward version of the dynamic programming algorithm can be applied. This version that starts with $\pi(n)$ and then generates the sequence of $\Delta(j),\ j = n, n-1, \ldots, 1$.

When applying the algorithm, the particularities of the moves have to be taken into account when using Eq. (1). Consider, for example, the application of dynasearch to the TSP using two-exchange moves as the underlying simple moves. Here we assume that a tour is represented as $\pi = (\pi(1), \ldots, \pi(n+1))$, where we define $\pi(n+1) = \pi(1)$. If $1 < i+1 < j \le n$ and $\pi(j+1) \ne \pi(i)$, a two-exchange move is obtained by removing edges $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$ from the current tour π and introducing edges $(\pi(i), \pi(j))$ and $(\pi(i+1), \pi(j+1))$. The move is accepted if the new tour is shorter, that is, if and only if

$$d(\pi(i), \pi(i+1)) + d(\pi(j), \pi(j+1)) > d(\pi(i), \pi(j)) + d(\pi(i+1), \pi(j+1)),$$

where $d(\cdot,\cdot)$ is the cost function defined on the set of edges.

The dynasearch two-exchange algorithm presented by Congram [18] tries to improve a Hamiltonian path between two fixed end points $\pi(1)$ and $\pi(n+1)$.

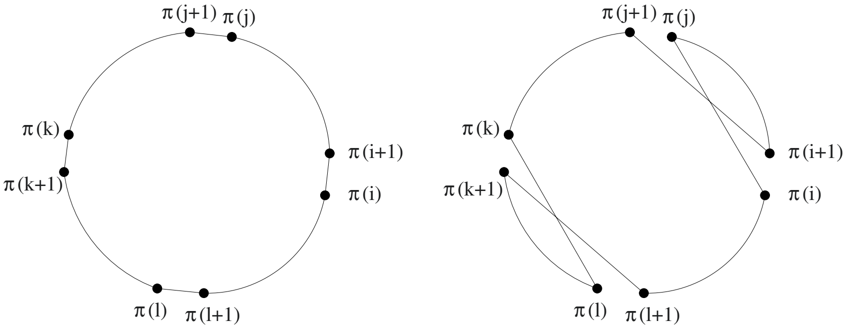


Fig. 1. Example of a dynasearch move that is composed of two independent two-exchange moves

In this case, two two-exchange moves that delete edges $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$, where $1 < i+1 < j \leq n$, $\pi(j+1) \neq \pi(i)$, and $(\pi(k), \pi(k+1))$ and $(\pi(l), \pi(l+1))$, where $1 < k+1 < l \leq n$, $\pi(l+1) \neq \pi(k)$, are independent if we have $j < k$ or $l < i$, because in this case the segments that are rearranged by the two moves do not have any edges in common. An example of such an independent dynasearch move is given in Fig. 1.

In this case, $\Delta(j)$ is the largest reduction of the length of the tour obtained by independent two-exchange moves involving only elements between 1 and j (both included) of the incumbent tour π. The initialization of the dynamic program is $\Delta(1) = \Delta(2) = \Delta(3) = 0$, because we need at least four vertices to apply a two-exchange move. The recursion formula then becomes

$$\Delta(j+1) = \max\left\{ \max_{1 \leq i \leq j-2} \{\Delta(i-1) + \delta(i, j+1)\}, \Delta(j) \right\}$$

and $\Delta(n+1)$ gives the maximal improvement. Dynasearch then repeats these neighborhood searches until no improvement can be found anymore, that is, until a local optimum is reached.

Current applications of dynasearch comprise the traveling salesman problem [18], the single machine total weighted tardiness problem [18, 19], and the linear ordering problem [18]. A general observation from these applications is that dynasearch, on average, is faster than a standard best-improvement descent algorithm and returns slightly better quality solutions. Particularly good performance is reported, if dynasearch is used as a local search routine inside an iterated local search [46]. Currently, iterated dynasearch is the best performing metaheuristic for the single machine total weighted tardiness problem [33] and very good results were also obtained for the traveling salesman problem and the linear ordering problem [18].

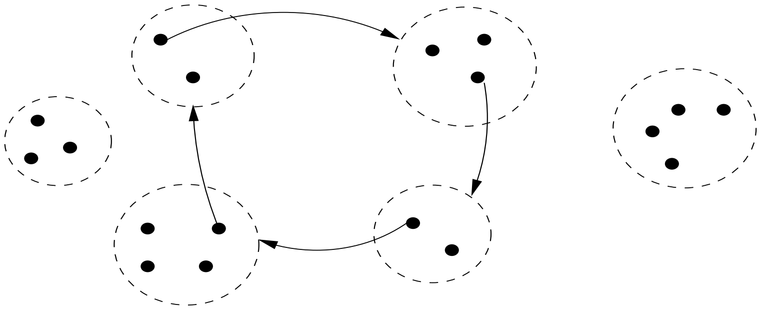


Fig. 2. Example of a cyclic exchange for a partitioning problem

2.2 Cyclic and Path Exchange Neighborhoods

Cyclic and path exchange neighborhoods have been considered as very large-scale neighborhoods for tackling problems where solutions may be represented in some form of a partitioning [4, 5]. Examples of such problems include vehicle routing, capacitated minimum spanning tree, parallel machine scheduling, clustering, and various others.

In a set partitioning problem is given a set W of n elements and a set of subsets $\mathcal{T}$ of W, $\mathcal{T} = \{T_1, \dots, T_K\}$ such that $W = T_1 \cup \dots \cup T_K$ and $T_k \cap T_{k'} = \emptyset$, $k, k' = 1, \dots, K$. A cost $c(T_k)$ is associated with any set T_k. The partitioning problem seeks the partition of W that minimizes the sum of the costs of the subsets in the partition. Formally, the partitioning problem can be defined as:

$$\min\, c(\mathcal{T}) = \sum_{k=1}^{K} c(T_k)$$
$$\text{s.t. } \mathcal{T} \text{ is a partition of } W.$$

The characteristics of the cost function are not important for the time being; its sole property that needs to be taken into consideration is its separability over subsets.

Frequently, partitioning problems are solved using local search algorithms that are in most cases based on the one- and two-exchange neighborhood. Cyclic exchange neighborhoods are a generalization of the two exchange neighborhood [4, 53, 54] in which instead of swapping only two elements from two subsets, several elements, each belonging to a different subset, are moved (see Fig. 2).

Formally, a cyclic exchange between k subsets (without loss of generality we can assume them to be $T_1, \dots, T_k$ and the elements we look at to be $a_1, \dots, a_k$, with $a_i \in T_i$) is represented by a cyclic permutation π of length k, $\pi \neq \mathbf{1}$, where $\pi(i) = j$ means that the element a_i from subset T_i moves into subset T_j. A partition $\mathcal{T}'$ is said to be a *neighbor* of the partition $\mathcal{T}$ if it is obtained from $\mathcal{T}$ by performing a cyclic exchange and it is feasible with respect to some problem specific constraints. The set of all neighbors of $\mathcal{T}$ defines the *cyclic exchange neighborhood* of $\mathcal{T}$. The cyclic exchange modifies the sets of the partition and therefore their cost. The cost difference for each subset will be the difference between the cost of the subset before performing the cyclic exchange and the cost of the subset after the exchange. The *cost of the cyclic exchange* is the sum of all the cost differences over all subsets in the partition.

In order to find the next move, Ahuja et al. define the *improvement (directed) graph*. The construction of the improvement graph depends on the current feasible partition of the partitioning problem considered. The set of vertices V of the improvement graph is defined as the collection of integers $1, \ldots, n$, each corresponding to an element of the set W. The improvement graph contains the arc (i, j) if the elements corresponding to i and j do not belong to the same subset in $\mathcal{T}$ and the subset to which the element corresponding to j belongs remains feasible after the removal of the element corresponding to j and the addition of the element corresponding to i. We define the partition $\mathcal{U}$ of V to be the collection of subsets $U_1, \ldots, U_K$ corresponding to the subsets in $\mathcal{T}$, that is, the elements of T_k are in one-to-one correspondence with the elements of U_k for each $k = 1, \ldots, K$. Therefore a subset disjoint cycle in the improvement graph, with respect to $\mathcal{U}$, will correspond to a cyclic exchange in $\mathcal{T}$. The arc (i, j) will have an associated cost, equal to the difference between the cost of the set after the removal of the element corresponding to j and the addition of the element corresponding to i, and the cost of the original set that contains the element corresponding to j. Thompson and Orlin [53] showed that there is a one-to-one correspondence between the cyclic exchanges with respect to $\mathcal{T}$ and the subset-disjoint cycles in the improvement graph (with respect to $\mathcal{U}$) and that both have the same cost.

The problem of finding the best neighbor within the cyclic exchange neighborhood can be modeled as a new problem, the *subset disjoint minimum cost cycle problem* (SDMCCP), or in some cases the *subset disjoint negative cost cycle problem* (SDNCCP). The SDMCCP is the problem of finding the minimum cost subset disjoint cycle (a cycle that uses at most one vertex from every subset of the partition of the set of vertices) in the improvement graph. The SDNCCP is the problem of finding a minimum subset disjoint cycle with the additional constraint that its cost is negative. The only real difference between SDMCCP and SDNCCP is that the feasible set of the latter is a subset of the feasible set of the former. Also note that if the network contains negative cycles, then the set of optimal solutions of both problems is the same.

If the SDNCCP has a solution, then this solution corresponds to an improvement in the solution quality of the partitioning problem considered. Hence, the SDNCCP is the problem to be solved when using VLSN in an iterative improvement algorithm. The SDMCCP is important if the VLSN search technique is embedded, for example, into a tabu search procedure;

then finding the best neighbor may be important regardless of whether or not this neighbor is better than the current solution. Ahuja et al. [4] solve the SDNCCP by a heuristic that takes advantage of the fact that only negative cycles are needed. However they make no attempt to solve the SDNCCP exactly. Several exact methods for both SDMCCP and SDNCCP are proposed by Dumitrescu [23]. They can be viewed as generalizations of dynamic programming algorithms commonly used for shortest path problems. The search is accelerated in the case of the SDMCCP by taking advantage of symmetry properties, and in the case of SDNCCP by an elegant lemma by Lin and Kernighan [45], which, as noted also by Ahuja et al. [5], is used to restrict drastically the number of extending paths under consideration. Although the SDNCCP and SDMCCP are $\mathcal{NP}$-hard, the proposed methods are very efficient for the subproblems that arise in practical applications of VLSN search methods. Dumitrescu also proposes some heuristics that are derived from the exact methods by limiting or truncating them in some way.

2.3 Lin-Kernighan Heuristic

While in the previous two examples the underling idea is to search the very large-scale neighborhood exactly, variable-depth search methods are designed with the idea of searching the neighborhood in some heuristic way. The probably best known VDS algorithm is the *Lin-Kernighan (LK) algorithm* for the TSP. It is an iterative improvement algorithm that constructs and systematically tests complex local search moves that are composed of simpler local search steps. The LK algorithm, in particular, is based on complex steps that are generated by a sequence of two-exchange steps.

Let us first explain the mechanism that underlies the construction of the complex search steps. This can be best illustrated by considering the sequence of Hamiltonian paths, i.e., paths which visit each vertex in the given graph G exactly once, which are visited in the move generation. (For an illustration consider Fig. 3.) Initially, a Hamiltonian path between vertices u and v is generated from a tour by removing the edge (u, v). One of the two endpoints, say u, is then kept fixed, while the other will vary. This is the situation depicted in Fig. 3a. In a next step, an edge (v, w) is added, which introduces a cycle into the Hamiltonian path (see Fig. 3b). This resulting structure is also called a δ-path. In the third step, the cycle in the δ-path is broken by removing the only possible edge (w, v') incident to w such that the result is a new Hamiltonian path that could be extended to a tour by adding an edge (v', u) (see Fig. 3c). Instead of closing the tour, another edge can be added that leads to a new δ-path, which is indicated in Fig. 3d. The move construction continues in this way by creating a sequence of δ-paths and tentative intermediate tours.

The heuristic examination of the so generated neighborhood is directed in the LK algorithm by considering the weights of the edges in determining which steps are tentatively tested. The LK algorithms starts from some initial tour s, deletes an edge and determines a δ-path p of minimal weight. If the

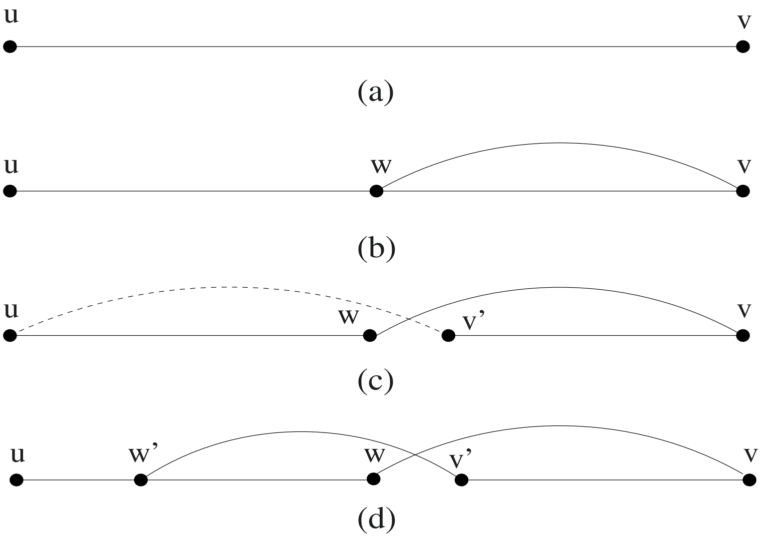


Fig. 3. Schematic view of a Lin-Kernighan exchange step. (a) The original Hamiltonian path; (b) shows a possible δ-path, (c) shows that next Hamiltonian path and (d) shows that a tentative next δ-path

Hamiltonian cycle s' obtained from p in the intermediate steps (as in Fig. 3c) has a weight lower than s, then s' becomes the new incumbent solution. These steps are continued from p and iterated until no δ-path can be obtained with weight smaller than that of the best Hamiltonian cycle found so far in the construction of the complex move. If the best Hamiltonian cycle found in this process improves over the initial solution, it replaces this solution.

In addition to the criterion that stops the construction of complex moves based on the weight of δ-paths and the incumbent solutions, the LK algorithm uses additional tabu criteria. In fact, in the construction of the complex moves any edge that has been added is not removed and no edge that has been removed is added anymore. This rule has the effect that a candidate sequence for a complex step is never longer than the number of vertices in the graph.

In addition to these basic steps, the LK algorithm uses various additional techniques and few exceptions to the above described rule for constructing complex moves. The most important technique is probably a type of backtracking mechanism that allows to consider alternative choices in the edges to be added for constructing the complex moves. Typically, this mechanism is limited to the first few decisions that are done and it is necessary to guarantee that the final tour found by the LK algorithm is locally optimal with respect to the two- and three-exchange neighborhoods. Other mechanisms concern specific types of moves that may be generated are exceptions to the basic construction rules. For a description of these details, we refer to the original article [45] or to other, more recent descriptions of various LK implementations [7, 39, 48].

VDS algorithms have been used with considerable success for solving a number of problems other than the TSP [41, 47, 57]. Generally, however, the implementation of high-performance VDS algorithms requires considerable effort, especially if large instances are to be tackled.

3 Cyclic and Path Exchange Neighborhoods for Graph Coloring

The graph coloring problem (GCP) is a central problem in graph theory [38] and it arises in a number of application areas such as register allocation [6], air traffic flow management [9], frequency assignment [27], light wavelengths assignment in optical networks [59], and timetabling [21, 43].

In the GCP, one is given an undirected graph $G = (V, E)$, with V being a set of $|V| = n$ vertices and E being a set of edges. A *k-coloring* of G is a mapping $\varphi : V \to \Gamma$, where $\Gamma = \{1, 2, \ldots, k\}$ is a set of $|\Gamma| = k$ integers, each representing one color. A k-coloring is *proper* if for all $(u, v) \in E$ it holds that $\varphi(u) \neq \varphi(v)$; otherwise it is *non-proper*. If for some $(u, v) \in E$ we have $\varphi(u) = \varphi(v)$, the vertices u and v are said to be *in conflict*. (Similarly, we say that edge (u, v) is *in conflict*.) The *conflict set* V^c is the set of all vertices that are in conflict. Alternative to the assignment point of view, a k-coloring can also be seen as a partitioning $\mathcal{C} = \{C_1, \ldots, C_k\}$ of the set of vertices V into k disjoint sets, called *color classes*.

In the optimization version of the GCP, one is asked for the smallest number k of colors such that a proper coloring exists. This number is an intrinsic property of a graph and is called *chromatic number*. The decision version of the GCP asks if, for a given k, a proper coloring exists. This version of the GCP is well known to be $\mathcal{NP}$-complete [40]. It is, hence, not surprising that exact methods for solving the GCP fail to be effective on a significant portion of the known benchmark instances, in part due to their large size [14]. As an alternative, a large number of SLS algorithms have been proposed. Most of these approaches are actually based on a very straightforward one-exchange neighborhood, in which one vertex at a time changes its color class. (See [16] for an overview of local search algorithms for the GCP.) However, the partitioning representation of the GCP directly suggests the application of the cyclic and path-exchange neighborhood that was described in Sect. 2.2. In what follows, we study the performance of SLS algorithms exploiting these neighborhoods and give some insights into their behavior.

3.1 Neighborhoods for the GCP

When tackling the optimization version of the GCP by local search methods, the most used approach is to solve a sequence of k-coloring problems. In a first step, one guesses an initial value for k, for example, by applying a construction algorithm based on the DSATUR heuristic [12] or the RLF heuristic [43]. Then, each time a proper k-coloring is found, the value of k is decreased by one. This can be done by removing one color and recoloring the vertices without any associated color uniformly at random. If in this iterative process the SLS algorithm cannot find a proper coloring for some k, it returns $k + 1$ as an upper bound on the chromatic number.

The evaluation function used by most SLS algorithms counts the number of conflicting edges: $g(\mathcal{C}) = \sum_{i=1}^{k} |E_i|$, where E_i is the set of edges with both end points in C_i. A candidate solution with an evaluation function value of zero corresponds to a proper k-coloring.

In local search, a solution to the GCP is represented by a vector of $|V|$ elements containing the color assigned to each vertex. In addition, for speeding up some of the computations in the local search (for example the neighborhood examination), it is advantageous to maintain a redundant representation consisting of a collection of k sets of vertices corresponding to the k color classes.

The basic *one-exchange* neighborhood operator changes the color of one single vertex. In the partitioning representation, this corresponds to moving one vertex from one color class to another. Clearly, improvements of the evaluation function are only possible if exchanges involve vertices that are in conflict. Hence, in many SLS algorithms the neighborhood is restricted to such vertices and the size of the one-exchange neighborhood is $|V^c| \cdot k$. We denote this neighborhood by N_1. (Note that the size of N_1 changes at run-time of an SLS algorithm with the size of the conflict set.)

Much less frequently used is the neighborhood structure based on the *swap operator*, in which, one vertex in V^c exchanges the color with another vertex in V. Besides being of quadratic size in the worst case (more precisely, of size $|V^C| \cdot |V|$), this neighborhood leaves the cardinality of the color classes unchanged and it is therefore less appealing than N_1.

The partitioning representation of the GCP lends itself to the definition of a *cyclic and path exchange* neighborhood as described in Sect. 2.2. Each partition corresponds to one color class and the cost is the number of conflicting edges in the class. The property of cyclic exchanges to leave the class cardinality unchanged suggests the desirability of allowing also path exchanges.

In the GCP, a *cyclic exchange* of length m acts on a sequence of vertices $(u_1, \ldots, u_m)$ belonging to mutually distinct color classes. For simplicity, we denote the color class of any u_i, $i = 1, \ldots, m$ by C_i and adopt the convention $C_{m+1} = C_1$. The *cyclic exchange* moves then any u_i, $i = 1, \ldots, m$ from C_i into C_{i+1}. The *path exchange*, instead, moves any u_i, $i = 1, \ldots, m - 1$ from C_i into C_{i+1} but maintains u_m in C_m. In this latter case, the sequence of exchanges is not closed and the cardinality of C_1 and C_m is modified.

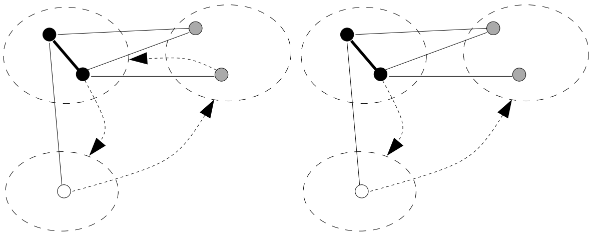


Fig. 4. An example where one-exchange and swap moves do not yield any improvement, while a cyclic (left side) or a path (right side) exchange can be found to yield a proper coloring.

3.2 Cyclic and Path Exchanges and the Number of Local Optima

The new cyclic and path exchange neighborhood for the GCP includes the one-exchange and the swap neighborhood as special cases. Figure 4 gives an example, which shows that by the adoption of the general cyclic and path exchange neighborhoods, improvements over these special cases may be expected. In fact, the one-exchange and swap neighborhoods would fail to yield any neighboring coloring with a better evaluation function value than the coloring depicted in Fig. 4; however, by allowing for cyclic and path exchanges, moves could be found that result in a proper coloring.

Further evidence for the desirability of using the cyclic and path neighborhood is given by experiments on small graphs. In particular, we generated for $n \in \{3, \ldots, 9\}$ all connected, non-isomorphic graphs.[1] In Table 1, we report in the first three columns the number of vertices, the number of such distinct graphs grouped by different chromatic number and for each graph the number of all distinct colorings that use a number of colors between two and the chromatic number.[2] We then applied, starting from these colorings, iterative best improvement algorithms that make use of different neighborhoods (or neighborhood combinations) and for each of these algorithms we counted the number of local optima that do not correspond to proper colorings. For $n \in \{3, \ldots, 7\}$ these experiments were done exhaustively, that is, starting once from each different initial coloring, while for $n \in \{8, 9\}$ we used a random sample of size equal to the number of initial colorings used for the case $n = 7$. The results show clearly that the number of local optima decreases considerably by using a larger neighborhood.

[1] These graphs where generated with the program `geng` of `nauty` by B. McKay, available from `http://cs.anu.edu.au/~bdm/nauty/` (1984-2007, downloaded December 2003).

[2] Given a graph of size n, the number of all colorings that use k colors is given by the Stirling number of second kind $S_{n,k}$ and corresponds to the number of all partitions of n labeled objects (the vertices) into k unlabeled sets (the colors classes). A procedure to generate all these partitions can be found, for example, in [42].

Table 1. Number of distinct local optima that do not correspond to proper colorings for iterative improvement algorithms that differ in the examined neighborhoods. Results pertain all connected, non-isomorphic graphs of different size. Iterative improvement algorithms are run on each graph starting once from each possible distinct coloring that use between two and the chromatic number of colors. For n in $\{8, 9\}$ we considered an overall sample of 421 555 distinct initial colorings.

$\|V\|$	# distinct graphs	# distinct colorings	One-ex	One-ex + swap	One-ex + cyclic	swap + path	cyclic + path
3	2	7	0	0	0	0	0
4	6	61	4	1	1	0	0
5	21	756	25	14	9	0	0
6	112	14113	468	231	148	24	14
7	853	421555	9129	4768	2419	495	265
8	11117	22965511	81944	49416	24879	8006	4913
9	261080	2461096985	133535	85169	42077	18964	11890

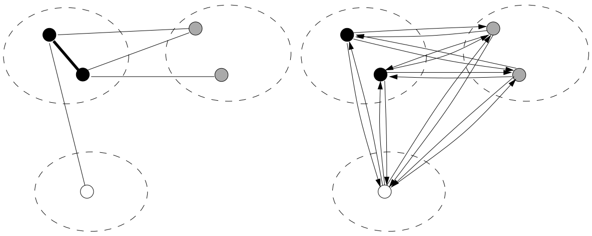


Fig. 5. A graph and a coloring for it (left) and the corresponding improvement graph (right).

3.3 Neighborhood Examination

A crucial element in the examination of the neighborhood is the computation of only local changes in the evaluation of a move. In the one-exchange neighborhood, the evaluation function value of a neighbor $\mathcal{C}'$ of the current coloring $\mathcal{C}$ can be obtained by the value of $\mathcal{C}$: $g(\mathcal{C}') = g(\mathcal{C}) - |A_{C_i}(v)| + |A_{C_j}(v)|$, where $A_{C_i}(v)$ is the set of edges connecting v to other vertices in the class C_i and we assume to move v from C_i to C_j. The value $|A_{C_i}(v)|$ can be obtained in constant time if we record it in an auxiliary matrix $\boldsymbol{\Delta}$ of $|V| \times k$ elements. The initialization of the matrix is done once at the beginning in $\mathcal{O}(|V|^2)$; its update after a one-exchange move is, however, much faster since we need to consider only the entries corresponding to the vertex that moved into a different color class and the vertices adjacent to it. Hence, the worst case complexity for updating $\boldsymbol{\Delta}$ is $\mathcal{O}(|V|)$, although in practice it is much lower.

In the examination of the cyclic and path exchange neighborhood, we can use the improvement graph model and the SDNCCP algorithm devised by Dumitrescu [23]. The improvement graph is the directed graph $G' = (V', D')$, obtained from the graph $G = (V, E)$ and the current color partition $\mathcal{C} =$

$\{C_1, \ldots, C_k\}$. The set of vertices of G' is $V' = \{1, \ldots, n\}$, each vertex in V' corresponding to exactly one vertex $v_i \in V$. The set of arcs D' consists of arcs (i, j) if vertices v_i and v_j belong to two different color classes in $\mathcal{C}$. The set of vertices V' is split into a partition $\mathcal{U}$ of k subsets $U_1, \ldots, U_k$, induced by $\mathcal{C}$, and the elements of U_h are in one-to-one correspondence with the elements of C_h, $\forall h = 1, \ldots, k$ (see Fig. 5). Hence, cyclic and path exchanges in $\mathcal{C}$ will correspond to subset disjoint cycles, respectively paths, with respect to $\mathcal{U}$. The cost associated to an arc (i, j) with $i \in U_i$ and $j \in U_j$ is the cost of inserting vertex i into C_j and removing vertex j from C_j. This cost can be easily computed using the matrix $\boldsymbol{\Delta}$ defined above: $c_{i,j} = |A_{C_j}(v_i)| - |A_{C_j}(v_j)|$. The update of $\boldsymbol{\Delta}$ after each cyclic or path exchange is done by considering the exchanges as a composition of one-exchanges and requires $\mathcal{O}(m|V|)$), where $m \leq k$ is the length of the cyclic or path exchange.

In order to solve the SDNCCP and to find the best cyclic exchange we base the algorithm on the one presented in [23] and another similar one [4]. The algorithm works in a dynamic programming fashion by extending *subset disjoint paths*, that is, paths that visit every subset U_i of the improvement graph $G'(V', D')$ at most once. It is clear that the path can be closed to form a subset disjoint cycle by an arc that connects the last and the first vertex of such a path. This arc always exists given the way the improvement graph is constructed.

We denote a path in the improvement graph by $p = (i_1, \ldots, i_l)$, where i_1 is the start vertex of p, denoted by $s(p)$ and i_l is the end vertex of p, denoted by $e(p)$. We associate a binary vector $w(p) \in \{0, 1\}^k$ with the path p, where $w_j(p) = 1$, if and only if p visits the subset U_j. The cost of p, denoted by $c(p)$, is the total cost of the arcs in the path, that is, $c(p) = \sum_{j=1}^{l-1} c_{i_j, i_{j+1}}$.

We say that the path p_1 *dominates* the path p_2 if $s(p_1) = s(p_2)$, $e(p_1) = e(p_2)$, $c(p_1) \leq c(p_2)$, $w(p_1) \leq w(p_2)$, and the paths do not coincide. This definition will help us to discard paths that are not promising. We define a *treatment* of a path as the extension of that path along all outgoing arcs. After a treatment, a path is marked as *treated*. At any time, only paths that have not previously been treated are selected for treatment. The overall SDNCC algorithm is given in Alg. 16. In the algorithm, q^* contains the ordered sequence of vertices for the most negative cyclic exchange in $\mathcal{C}$ and c^* is its cost.

The algorithm uses a well known lemma from [45] which establishes that if a sequence of edge costs has negative sum, there is a cyclic permutation of these edges such that every partial sum is negative. This allows to restrict the dynamic programming recursion by (i) creating a label of length one only for negative edges (line 2) and (ii) extending a label of length larger than one with a new edge only if the partial sum of the costs associated with the edges remains negative (line 15). The application of this rule does not cause the omission of any promising subset disjoint negative cost cycle. If, on line 11, all the paths in $\mathcal{P}$ are transferred for examination into $\widehat{\mathcal{P}}$ without any restriction, the algorithm is exact and its complexity is $\mathcal{O}(|V'|^2 2^k |D'|)$.

Algorithm 16 to solve the SDNCCP

```
 1: input: a graph $G'(V', D')$
 2: Let $\mathcal{P}$ all negative cost paths of length 1, i.e., $\mathcal{P} = \{(i,j) : (i,j) \in D', c(i,j) < 0\}$
 3: Mark all paths in $\mathcal{P}$ as untreated
 4: Initialize the best cycle $q^* = ()$ and $c^* = 0$
 5: for each $p \in \mathcal{P}$ do
 6:    if $(e(p), s(p)) \in D'$ and $c(p) + c(e(p), s(p)) < c^*$ then
 7:       $q^*$ = the cycle obtained by closing $p$ and $c^* = c(q^*)$
 8:    end if
 9: end for
10: while $\mathcal{P} \neq \emptyset$ do
11:    Let $\widehat{\mathcal{P}} = \mathcal{P}$ be the set of untreated paths
12:    $\mathcal{P} = \emptyset$
13:    while $\exists\, p \in \widehat{\mathcal{P}}$ untreated do
14:       Select some untreated path $p \in \widehat{\mathcal{P}}$ and mark it as treated
15:       for each $(e(p), j) \in D'$ s.t. $w_{\varphi(v_j)}(p) = 0$ and $c(p) + c(e(p), j) < 0$ do
16:          Add the extended path $(s(p), \dots, e(p), j)$ to $\mathcal{P}$ as untreated
17:          if $(j, s(p)) \in D'$ and $c(p) + c(e(p), j) + c(j, s(p)) < c^*$ then
18:             $q^*$ = the cycle obtained by closing the path $(s(p), \dots, e(p), j)$
19:             $c^* = c(q^*)$
20:          end if
21:       end for
22:    end while
23:    for each $p' \in \mathcal{P}$ subject to $w(p') = w(p)$, $s(p') = s(p)$, $e(p') = e(p)$ do
24:       Remove from $\mathcal{P}$ the path of higher cost between $p$ and $p'$
25:    end for
26: end while
27: return: a minimal negative cost cycle $q^*$ of cost $c^*$
```

Unfortunately the algorithm SDNCC cannot be modified efficiently to search exactly also path exchanges in $\mathcal{C}$. This is due to the way we constructed the improvement graph. Indeed, the cost of a subset disjoint path in the improvement graph does not correspond to the cost of a path exchange in the real graph and an adjustment is required. Given the subset disjoint path $p = (i_1, \dots, i_l)$ in G', which corresponds to $\tilde{p} = (v_{i_1}, \dots, v_{i_l})$ in G, the evaluation function of a neighbor $\mathcal{C}'$ obtained from $\mathcal{C}$ after a path exchange given by $\tilde{p}$ is: $g(\mathcal{C}') = g(\mathcal{C}) + c(p) - |A_{C_{i_1}}(v_{i_1})| + |A_{C'_{i_l}}(v_{i_l})|$, where $C'_{i_l} = C_{i_l} \cup \{v_{i_{l-1}}\}$. In order to find the best (most improving) cyclic *or* path exchange in $\mathcal{C}$ we must then modify algorithm SDNCC such that it checks also the cost of paths whenever it checks the cost of cycles, on lines 6 and 17. Moreover, in an exact search we must also avoid to use the lemma of Lin and Kernighan [45], as it would not be anymore valid given that the cost of subset disjoint paths in $\mathcal{U}$

does not correspond to the cost of path exchanges in $\mathcal{C}$. In the final output of the algorithm thus modified, q^* becomes the most negative cyclic or path exchange in $\mathcal{C}$ and c^* its cost.

The algorithm in Alg. 16 returns the most improving neighboring solution for the considered neighborhoods. However, the computational cost of the exact neighborhood exploration may be prohibitively high and therefore various heuristics to prune the search have been implemented. The first is to limit the number of paths that are explored to a maximum of `LAB_LIM` paths. This can be achieved by modifying line 11 of Alg. 16 to let $\widehat{\mathcal{P}} \subseteq \mathcal{P}$ be the set of the `LAB_LIM` cheapest untreated paths. In this way the final solution is not optimal (although it remains very close [23]) but the time complexity drops to $\mathcal{O}(|V'|^2)$. In addition, it is clear that the Lin and Kernighan lemma might be very helpful to further reduce the computational cost of the algorithm. We therefore investigated the usage of the following further rules to prune the search.

Rule 1a. Only subset disjoint negative paths of length one are treated on line 2 and only subset disjoint paths of negative or null cost are treated on line 15.

Rule 1b. Only subset disjoint negative cost paths are treated on both lines 2 and 15 as from lemma of [45].

Rule 2. We introduce the restriction to maintain no more than `LAB_LIM` $< \infty$ paths on line 11.

Rule 3. When considering an iterative improvement procedure, we search for cyclic and path exchanges and hence run the algorithm SDNCC only if the best one-exchange is a non-improving move.

Using the same experimental setting as described Table 1 we compared the number of local optima determined with an exhaustive search of the cyclic and path neighborhood (hence the numbers in the last column of Table 1) against those determined by a search pruned by the combined application of the rules 1a+2+3 and 1b+2+3. In both these two latter cases we could not detect any deterioration of performance, that is, the number of local optima remained the same.

Small graphs, however, are not very helpful to detect the impact of the parameter `LAB_LIM`. Preliminary computational analysis on common benchmark graphs with at least 125 vertices indicated, as expected, that the number of local optima decreases by augmenting `LAB_LIM`. As `LAB_LIM` increases, also the number of iterations to reach local optima decreases; nevertheless, the overall computation time increases strongly. Furthermore, the decrease of the number of local optima becomes less strong as `LAB_LIM` increases. We decided, therefore, to fix `LAB_LIM` to a value of 50 which yields a reasonable trade off between solution quality and running time [14].

In Table 2, we study the impact in computation time for the different neighborhood restrictions in iterative improvement algorithms. The experiments were conducted on uniform random graphs of 125 vertices with three

Table 2. Results on the instances `DSJC125` with edge densities $\delta \in \{0.1, 0.5, 0.9\}$. The table reports for each instance, the performance of iterative best improvement in solving the decision problem associated with different values for k. The indicators for the performance are: the percentage (%) success rate of finding a proper coloring; the median number of conflicting edges $\widetilde{g}$; and the median CPU time expressed in hundredth of seconds to complete a single run (the machine used for the experiment is a 2 GHz AMD Athlon MP Processor with 256 KB cache and 1 GB RAM).

		One-ex			One-ex + swap			cyclic + path			cyclic + path			cyclic + path		
ρ	k							Exhaustive			1a + 2 + 3			1b + 2 + 3		
		%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.	%	$\widetilde{g}$	hsec.
0.1	9	74	1	0	93	1	0	100	–	107	100	–	26	100	–	1
	8	29	1	0	64	1	0	100	1	119	100	1	29	100	1	3
	7	2	4	0	12	2	0	88	1	134	91	1	37	87	1	5
	6	0	10	0	0	6	0	12	2	152	14	2	48	12	2	12
	5	0	23	0	0	18	0	0	13	165	0	13	61	0	13	15
0.5	25	1	4	1	7	3	1	100	1	166	82	1	46	96	1	4
	23	0	8	1	0	6	1	96	1	180	53	1	49	81	1	8
	21	0	14	1	0	11	1	56	1	204	16	2	53	33	1	12
	19	0	23	1	0	19	1	0	6	220	0	6	58	0	6	17
	17	0	36	1	0	31	1	0	17	232	0	17	61	0	17	19
0.9	50	0	15	1	0	9	1	63	1	172	3	3	102	22	2	19
	48	0	19	1	0	12	1	35	2	184	1	5	101	10	2	21
	45	0	23	1	0	18	1	0	5	249	0	8	100	0	5	24
	42	0	29	1	0	24	1	0	11	218	0	12	95	0	11	23
	39	0	37	1	0	31	1	0	18	225	0	19	96	0	18	26
	36	0	49	1	0	41	1	0	28	233	0	28	93	0	28	27
	33	0	62	1	0	54	1	0	40	238	0	40	86	0	40	27
	30	0	77	1	0	70	1	0	55	242	0	55	8	0	55	28

different edge densities, namely 0.1, 0.5, 0.9. Results are presented in terms of the ability to solve the various decision problems encountered when decreasing k to the lowest value of k for which a proper coloring is known to exist. At each k, all iterative improvement algorithms are started from the same 1000 initial solutions. The data reported are the success rates for solving the decision problem for each value of k, the median number of conflicting edges in the cases where no proper coloring was found, and the computation time to reach a local optimum. We observe that the contribution of the new neighborhoods remains important also in the large graphs and that it remains pronounced even with a pruned neighborhood examination. Looking at the effect of the two rules 1a and 1b, we observe a considerable difference in computation time and also, rather unexpectedly, higher capacity to solve the problem for the rule 1b, which is therefore to be preferred.

In Table 3 we try to gain deeper insight into which kind of moves are the most important in the new neighborhood. The experimental setting is the same as for Table 1. This time we distinguish two versions for exhaustive

Table 3. Number of moves chosen from each neighborhood. The experimental setting is the same as for Table 1 except for the overall sample of distinct initial coloring that has here size 14 113. In parenthesis is given the maximal length registered for cyclic and path exchanges. Note that a cyclic exchange of length 3 corresponds to $\{v_1, v_2, v_3\}$ and entails that 3 vertices change colors; a path exchange of length 3 corresponds to $\{v_1, v_2, v_3, v_4\}$ and also entails that 3 vertices change colors, as v_4 serves only to determine the arrival color for v_3.

	n	cycle, path exhaustive	path, cycle exhaustive	cycle, path truncated
one-ex	6	24981	24981	24981
	7	29483	29484	29483
	8	34200	34202	34200
	9	38944	38928	38944
swap	6	794	794	794
	7	1138	1138	1138
	8	1489	1491	1489
	9	1859	1861	1859
cyclic	6	370 (3)	65 (3)	370 (3)
	7	743 (4)	103 (3)	743 (4)
	8	1082 (4)	164 (4)	1082 (4)
	9	1449 (5)	218 (4)	1449 (5)
path	6	684 (3)	989 (3)	684 (3)
	7	775 (3)	1415 (3)	775 (3)
	8	936 (3)	1857 (3)	936 (3)
	9	1034 (3)	2276 (4)	1034 (3)

search, depending on whether ties are broken in favor of cyclic or path exchanges (preference is given to the first listed of the two). For cyclic and path exchanges we report also the maximal length of the exchange registered.

By comparing the third and fourth columns in the table, we observe that if paths are taken when they have equal gain as the cyclic exchanges, then the number of cyclic exchanges drops considerably. This indicates that the use of path exchanges alone would miss less gains than the use of cyclic exchanges alone. Hence, the contribution of path exchanges seems higher than the contribution of cyclic exchanges.

The maximal length of cyclic and path exchanges registered indicates that for small graphs of size 9 and number of color classes less or equal to 9, up to 5 vertices may have to change color at the same time in a cyclic exchange and up to 4 in path exchanges. We take this as a further evidence that searching beyond the one-exchange neighborhood might be profitable, as an improving exchange can be found only by moving several vertices. As mentioned, no impact was detected by truncation rules on these small graphs.

3.4 Integrating Cycle and Path Exchanges into SLS Algorithms

The results in Table 2 indicate that the simple iterative improvement algorithms perform rather poorly compared with the results achieved by SLS algorithms. In fact, many SLS algorithms find proper coloring for the smallest values of k indicated in the table within a few seconds on an office PC as of 2005 [14]. It is therefore important to test the use of the new neighborhood within SLS algorithms.

For the GCP, rather simple tabu search algorithms have shown very good and robust performance over a wide range of different graphs. A first tabu search algorithm based on the one-exchange neighborhood was proposed by Hertz and de Werra [36] and it was later improved leading to the best performing tabu search variant [22, 26], which we denote by $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ chooses at each iteration a best non-tabu or tabu but aspired neighboring candidate solution from the restricted one-exchange neighborhood. If a one-exchange move puts vertex v from color class C_i into C_j, it is forbidden to re-assign vertex v to C_i in the next tt steps; the tabu status of a neighboring solution is overruled if it improves over the best candidate solution found so far (aspiration criterion). If more than one move produces the same effect on the evaluation function, one of those moves is selected uniformly at random. The tabu list length in $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ is set to $tt = random(10) + \delta \cdot |V^C|$, where $random(10)$ is an integer uniformly chosen from $\{0, \ldots, 10\}$ and δ is a parameter typically set to 0.5. Since tt depends on $|V^C|$, the tabu list length varies dynamically with the evaluation function value.

To test the usefulness of the cyclic and path exchange neighborhoods, we integrated them into a tabu search algorithm, which we denote as $\mathsf{TS}_{\mathsf{VLSN}}$. $\mathsf{TS}_{\mathsf{VLSN}}$ is closely related to $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. It first selects the best non-tabu move in the one-exchange neighborhood. If this move has the same evaluation function value as the current one, the cyclic and path exchange neighborhood is searched. In all other cases, that is, if the best non-tabu one-exchange leads to an improvement or a deterioration of the evaluation function, the one-exchange move is applied and the tabu list is updated.

The algorithm SDNCC is modified by including in the initial set $\mathcal{P}$ of line 2 only paths of length one consisting of at least one vertex from V^c. The tabu mechanism is applied to the search for cyclic and path exchanges by discarding any neighboring candidate solution that involves the reassignment of a vertex to some color class that is currently tabu. The tabu list is updated by considering the path or the cyclic exchange as a composition of one-exchanges and the value of tt for a specific vertex–color class pair (v, i) is chosen using the rule of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. Yet, contrarily to $\mathsf{TS}_{\mathsf{1\text{-}ex}}$, $\mathsf{TS}_{\mathsf{VLSN}}$ does not use an aspiration criterion. This allows to discard vertices that are tabu very soon in the search for subset disjoint paths.

The comparison of SLS algorithms is generally performed on the set of benchmark instances available at `http://mat.tepper.cmu.edu/COLOR04`. This repository includes quite heterogeneous graphs and a full account of the results is given in [14]. Here, we focus on two types of random graphs: geometric (G) graphs and uniform (U) graphs. The geometric graphs are generated from points in a two dimensional grid with random, integer coordinates in $[0, 1)$. There is a one-to-one correspondence between points and vertices in the graph. Edges are assigned between pairs of vertices if the Euclidean distance between the corresponding points is less or equal to a value d. The uniform graphs are generated by including each of the $\binom{|V|}{2}$ possible edges independently with probability p. For both types of graphs, we fixed the parameter d and p such that three different edge densities are considered: 0.1, 0.5 and 0.9. A number of graphs (see strip text in the plots of Fig. 6) with 1000 vertices are generated and the algorithms are run once on each instance with a fixed time limit, equal for all algorithms. The time limit of 150, 700, and 1500 seconds[3] for graphs of density 0.1, 0.5, 0.9, respectively, was chosen so as to allow $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ to accomplish about $10000 \times |V|$ iterations. Further improvements after such long runs become unlikely for $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. As reference, we add in the analysis two algorithms that have shown the best results for the classes of instances studied. These are the hybrid evolutionary algorithm [26] and a guided local search [14].

In Fig. 6, we report the results of a rank-based analysis. In particular, we rank within each instance the results in terms of the lowest k for which a proper coloring is found by each algorithm. In this way, the problem of different instance scales is removed. We then aggregate the ranks over the instances by computing the average rank for each algorithm. A lower average rank means that the algorithm performs better. Moreover, we use simultaneous confidence intervals extended around the average values to provide an indication of the statistical significance of the differences. These intervals are determined by an all-pairwise comparison procedure carried out through the Friedman statistical test at a 5 % protected level of significance [15],[20, p. 371]. In the figure, two algorithms are significantly different if the confidence intervals of the corresponding average ranks do not overlap.

These results indicate that the usage of the cyclic and path exchange neighborhoods does not result into improved performance of the tabu search algorithm. On the contrary, for the uniform graphs the performance of $\mathsf{TS}_{\mathsf{VLSN}}$ is significantly worse than that of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. The same is true for the geometric graphs of density 0.5. Only on the geometric graphs with densities 0.1 and 0.9, $\mathsf{TS}_{\mathsf{VLSN}}$ yields statistically similar performance to $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. Hence, we obtain the opposite result to what we may have expected from the previous analysis.

A possible explanation for this result is given in Fig. 7, where we depict the behavior of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ and $\mathsf{TS}_{\mathsf{VLSN}}$ from two different perspectives: over time and over iteration number. It can be observed that $\mathsf{TS}_{\mathsf{VLSN}}$ remains competitive comparing results based on an iteration number, but it is not competitive when comparing the development of the solution quality based on computation

[3] Times refer to a computer with 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB of RAM memory.

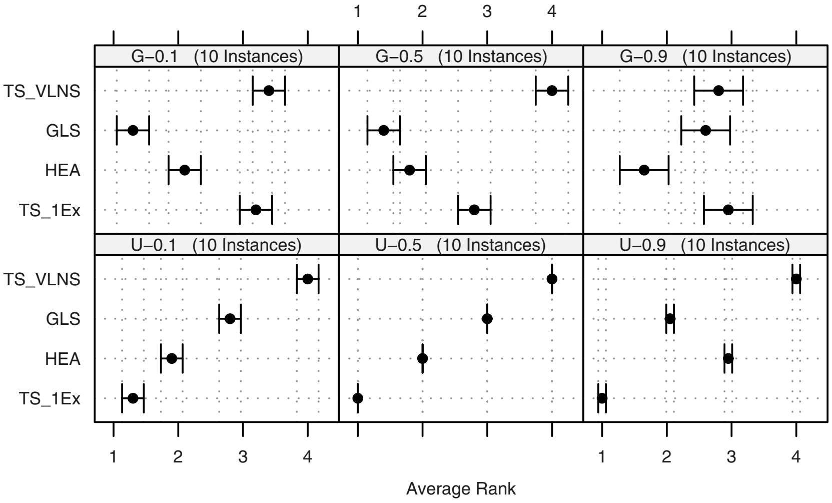


Fig. 6. The 95 % confidence intervals for the all pairwise comparisons of SLS algorithms on aggregated geometric (indicated by G) and uniform (U) random graphs in the GCP. The x-axis indicates the average rank while the confidence intervals are derived from the Friedman test. The more the interval is shifted towards the left the better is the algorithm performance.

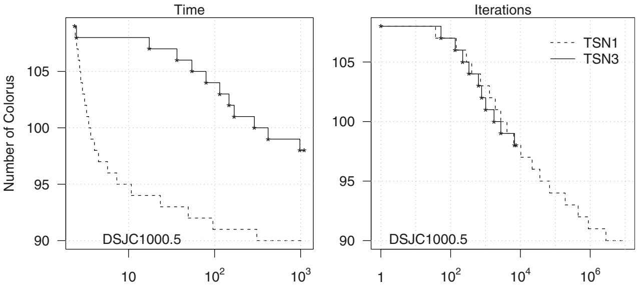


Fig. 7. The development of the solution quality of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ and $\mathsf{TS}_{\mathsf{VLSN}}$ over time (left) and number of iterations (right). Given is the development of the median number of colors across 10 runs per algorithm.

time. This is due to the high added computational cost of searching the cyclic and path exchange neighborhoods. Ideally, this increased computation cost of the neighborhood exploration should be payed off by a faster improvement in solution quality. Although this seems to be the case in the right figure, where the curve of solution cost for $\mathsf{TS}_{\mathsf{VLSN}}$ is slightly lower than the one for $\mathsf{TS}_{\mathsf{1\text{-}ex}}$, this minor improvement is probably not enough to yield any concrete advantage inside $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ for the GCP.

A cause for the weak performances of $\mathsf{TS}_{\mathsf{VLSN}}$ might be the inefficient use of the SDNCC algorithm in the search for path exchanges in $\mathcal{C}$. In a late discovery we found a way to improve on this issue. In detail, the construction of the improvement graph can be modified by adding k vertices, $v_{n+1}, \ldots, v_{n+k}$ one for each of the U_i partitions. These are "dummy" vertices in the sense that they do not correspond to any vertex in V but are useful to represent the cases in which a vertex enters in a class C_i and no vertex leaves it. The cost of the arcs connecting the new vertices with other vertices of G' not in the same class is defined by: $c_{i,n+l} = |A_{C_l}(v_i)|$ for $l = 1, \ldots, k$; $c_{n+l,j} = -|A_{C_j}(v_j)|$ for $l = n+1, \ldots, n+k$; and $c_{n+h,n+l} = 0$ for $l, h = 1, \ldots, k$. In this way it is possible to model in the improvement graph the costs of path exchanges and the lemma by Lin Kernighan in the SDNCC algorithm turns valid again. Note that the inclusion of edges $(n+h, n+l)$ for $l, h = 1, \ldots, k$ with cost zero is not necessary but it permits to find, eventually, more than one independent exchanges in a single run of the SDNCC algorithm (in a dynasearch fashion). However, in spite of all these favorable premises, preliminary experiments resulted in the same conclusions, that is, $\mathsf{TS}_{\mathsf{VLSN}}$ does not improve over $\mathsf{TS}_{\mathsf{1\text{-}ex}}$.

3.5 Other Studies in the Literature

Beside the work presented here, there are few other studies in the literature on very large-scale neighborhood structures for the GCP.

Glass and Prügel-Bennet [29] observed that permuting the colors within a subgraph does not affect the contribution to the evaluation function of the subgraph itself. They devised a permutation neighborhood that first determines a subgraph that defines a partition of the graph and a cut of the edges joining vertices in the subgraph to vertices outside the subgraph. Then, the permutation of colors within the subgraph that minimizes the number of conflicting edges in the cut is found by solving a matching problem in a suitably defined bipartite graph. Tests on this neighborhood structure used together with the one-exchange and enhanced by a perturbation mechanism in an iterated local search fashion failed to show this approach to be competitive.

Gonzalez-Velarde and Laguna [32] proposed an ejection-chain neighborhood which implements cyclic exchanges of vertices but they limit the chains to maximum length 3. Avanthay, Hertz and Zufferey [8] propose several distinct neighborhoods some of which are of exponential size. Nevertheless they do not provide any insight on how to search them efficiently. In none of these cases results are competitive.

Trick and Yildiz [55] devise α-β-swaps and α-expansions. The former consist of exchanging the color of a subset of vertices that belong to two different color classes defined by α and β. The latter consists of changing the color of any set of vertices to α. It is shown that finding the best neighbor in both these neighborhoods corresponds to solving a MAX-CUT problem in an opportunely defined graph. In the experiments the authors use a heuristic algorithm for solving the MAX-CUT problem. Although promising, the preliminary results remain inferior to the state-of-the-art.

Other neighborhoods, though not of exponential size, have been studied in Gendron, Hertz, and St-Louis [28]. They build on the fact that the problem of finding the chromatic number of a graph is equivalent to the problem of finding a circuit-free orientation of the edges of a graph so that the length of the shortest path in the resulting digraph is minimum. Hence they study four neighborhoods in the edge orienting problem and derive graph theoretical properties that allow a polynomial search of each neighborhood. In the results presented, the best neighborhood consists of the reversal of a single arc on a longest path of the current orientation. However, again, the results are not competitive with the state-of-the-art.

We can therefore say that various types of very large-scale neighborhoods have been studied for the GCP but none of these studies could clearly identify any positive impact of these very large-scale neighborhoods for improving state-of-the-art algorithms for the GCP. Hence, our study adds further evidence that the improvement in the solution quality per local search step given by the usage of very large-scale neighborhoods does not pay off the additional computation time required for searching the neighborhoods for the GCP.

4 Color Reassignment Neighborhood for Graph Set T-Coloring

The *graph set T-coloring problem* (GSTCP) generalizes the *graph coloring problem* in the sense that it asks for the assignment of sets of integers to the vertices of a graph. The assignment must satisfy constraints on the separation of any two numbers assigned to a single vertex or to adjacent vertices. The GSTCP arises in the modeling of various real-life problems, the most important being the assignment of frequencies to radio transmitters when designing mobile phone networks. In this case, vertices represent transmitters and colors the frequencies to be assigned to the transmitters subject to certain interference constraints, the T-constraints [35]. Other applications stem from traffic phasing and fleet maintenance [50], or task assignment in which a large task is partitioned into incompatible subtasks and a set of time periods needs to be assigned to each subtask so that incompatible subtasks are in different time periods [52].

More formally, in the GSTCP we are given: (i) an undirected graph $G = (V, E)$, where V is the set of $n = |V|$ vertices and E is the set of edges, (ii) a set Γ of nonnegative integer numbers (called colors), (iii) a number $r(v)$ of required colors at each vertex $v \in V$, and (iv) a collection T (called T-set) of nonnegative integers including zero, such that there is an integer t_{uv} for each edge $(u, v) \in E$ and an integer t_u for each vertex $u \in V$ representing the allowed *separation distances* of colors between and within vertices. The decision version of the GSTCP asks for a mapping $\varphi : V \to \mathcal{P}(\Gamma)$ such that the three groups of constraints

$$|\varphi(v)| = r(v) \qquad \forall\, v \in V \tag{2}$$
$$|x - y| \geq t_u \qquad \forall\, u \in V,\ \forall\, x, y \in \varphi(u), x \neq y \tag{3}$$
$$|x - y| \geq t_{uv} \qquad \forall\, uv \in E,\ \forall\, x \in \varphi(v),\ \forall\, y \in \varphi(u) \tag{4}$$

are satisfied. We call such a multi-valued function φ a *proper set T-coloring*, if all these constraints are satisfied and *improper*, otherwise. The three groups of constraints to be satisfied are called *requirement constraints*, *vertex constraints*, and *edge constraints*, respectively.

We consider the optimization version of the GSTCP in which we are asked for a proper set T-coloring of minimal span, that is, a φ of minimal maximal difference between the colors used, $\min_\varphi \max_{u,v \in V}\{|x - y| : x \in \varphi(u), y \in \varphi(v)\}$. Without loss of generality we fix $\min_{u \in V}\{x : x \in \varphi(u)\}$ to 1. Hence, $\Gamma = \{1, 2, \ldots, k\}$ and the span is $k - 1$. Our task is minimizing k.

4.1 Neighborhood Definitions

A possible approach for applying local search to the GSTCP is to solve a sequence of decision problems, in which a proper set T-coloring is searched for a fixed number k of available colors. This approach has been shown in the literature to be preferable to some others [14]. In this case, a solution is represented by a vector of colors of length $\sum_{v \in V} r(v)$ and an array of pointers indicating where the set of colors assigned to each vertex starts. The effective search space is reduced to only those candidate colorings that satisfy the vertex constraints [14]. Hence, requirement constraints and vertex constraints are always satisfied and the evaluation function needs only to count the number of unsatisfied edge constraints.

The standard one-exchange neighborhood N_E is defined by the operator that changes a single color at a vertex without breaking any vertex constraint. We also studied a new very large-scale neighborhood for the GSTCP. This neighborhood is defined by the operator that reassigns all the colors of one single vertex. In particular, the reassignments are those that satisfy the requirement, vertex and edge constraints acting on the vertex. As such, the application of this operator can be seen as an exact solution to the subproblem of finding an assignment of k colors to one vertex such that no constraint acting on this vertex is violated and no other vertex changes its color assignment. Clearly, given a current configuration, it is possible that for a vertex a reassignment of colors that satisfies all constraints on that vertex does not exist. In this case, if only vertices involved in at least one conflict are considered, the neighborhood is empty. We call this neighborhood the vertex color reassignment neighborhood and we denote it by N_R.

4.2 Neighborhood Examination

Next, we describe the algorithm for the examination of N_R. First a vertex, involved in at least a conflict, is chosen. Then a set F is constructed, which contains the colors that are feasible with respect to the constraints acting on the vertex. This can be done in $\mathcal{O}(|V|k)$ if standard speed-up techniques from the GCP are used [25]. Finding $r(v)$ colors in F that satisfy the vertex constraints, that is finding one neighbor in N_R, is easy: order the values in F, scan F, and remove all colors that are not distant enough from the previous ones. This procedure is deterministic and, unfortunately, it yields the same color reassignment if a vertex is visited twice and its adjacent vertices have not changed the colors. To avoid this cycling behavior, one may introduce some randomization in the reassignment and, hence, when visiting a vertex a second time, a different configuration is obtained, which possibly can be profitably propagated. An implementation of this strategy requires to determine all subsets of F of size $r(v)$ that satisfy the vertex constraints and to pick one at random. More formally, this problem can be formulated as finding a *subsequence of length L of H integers with mutual distance not smaller than D:* given an arbitrary sequence of integers $s = \{s_1, \ldots, s_H\}$, find a subsequence $l = \{l_1 \ldots l_L\}$ of length L with $l_i \in s$, $\forall\, i = 1, \ldots, L$ and such that the mutual distances are not smaller than D, that is, having $|l_i - l_j| \geq D$, $\forall i, j$. Hence, finding all such subsets of F is the same problem as *enumerating all subsequences of length L of H integers with mutual distance not smaller than D.*

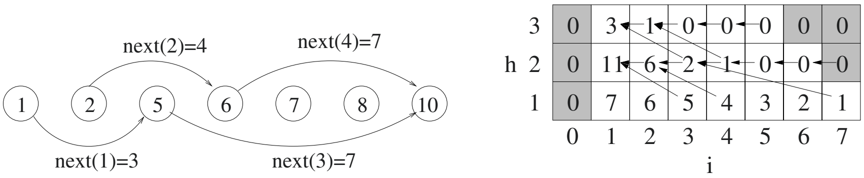
It is possible to use solve this problem using a dynamic programming style algorithm, which solves subproblems and stores their solutions in a table. If we have an ordered sequence of integers in F, a proper coloring can be seen as an ordered subsequence of integers that is itself composed of other subsequences. Each of these subsequences, in turn, can be extended, in principle, in a number of different ways to a sequence of length $r(v)$ by adding elements from F. The total number of possible extensions for each subsequence can be defined recursively. Once this is done, it is possible to choose one solution randomly.

More formally, let s be an ordered vector of integers in F, $L = |F|$, $D = t_v$ and $H = r(v)$. For each position i of s we have $next(i) = \min_j\{j : j > i,\ s[j] - s[i] \geq D\}$. For each subsequence l of s, the number $M_H[i]$ of proper subsequences of s of length $h \in \{1, \ldots, H\}$ containing $l = \{s[1], \ldots, s[i]\}$, can be computed recursively as

$$M_h[i] = \begin{cases} L - i + 1, & \text{if } h = 1 \\ M_{h-1}[next(i)] + M_h[i+1], & \text{if } L - i - 1 > h \geq 2 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

if $i \geq 1$ and $M_h[0] = 0$ by convention.

The total number of proper subsequences of length $r(v)$ is given by $M_H[1]$. For selecting one at random, one may scan the sequence s and select each element with probability $M_{h-1}[next(i)]/M_h[i]$, that is, the fraction of extensions



h \ i	0	1	2	3	4	5	6	7
3	0	3	1	0	0	0	0	0
2	0	11	6	2	1	0	0	0
1	0	7	6	5	4	3	2	1

Fig. 8. Given is an example for a vertex exact color reassignment where we have that $F = \{1, 2, 5, 6, 7, 8, 10\}$, $L = |F| = 7$, $D = t_v = 4$ and $H = r(v) = 3$. On the left is given the vector s of 7 integers. For each integer in the sequence the pointer $next()$ is computed; where it is not otherwise indicated, it is set to 0. On the right is given the table of $M_h[i]$ values. This table is filled starting from the low right corner; arrows indicate which already stored values are used to determine new table entries. The grey cells indicate cells that are not used in the computation and, hence, are assigned values by convention.

that contain the element in question. If an element is chosen, the scan continues from $next(i)$. Clearly, the sequence of numbers can be generated in linear time using the information provided by $M_h[i]$. However, computing the recursion (5) for all h and i takes exponential time. Fortunately, this can be done more efficiently by iteratively computing the values $M_h[i]$ at the beginning and recording them in a table. This iterative process is illustrated in Fig. 8, right: the table is completed going from bottom to top and from right to left within each row. Each new table entry requires the values of $M_{h-1}[next(i)]$ and $M_h[i + 1]$, which are already computed and stored. Hence, the $next$ function and the table can be computed in $\mathcal{O}(\max(D, H, \log L) \cdot L)$.

4.3 Experimental Results with SLS Algorithms

To assess the contribution of the new neighborhood, we consider its use within more complex SLS algorithms. As for the GCP, we based the new algorithm on a tabu search algorithm that uses in addition a standard one-exchange neighborhood (N_1). We compare the final algorithms with our reimplementation of the squeaky wheel algorithm that was shown to be effective for this problem [44].

With N_1 we use a standard Tabu Search procedure that chooses at each iteration a best non-tabu move or a tabu but "aspired" neighboring solution from the one-exchange neighborhood. The tabu list forbids to reverse a move and the tabu tenure is chosen as $tt = random(10) + 2\delta|V^c|$, where V^c is the set of vertices which are involved in at least one conflict, δ is a parameter, and $random(10)$ is an integer random number uniformly distributed in $[0, 10]$; this choice follows that of a successful tabu search algorithm for the graph coloring problem [26] and yields an algorithm analogous to that proposed in [22]. We denote this algorithm $\mathsf{TS}_{\mathsf{1\text{-}ex}}$.

When applying Tabu Search with N_R, a heuristic rule is used to reduce the effort of neighborhood exploration. At each iteration, first the best non-tabu move in N_1 is determined. If it improves on the current solution, it is accepted. If it leaves the evaluation function value unchanged or worsens it, a move is searched in N_R, restricted to vertices involved in at least one conflict. If a proper reassignment is found, it is applied; otherwise the best non-tabu move in N_1 is applied. In fact, the tabu search mechanism is applied only to moves in N_1 and it is in all equal to $\mathsf{TS}_{\mathsf{1\text{-}ex}}$. The randomized examination of N_R implements already an anti-cycling mechanism and this is the reason why preliminary experiments indicated the use of randomization preferable to that of determinism. We call the overall algorithm TS+R.

We also developed a variant of TS+R that considers a color reassignment for a randomly chosen vertex from V if no move is found in N_R restricted to conflicting vertices. This is motivated by the fact that a random reassignment of colors to vertices where no conflict is present may produce a change that can propagate profitably. We denote this variant TS+R*.

There are four sets of instances for benchmarks on this problem. We refer to [17] for a complete list of references and results on all classes of instances. Here, we restrict ourselves to presenting results on random uniform graphs of size $n = 60$. In such graphs, each of the $\binom{n}{2}$ possible edges is present with a probability $p = \{0.1, 0.5, 0.9\}$. Vertex requirements are chosen uniformly from the set $\{1, \ldots, r\}$ and vertex and edge separation distances are chosen uniformly from the set $\{1, \ldots, t\}$.

First we examine whether the reassignment neighborhood impacts the performance of tabu search. For this, we give in Table 4 the number of improvements due to a move in N_R in comparison to improvements in N_1. We consider 10 instances for different classes of uniform graphs and report the median values derived from one run of TS+R on each single instance. The algorithm was stopped after 10 000 iterations. The first column reports the number of iterations in which an improving solution was found in N_1 while the second column reports the number of improving solutions that was found in N_R. Note that this latter case occurs only if no possible improvement was found in N_1 while an improvement was still possible in N_R. Hence, the second column indicates a positive contribution in terms of number of improvements of the new neighborhood. The third column gives an indication of how many times the neighborhood N_R was searched in 10 000 iterations. We count a search for each attempted reassignment of colors to a single vertex. Finally, the last two columns report the values of $|F|$ and the span of integers in F in order to give an indication of the size of problems being solved.

We then compare the five SLS algorithms on several classes of instances. In [14], it was shown that the primary source of differences in the relative performance of the algorithms under study is the edge density of the graphs, while the combinations of values for r and t resulted to be less important (the study was limited however to values of r and t smaller or equal 10, and further investigation would be needed for larger values). Here, we fix $r = 10$ and $t = 5$ and consider graphs with edge density of $\{0.1, 0.5, 0.9\}$. In our experiments, we impose the same computation time limit for all algorithms. This is necessary

Table 4. Median statistics from 10 000 iterations of TS+R on 10 instances per class. The first two columns report the improvements found in N_1 and N_R, respectively. The third column reports the number of single-vertex searches in N_R. The last two columns report the number of colors in F, that is, the colors which are feasible according to the edge-distance constraints of vertices adjacent to v, and the range of colors in F, that is, $f_M - f_m$, with $f_M = \max\{c : c \in F\}$ and $f_m = \min\{c : c \in F\}$.

class	# improv. in N_1	# improv. in N_R	single-vertex searches in N_R	$\|F\|$	$f_M - f_m$
size=60, dens.=0.1, $r = 5, t = 5$	1179	59	17698	7	25
size=60, dens.=0.1, $r = 5, t = 10$	839	98	15953	8	50
size=60, dens.=0.1, $r = 10, t = 5$	6	0	20110	26	73
size=60, dens.=0.5, $r = 5, t = 5$	776	194	25463	6	43
size=60, dens.=0.5, $r = 5, t = 10$	926	113	22112	6	133
size=60, dens.=0.5, $r = 10, t = 5$	851	138	35321	9	69
size=60, dens.=0.9, $r = 5, t = 5$	651	117	21370	5	53
size=60, dens.=0.9, $r = 5, t = 10$	223	30	13782	8	159
size=60, dens.=0.9, $r = 10, t = 5$	640	77	23402	10	209

because the single iterations of $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ and TS+R have different computation time requirements. The time limit was chosen such that $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ could accomplish $I_{max} = 10^5 \times \sum_{v \in V} r(v)$ iterations. More details on the time limit are reported in [14]. Before running the experiments, each algorithm was fine-tuned using the F-race algorithm [10, 11], which is a fully automatic tuning procedure based on sequential testing. In the tabu search algorithms the only parameter to be determined is δ. For each of the algorithms we used numbers in $\{0.5; 1; 10; 20; 30; 40; 50; 60; 70; 100\}$ as candidates; the best value found was 10 for the random uniform instances. In what follows, each algorithm uses this value for δ.

We carried out the experiments and the analysis in a similar way as discussed in Sect. 3.4. That is, we run each algorithm once on each instance and ranked within the instance the result expressed in terms of color span. In Fig. 9 we report the analysis through simultaneous confidence intervals derived from the Friedman test [15, 20] at a 5 % protected level of significance. The more the interval is shifted towards the left the better the performance of the algorithm is. We made a different analysis for each of the three instance classes determined by edge density.

Results vary among the three classes of instances. On the low density instances the vertex color reassignment neighborhood does not introduce any significant improvement and, in fact, it may even worsen the basic $\mathsf{TS}_{\mathsf{1\text{-}ex}}$ slightly. However, as the edge density increases, using this new neighborhood becomes worthwhile and TS+R results to be a new state-of-the-art algorithm. Hence, TS+R is an example of an SLS algorithm where a very large-scale neighborhood gives a significant contribution towards enhancing performance.

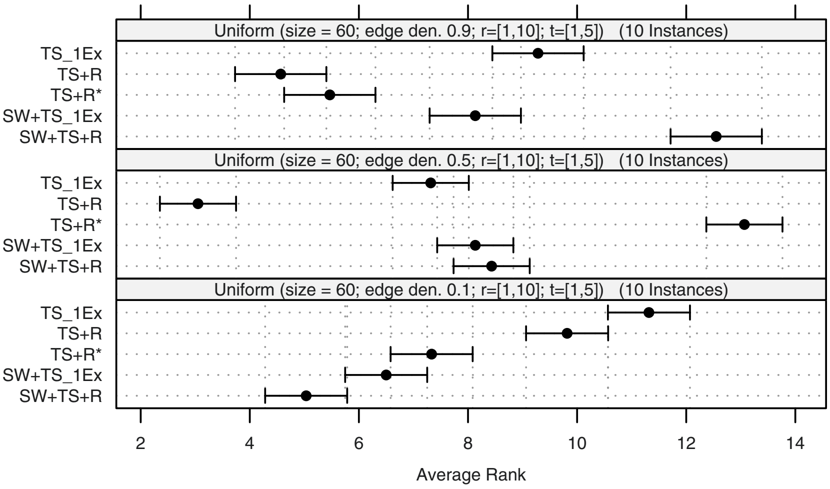


Fig. 9. The 95 % confidence intervals for the all-pairwise comparisons of SLS algorithms on aggregated uniform random instances of the GSTCP. The x-axis indicates the average rank while the confidence intervals are derived from the Friedman test

5 Conclusions

Very large-scale neighborhood searches have received a significant attention, especially in the last few years. Many different ways of defining and searching neighborhoods have been proposed and for various problems, very large-scale neighborhood searches are a central part of state-of-the-art SLS algorithms. Probably the best known example is the Lin-Kernighan heuristic for the traveling salesman problem. For several other problems, including generalized assignment [58], scheduling problems [33], or capacitated minimum spanning trees [5] excellent results have been reported. However, the design and implementation of very large-scale neighborhood searches requires significant insights into the particular problem being tackled, the usage of efficient data structures, and often substantial implementation time. In addition, despite significant efforts, for various problems rather negative results have been obtained in the sense that very large-scale neighborhood search algorithms failed to improve algorithmic performance. One such example is the graph coloring problem, which we also treated in this chapter, or the specific scheduling problem reported in [13]. Certainly, the list of negative examples may be longer – but unfortunately, often such negative results are not published or made available in some other form to the research community.

Given this overall picture, it seems that no clear prediction can be made as to when a very large-scale neighborhood search would result into a highly performing algorithm. This is in part due to missing guidelines as to how to use very large-scale neighborhood searches and the development of such algorithms is much left to the intuition of the researcher. Possibly, by some preliminary examination of the trade-off between computation time and solution quality improvement incurred by a very large-scale neighborhood some more directed choice may be done. Hence, the usage of very large-scale neighborhoods appears to be a promising possibility but there are still a large number of open research questions to be investigated.

Acknowledgments

Thomas Stützle acknowledges support from the Belgian F.R.S.–FNRS of which he is a research associate.

References

1. E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Chichester, UK, 1997.
2. R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3): 75–102, 2002.
3. R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. Very large-scale neighborhood search: Theory, algorithms, and applications. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 20-1–20-15. Chapman & Hall/CRC, Boca Raton, FL, USA, 2007.
4. R. K. Ahuja, J. B. Orlin, and D. Sharma. Very large-scale neighbourhood search. *International Transactions in Operational Research*, 7(4–5):301–317, 2000.
5. R. K. Ahuja, J. B. Orlin, and D. Sharma. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91(1):71–97, 2001. Series A.
6. M. Allen, G. Kumaran, and T. Liu. A combined algorithm for graph-coloring in register allocation. In D. S. Johnson, A. Mehrotra, and M. Trick, editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 100–111, Ithaca, New York, USA, 2002.
7. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.
8. C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.
9. N. Barnier and P. Brisset. Graph coloring for air traffic flow management. *Annals of Operation Research*, 130:163–178, 2004.
10. M. Birattari. The race package for R. Racing methods for the selection of the best. Technical Report TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2003.
11. M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2002.

12. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
13. T. Brueggemann and J. L. Hurink. Two very large-scale neighborhoods for single machine scheduling. *OR Spektrum*, 29:513–533, 2007.
14. M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems.* PhD thesis, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany, August 2005.
15. M. Chiarandini, D. Basso, and T. Stützle. Statistical methods for the comparison of stochastic optimizers. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. J. Gutjahr, R. F. Hartl, and M. Reimann, editors, *MIC2005: The Sixth Metaheuristics International Conference*, pages 189–196, Vienna, Austria, August 2005.
16. M. Chiarandini, I. Dumitrescu, and T. Stützle. Stochastic local search algorithms for the graph coloring problem. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 63-1–63-17. Chapman & Hall/CRC, Boca Raton, FL, USA, 2007.
17. M. Chiarandini, T. Stützle, and K. S. Larsen. Colour reassignment in tabu search for the graph set t-colouring problem. In F. Almeida, M. Blesa, C. Blum, J. M. Moreno, M. Pérez, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics*, volume 4030 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, Berlin, Germany, 2006.
18. R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimization.* PhD thesis, Southampton University, Faculty of Mathematical Studies, Southampton, UK, 2000.
19. R. K. Congram, C. N. Potts, and S. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.
20. W. J. Conover. *Practical Nonparametric Statistics.* John Wiley & Sons, New York, NY, USA, third edition, 1999.
21. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.
22. R. Dorne and J. K. Hao. Tabu search for graph coloring, T-colorings and set T-colorings. In S. Voß, S. Martello, I. H. Osman, and C. Roucairol, editors, *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer Academic Publishers, Boston, MA, USA, 1999.
23. I. Dumitrescu. *Constrained path and cycle problems.* PhD thesis, The University of Melbourne, Melbourne, Australia, 2002.
24. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
25. C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–464, 1996.
26. P. Galinier and J. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.
27. A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions of Vehicular Technology*, 35(1):8–14, 1986.
28. B. Gendron, A. Hertz, and P. St-Louis. On edge orienting methods for graph coloring. *Journal of Combinatorial Optimization*, 13(2):163–178, 2007.
29. C. A. Glass and Adam Prügel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, 56(3):324–330, 2005.

30. F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1–3):223–253, 1996.
31. F. Glover. Tabu search and adaptive memory programming – advances, applications and challenges. In R. S. Barr, R. V. Helgason, and J. L. Kennington, editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, pages 1–75. Kluwer Academic Publishers, Boston, MA, USA, 1996.
32. J. L. González-Velarde and M. Laguna. Tabu search with simple ejection chains for coloring graphs. *Annals of Operations Research*, 117(1-4):165–174, 2002.
33. A. Grosso, F. Della Croce, and R. Tadei. An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32(1):68–72, 2004.
34. G. Gutin and A. Yeo. Small diameter neighbourhood graphs for the traveling salesman problem: at most four moves from tour to tour. *Computers & OR*, 26(4):321–327, 1999.
35. W. K. Hale. Frequency assignment: Theory and applications. *Proceedings of the IEEE*, 68(12):1497–1514, 1980.
36. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.
37. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.
38. T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley & Sons, New York, NY, USA, 1994.
39. D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.
40. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, USA, 1972.
41. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technology Journal*, 49:213–219, 1970.
42. D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3 – Generating All Combinations and Partitions*. Addison Wesley, third edition, 2005.
43. F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.
44. A. Lim, Y. Zhu, Q. Lou, and B. Rodrigues. Heuristic methods for graph coloring problems. In *SAC'05: Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 933–939, New York, NY, USA, 2005. ACM Press.
45. S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.
46. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
47. P. Merz and B. Freisleben. Greedy and local search heuristics for the unconstrained binary quadratic programming problem. *Journal of Heuristics*, 8(2):197–213, 2002.

48. D. Neto. *Efficient Cluster Compensation for Lin-Kernighan Heuristics*. PhD thesis, University of Toronto, Department of Computer Science, Toronto, Canada, 1999.
49. C. N. Potts and S. van de Velde. Dynasearch: Iterative local improvement by dynamic programming; part I, the traveling salesman problem. Technical Report LPOM–9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995.
50. F. S. Roberts. T-colorings of graphs: Recent results and open problems. *Discrete Mathematics*, 93(2-3):229–245, 1991.
51. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. J. Maher and J.-F. Puget, editors, *Proceedings of Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer-Verlag, Berlin, Germany, 1998.
52. B. A. Tesman. Set T-colorings. *Congressus Numerantium*, 77:229–242, 1990.
53. P. M. Thompson and J. B. Orlin. The theory of cycle transfers. Working Paper No. OR 200-89, 1989.
54. P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
55. M. A. Trick and H. Yildiz. A large neighborhood search heuristic for graph coloring. In P. Van Hentenryck and L. Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 4510 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, Berlin, Germany, 2007.
56. G. J. Woeginger V. G. Deineko. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical Programming*, 87(3):519–542, 2000.
57. M. Yagiura and T. Ibaraki. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation. *Journal of Heuristics*, 7(5):423–442, 2001.
58. M. Yagiura, T. Ibaraki, and F. Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2):133–151, 2004.
59. A. Zymolka, A. M. C. A. Koster, and R. Wessäly. Transparent optical network design with sparse wavelength conversion. In *Proceedings of the 7th IFIP Working Conference on Optical Network Design & Modelling*, pages 61–80, Budapest, Hungary, 2003.

Hybrids of Constructive Metaheuristics and Constraint Programming: A Case Study with ACO

Bernd Meyer

FIT Centre for Research in Intelligent Systems
Monash University, Clayton, Australia
bernd.meyer@acm.org

Summary. A long-standing problem in combinatorial optimization with metaheuristics has been how to handle hard constraints effectively. Integrating metaheuristic methods with Constraint Programming (CP), an exact technique for solving hard constraints, promises a solution to this problem.

This chapter explores how such an integration can be achieved. We discuss possible types of couplings between the two algorithmic frameworks and define hybrid algorithms for each type. The central distinction is between tight coupling in which both components collaborate in an interleaved fashion and loose coupling where both components run in parallel, exchanging only (partial) solutions and bounds.

1 Introduction

Evolutionary metaheuristics are widely used for combinatorial optimization and achieve a competitive performance in many practically relevant applications. However, real-world problems are often subject to hard constraints and handling these with evolutionary metaheuristics is generally not easy. Constraint handling in evolutionary methods has thus been the subject of extensive research.

In this chapter we will discuss a new approach to handling hard constraints in metaheuristics by hybridizing these with Constraint Programming, an exact technique for solving hard constraints.

From a conceptual perspective our hybridization is important for two reasons: (1) It introduces declarative problem models into metaheuristics. This opens a way to fine-tune the function of the metaheuristics for a new problem domain in a systematic way. Such fine-tuning normally has to be performed in an ad-hoc fashion by programming problem-specific repair methods, new neighbourhood functions etc. The hybrid approach allows us to adapt some of this functionality automatically and systematically by re-defining the declarative problem model. (2) It introduces automatic learning of search strategies

B. Meyer: *Hybrids of Constructive Metaheuristics and Constraint Programming: A Case Study with ACO*, Studies in Computational Intelligence (SCI) **114**, 151–183 (2008)
www.springerlink.com

into CP. To cope with very large feasible spaces, the CP method requires us to design and implement problem-specific search strategies. In realistic cases, this task represents a significant development effort and requires considerable CP programming expertise. This has recently prompted leading CP researchers to declare ease-of-use as the next big challenge for CP systems [47, 56]. A hybridization of CP with metaheuristics is certainly not the "magic bullet" for the ease-of-use problem, but it can help to automate the problem of finding a search strategy to some degree.

1.1 Constraint Handling in Metaheuristics

Traditionally, three types of approaches have been used for constraint handling in metaheuristics: penalty-based relaxation techniques, multi-phase methods, and repair techniques [14]. Relaxation techniques allow the search to use infeasible solutions and simply modify the objective function by a penalty based on the amount of constraint violation. Multi-phase methods attempt to split the search into phases that try to find feasible subspaces and phases that try to optimize feasible solutions (Section 3.4 will discuss this in more detail). Repair techniques attempt to explicitly modify infeasible solutions to make them feasible.

Arguably, penalty-based relaxation is the most widely used way to handle constraints in Evolutionary Algorithms. For this the objective function $f(\mathbf{x})$ that is to be minimized is modified into a Lagrangian (an auxiliary objective function) $\hat{f}(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda} \cdot g(\mathbf{x})$ where $g(\mathbf{x})$ is a positive function measuring the amount of constraint violation and the Lagrange multipliers $\boldsymbol{\lambda}$ are weights for these violations. For example, in the simplest case of constraints $h_i(\mathbf{x}) = 0$ a natural choice for $g(\cdot)$ is $g(\mathbf{x}) = \sum_i \mid h_i(\mathbf{x}) \mid$.

The problem now becomes how to set the multipliers $\boldsymbol{\lambda}$ so that the optimal balance in the search between focusing on feasibility and emphasizing optimality is achieved. Many different schemas for static, dynamic, and adaptive penalties have been explored in the context of evolutionary computation [13], but unsurprisingly no single technique stands out that is best independently of the problem.

What makes matters more difficult is that any relaxation method can, in principle, not guarantee that solutions are fully feasible (as opposed to having only a minimal amount of constraint violation). Unfortunately, in practice, constraints are often hard and must be satisfied exactly.

Repair methods and other feasibility-maintaining methods, such as decoder representations or specialized cross-over and mutation operators [37], can be used to handle hard constraints.[1] However, their central disadvantage

[1] There are countless instances of repair method applications in the literature. As we will base our case study on ACO, we suggest that the interested reader may want to have a look at [52, 53] as an instructive example of ACO combined with a local search repair method for handling hard constraints in timetabling.

is that this requires the manual design of problem-specific algorithms for every new application. Similar considerations apply to multi-phase approaches.

An alternative to these approaches is to hybridize the stochastic metaheuristic with a method specialized for solving hard constraints, such as constraint propagation [17] or integer programming [41]. Such exact methods have complementary strengths to stochastic metaheuristic, and recent surveys have emphasized the importance of research into such hybridizations [7, 6, 26, 46]. The majority of work in this area, however, has focused on integer programming. In this chapter we focus on constraint programming (CP [33]) instead.

The advantage of a stochastic metaheuristic is its capability to automatically learn a promising search strategy for a large solution space. It is thus aimed at finding high quality solutions among an overwhelming number of feasible solutions. CP, on the other hand, is aimed at effectively reducing the search space by eliminating infeasible regions. It is thus at an advantage when finding any feasible solution is hard. In its generic form, the CP approach is at a disadvantage if the feasible space is very large. A hybrid approach holds the hope to combine these complementary advantages.

In this chapter we explore two different types of hybridization of a metaheuristic with constraint programming. We use Ant Colony Optimization (ACO [22]), a particular constructive model-based metaheuristic [7, 58], as the vehicle for this exploration. We discuss a loose coupling, in which both methods are used in parallel, and a tight coupling, where constraint propagation is interleaved with the solution construction phase of the metaheuristic.

We perform experimental evaluations of the two hybrid algorithms. Our example domain is machine scheduling with sequence-dependent setup times, a problem that in our experience is difficult to solve with penalty-based and multi-phase ACO techniques. The experiments show that the hybrid approach yields an improved performance for some problem types.

In Section 6 we summarize the general ideas of our framework and discuss some pertinent issues for the generalization of our approach to Estimation of Distribution Algorithms [39, 30, 32].

1.2 Basic Coupling of ACO and CP

Ant Colony Optimization [22] is a good candidate for our integration, because it is a constructive metaheuristics [7], i.e. a method that incrementally constructs solutions by iteratively adding new components to a partial solution. As we will see below, this incremental way of constructing solutions naturally corresponds to the CP approach and provides us with an immediate hook for an integration.

Ant Colony Optimization

Let us give a quick review of the ACO approach. In ACO a number of agents ("ants") independently construct solutions in parallel by iteratively augmenting partial solutions.

```
$\forall i,j : \tau_{i,j} = \tau_0$                          /* initialize pheromone matrix
$\forall i,j : \eta_{i,j} = d_{i,j}^{-1}$                      /* heuristic $\eta$=inverse distance
$l_{gb} := +\infty$; $T^{gb} := nil$                 /* initialize global best
for $t := 1$ to max_iterations do
    for $k := 1$ to number_of_ants do
        $T^k := nil$                       /* intialize tour of ant $k$ as empty
        mark all cities as unvisited by ant $k$
        $i :=$ random city not yet visited by ant $k$
        for $n := 2$ to number_of_cities do
            mark city $i$ as visited by ant $k$
            $C :=$ set of cities not yet visited by ant $k$
            choose next city $j$ to be visited by ant $k$ with probability
                $p_j := \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_{j \in C} \tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}$
            $T^k := append(T^k, (i,j))$
            $i := j$
        od
        $l_k := length(T^k)$
    od
    $ib := argmin_k(l_k)$                         /* best tour index
    if $l_{ib} < l_{gb}$ then begin $T^{gb} := T^{ib}$; $l_{gb} := l_{ib}$ end
    $\forall i,j : \tau_{i,j} := (1-\rho)\tau_{i,j} + \Delta\tau_{i,j}$           /* evaporate and reinforce
    where $\Delta\tau_{i,j} = \sum_{k=1}^{number\_of\_ants} \Delta\tau_{i,j}^k$ and
        $\Delta\tau_{i,j}^k = \begin{cases} Q \cdot l_k^{-1} & if\ (i,j) \in T^k \\ 0 & otherwise \end{cases}$
od.
```

Fig. 1. The Original Ant System Algorithm for TSP

Consider a TSP, where a partial solution corresponds to a partial path. Every construction step extends a partial path to a new city. In ACO the ants make the choice of the next city to be visited based on a so-called "pheromone value", which models the preference for a particular choice and is cooperatively learned by the ants during the search. In a TSP the pheromone value is commonly attached to a pair of cities c_i, c_j and models the preference to go from city c_i to c_j. The pheromone values are learned through a reinforcement strategy in which each agent, after the complete solutions have been constructed, reinforces the choices it has made during the solution construction with an amount of reinforcement that depends on the solution quality obtained. The original and most basic form of ACO is the Ant System (AS) algorithm. We present the AS algorithm in its common form for the TSP in Figure 1 before modifying it for machine scheduling in Section 3.2.

Constraint Programming

A very brief introduction to CP may be in place. For a comprehensive introduction the interested reader is referred to [33]. The core idea of CP is to give a programmer support for maintaining and handling relations (constraints) between variables. To this end, CP introduces a new kind of variable, called constraint variable, into the programming model. A constraint solver allows

```
Algorithm CP-basic
    setup domains for $x_1, \dots, x_n$
    post initial constraints
    $label([x_1, \dots, x_n])$
end.

procedure $label$(list $xs$)
    if $xs$=nil then return true
    else let $x = first(xs)$ in
        if not $bind(x)$ then return false
        else begin
            if $label(rest(xs))$ return true
            else begin
                $unbind(x)$
                return $label(xs)$
            end
        end
    end
end.
```

```
procedure $bind$(varname $x$)
    let $d = domain(x)$ in
        if $empty(d)$ then return false
        else begin
            $v := first(d)$
            $success := post(x = v)$
            if $success$ return true
            else begin
                $post(x \neq v)$
                return $bind(x)$
            end
        end
end.

procedure $unbind$(varname $x$)
    let $v$ = current_value($x$) in
        remove ($x = v$) from
           constraint store
        $post(x \neq v)$
    end
end.
```

Fig. 2. Basic CP Search Algorithm

variable values to be assigned or further constraints to be added ("posted"). There are many different forms of CP and the particular form that we use here is finite domain constraint programming. This name refers to the fact that the variable domains (i.e. the set of possible values that the variable can take) have to be finite. More specifically, we will only use integer variables.

The constraint solver analyses the restrictions on variables automatically "behind the scenes" and provides at least two services to the program: (1) It analyses whether a new constraint is compatible with the already existing ones and signals this as success or failure when it is posted. (2) It automatically reduces the domains of constraint variables according to the explicit and implicit restrictions. The program can query the solver for the domain of a variable and obtain the set of values that have not been ruled out for this variable.

An example will clarify this: Assume that X is an integer variable and that the constraint $0 \leq X < 3$ has been posted. We have $domain(X) = \{0, 1, 2\}$. Later we post $X < Y$ and $Y \leq 2$, which results in $domain(X) = \{0, 1\}$. If we now post the constraint $Y = 0$, the solver will signal that it is not consistent with the previous ones as it does not leave any possible value for X. If, instead, we post $Y = 1$, the solver will automatically infer that the only remaining possible value for X is 0 and bind it to this value. It is important to note that $v \in domain(X)$ does not guarantee that there is a value assignment for all the remaining constraint variables that satisfies all the problem constraints if $X = v$. This is because constraint solvers necessarily have to be incomplete as they are trying to solve NP-hard problems.

To solve a problem with finite domain CP we have to define a mathematical model of the problem that captures its structure using finite domain variables. After setting up the decision variables and their domains and posting any further problem constraints the solver will reduce the domains of all decision variables as much as possible. The second phase of a constraint program is the so-called labelling phase, which is essentially a search through the remaining space of domain values. In its simplest form, this search proceeds in individual labelling steps, each of which attempts to assign a particular decision variable a concrete value from its domain. A crucial question is, of course, in which order the variables are bound (the *variable ordering*) and in which order the corresponding domain values are tried (the *value ordering*). These two orderings significantly influence the efficiency of the search, and usually a substantial proportion of the programming effort has to be invested into finding more elaborate labelling strategies and into optimizing these. A high-level description of basic CP with a (trivial) lexicographical variable ordering is given in Figure 2.

Integrating CP and ACO

We can now analyse how ACO can be integrated with a constraint programming approach. Consider the basic generic constraint programming approach (Figure 2). In the context of this chapter, we consider the propagation method as fixed. The motivation for this is that in many constraint languages propagation solvers are treated as black boxes, so that the propagation cannot easily be manipulated. If we assume the propagation method to be fixed, the best way to couple the two methods is via the labelling procedure.

One way to tightly integrate ACO into CP is to use ACO as a mechanism to automatically learn an effective labelling strategy. Assuming as a simplification that the variable ordering is fixed, we can use ACO to learn a value ordering that is more likely to produce good solutions. As in the basic AS algorithm, the pheromone values $\tau_{i,j}$ can be used to learn the desired probabilities of selecting a given domain value for each decision variable. Instead of using a fixed value ordering, we make a probabilistic decision based on $\tau_{i,j}$ using the same probabilistic choice function as in the AS algorithm. Each ant independently attempts to construct a complete feasible solution. After all ants have completed their tours, the pheromone values are modified using the same evaporation and reinforcement used in ACO. The pseudo-code for the parts of the algorithm that differ from Figure 2 is given in Figure 3. Obviously, the same technique could be used to learn a variable ordering.

What we have described could be viewed as a “CP perspective” of the integration. As an alternative to this CP-centric perspective, we will in Section 4.3 take a “metaheuristic perspective” and consider ACO as the main component which we extend with constraint propagation. Essentially, we will use constraint propagation as a way of strengthening the construction phase of the metaheuristic with a powerful lookahead mechanism that utilizes a declarative problem model to avoid the construction of infeasible solutions. It will

```
Algorithm CP-with-ACO
    for each ant begin
        setup domains for $x_1, \ldots, x_n$
        post initial constraints
        if $label([x_1, \ldots, x_n])$
        then update global best solution
    end
    evaporate and reward solutions
end.
```

```
procedure $bind$(varname $x$)
    let $d = fd\_domain\_list(x)$ in
    if $empty(d)$ then return false
    else begin
        choose $v \in d$ probabilistically
        $success := post(x = v)$
        if $success$ return true
        else begin
            $post(x \neq v)$
            return $bind(x)$
        end
    end
end.
```

Fig. 3. CP Search with ACO labelling

become clear that these two ways of interpreting the coupling problem, while conceptually different, lead to virtually identical hybrid algorithms.

In the following we will fully define and analyse two types of coupling. Firstly, we will study an instance of a loose coupling, in which CP and ACO run in parallel, exchanging only information on bounds and candidate solutions. The second method is a tight coupling of the two approaches, in which the constraint propagation is integrated into the solution construction of ACO and guides which solution components are selected in each step. We will establish a baseline for comparison by testing basic ACO and basic constraint programming on our example problems (propagation plus backtracking as in Figure 2) using the model detailed in Section 4.1.

2 Problem Domain: Machine Scheduling with Sequence-dependent Setup Times

Our test domain is machine scheduling with sequence-dependent setup times. Machine scheduling is an important application area that arises in a variety of contexts [31, 1] and has been the subject of a significant amount of research by both the Operations Research and Artificial Intelligence communities. However, the results of this research have not always been useful in practice for a variety of reasons among them the over-simplification of the scheduling problems to make them tractable by the techniques applied [57].

Formally this problem can be defined as follows. Let J be a set of n jobs to be scheduled on a single machine. Each job has a job identification $i \in \{1, \ldots, n\}$, a processing time $\overline{duration}_i$, a release time $\overline{release}_i$ and a due date $\overline{due}_i$. In addition between any pair of jobs $i,\ j \in J$ there is a changeover time $\overline{setup}_{i,j}$. A feasible schedule assigns each job j a start time $start_j$ and finish time end_j such that $start_j \geq \overline{release}_j$, $end_j = start_j + \overline{duration}_j \leq \overline{due}_j$ and for any pair of jobs (i,j) where j succeeds i in the schedule $end_i + \overline{setup}_{i,j} \leq start_j$. The objective is to minimize the makespan $C_{\max} = \max_{j \in J} end_j$.

A number of related problems have been discussed in the literature. If $\forall\ j \in J : \overline{due_j} = \infty\ \wedge\ \overline{release_j} = 0$ then the minimum of the makespan is determined by minimizing the changeover times and hence this case is just the asymmetric travelling salesman problem (ATSP); see [21, 54] for the application of ACO to the ATSP. With release and due dates this problem is similar to the ATSP with time windows except that the latter only minimizes the duration of changeovers while the makespan also penalizes idle time that may need to be inserted in order to satisfy the release times.

Previous work on job scheduling includes pure CP approaches [45] and an attempt by the same authors to include CP in a tabu-search approach [44]. Improvements of constraint programming approaches have been discussed in [11, 40], and hybrids of two exact methods, CP and Integer Programming, in [27]. Alternatively Shin et al [51] consider this problem with relaxed due date constraints and the objective of minimizing the maximum lateness. Using the relaxed version of the problem makes it easy for Shin et al to apply a meta heuristic (in this case tabu search). ACO has been used before for job-scheduling with sequence-dependent setup times, but to the best of our knowledge only without hard constraints [18, 4]. A case study illustrating the practical importance of constraints for this problem is given in [28] where an ACO approach solves a single machine scheduling problem with sequence-dependent setup times in an aluminium casting centre. Constraints are handled via relaxation with penalties for violations of due dates and incomplete draining of the furnaces.

3 Constraint Handling in ACO

Our first goal is to test a loose coupling of CP and ACO. By “loose” we mean that both algorithms run in parallel and exchange only information on candidate solutions and bounds. In such a setting both components must produce feasible solutions, so it is imperative that the ACO component itself can handle constraints as effectively as possible on its own.

Unfortunately, handling hard constraints in ACO is far from trivial. This is, after all, one of the drivers for the research on hybrid algorithms. Essentially, the same basic methods that are used for other Evolutionary Algorithms are also used to accommodate hard constraints in ACO: penalty techniques, multiphase methods, and repair techniques. As an alternative to penalty-based ACO-methods we have earlier proposed to use stochastic ranking (SR [49]) in combination with ACO [35]. Stochastic Ranking appears to perform highly competitively [25] in the context of Evolution Strategies [50], and it lends itself to an integration into ACO. We adopt this method to obtain a suitable ACO base version for the loose coupling.[2]

[2] Some authors would consider this in itself already as a hybrid metaheuristic.

```
$\forall j \in \{1,\ldots,n\} : s_j := j$;
for $i$ :=1 to $N$ do
    for $j$ := 1 to $N$ do
        $u := random()$;
        if ($g(\mathbf{x}_{s_j}) = g(\mathbf{x}_{s_{j+1}}) = 0$ or $u < p_f$) then
            begin
                if ($f(\mathbf{x}_{s_j}) > f(\mathbf{x}_{s_{j+1}})$) then $swap(s_j, s_{j+1})$;
            end
        else
            if ($g(\mathbf{x}_{s_j}) > g(\mathbf{x}_{s_{j+1}})$) then $swap(s_j, s_{j+1})$;
    od
od.
```

Fig. 4. Stochastic Ranking Bubble Sort

3.1 Stochastic Ranking

SR replaces the Lagrangian objective measure $\hat{f}(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\lambda} \cdot g(\mathbf{x})$ with a ranking. The position in the ranking is taken as a surrogate solution quality and serves as a proxy for $\hat{f}(\mathbf{x})$. Switching to a ranking system allows SR to take into account the objective value $f(\mathbf{x})$ as well as the amount of constraint violation $g(\mathbf{x})$ without having to explicitly quantify penalty factors $\boldsymbol{\lambda}$.

This is best understood based on the concepts of over-penalization and under-penalization. Consider a ranking based on $\hat{f}(\mathbf{x})$ using an explicit penalty formulation. If $\boldsymbol{\lambda}$ is very large, in particular

$$\boldsymbol{\lambda} \cdot min_{\mathbf{x}}\ g(\mathbf{x}) > max_{\mathbf{x}}\ f(\mathbf{x}) - min_{\mathbf{x}}\ f(\mathbf{x})$$

we have an over-penalization. The consequence is that every infeasible solution (no matter how good its objective value) will always rank lower than any feasible solution (no matter how bad its objective value). In short, the ranking is dominated by the feasibility. In the extreme case this can amount to a death penalty. On the other hand, we have an under-penalization if

$$\boldsymbol{\lambda} \cdot max_{\mathbf{x}}\ g(\mathbf{x}) \ll max_{\mathbf{x}}\ f(\mathbf{x}) - min_{\mathbf{x}}\ f(\mathbf{x})$$

In this case, the ranking is clearly dominated by the objective function and in the extreme case $\boldsymbol{\lambda} \cdot max_{\mathbf{x}}\ g(\mathbf{x}) < min_{\mathbf{x},\mathbf{y}}\ (f(\mathbf{x}) - f(\mathbf{y}))$ we are ignoring feasibility.

Penalty adjustment is intended to balance this situation. SR dispenses with the penalty factors and instead explicitly adjusts the probability of an infeasible solution to be ranked more highly.

To compute such a ranking SR uses a stochastic bubble-sort procedure (Figure 4).[3] For every pairwise comparison of solutions a probabilistic decision

[3] Note that the stochastic ranking procedure given here is slightly modified from the original version in [49] which stops after the first sweep in which no swap occurred. This modification does not appear to influence the performance significantly and is made to simplify the analysis.

is made whether to use an over-penalization or an under-penalization. Clearly, two feasible solutions will always be compared according to objective value.

If $p_f = 0$, a comparison between between a feasible and an infeasible solution will always be won by the feasible solution as this has $g(\mathbf{x}) = 0$. Two infeasible solutions will always be compared based on their amount of constraint violation. This is clearly an over-penalization scenario. On the other hand, if p_f=1, we have under-penalization: All solutions will always be compared based on the objective value only, ignoring any constraint violation.

Setting p_f to intermediate values allows us to balance these two scenarios explicitly. When used in Evolution Strategies, SR thus gives us an explicit way to adjust the survival probability of infeasible solutions without having to have any *a priori* knowledge about the numerical range of objective and constraint violations.

Similarly, we can use SR in ACO to determine the amount of reward given to a solution. Let us look at the fundamental conceptual similarities between Evolution Strategies and ACO. In an Evolution Strategy the (potentially penalized) objective value of a solution determines its probability of survival into the next generation. Thus the ratio of fitness of two solutions implicitly determines their ratio in the next generation's population. Similarly, in ACO the quality of a solution determines the reward it receives. In the next iteration new solutions are generated by sampling the (implicit) probability distribution given by the pheromone values $[\tau_{i,j}]$. Thus, components of better solutions are more likely to re-appear in later samples, and the set of solutions generated in a later iteration (the equivalent of a later generation) will contain a higher proportion of fitter solutions. Both algorithm frameworks are on a fundamental level based on the idea of importance sampling. In fact, in [23] it was shown that a particular form of ACO is equivalent to an importance sampling-based optimization method (cross-entropy [48, 16]).

With this in mind, we transfer the idea of SR into ACO. As the base algorithm for integrating SR, we use Ant Colony System (ACS [19, 20, 21]), one of most successful and most commonly used refinements of the simple AS schema given in the introduction.

3.2 ACS

ACS introduces three modifications to the basic AS idea: (1) The selection of the next solution component is semi-greedy. With probability p the so-called *pseudo-random-proportional selection* in ACS chooses the next component exactly as in AS. However, with probability $(1 - p)$ the next component is chosen greedily as the one with the highest pheromone value. (2) ACS introduces an elitist concept: In each iteration only a single solution is considered for reinforcement. This is the globally best path found so far. (3) The normal global evaporation $\forall i, j : \tau_{i,j} := (1 - \rho)\tau_{i,j}$, which takes place after all ants have completed a tour construction, is complemented by local evaporation:

```
$\forall i, j : \tau_{i,j} := \tau_0$                       /* initialize pheromone matrix
$\forall i, j : \eta_{i,j} := setup\_time(i, j)^{-1}$       /* initialize heuristic function
$job_0 := 0$
$\forall j : \tau_{0,j} := \tau_0 \wedge \eta_{0,j} := 1$   /* virtual start job 0
$l_{gb} := +\infty$; $T^{gb} := nil$                        /* initialize global best
for $t := 1$ to max_iterations do
    for $k := 1$ to number_of_ants do
        $T^k := nil$                                        /* intialize tour of ant $k$ as empty
        mark all jobs as unscheduled by ant $k$
        for $n := 1$ to number_of_jobs do
            $C$ := set of jobs not yet scheduled by ant $k$
            $i := job_{n-1}$
            if $random() \leq p$ then choose next job $j \in C$ to be scheduled
                by ant $k$ with probability $p_j := \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\Sigma_{j \in C}\, \tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}$
            else choose $j = argmax_{j \in C}(\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta})$
            $T^k := append(T^k, (i, j))$
            $\tau_{i,j} := (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \tau_0$        /* local evaporation
            mark job $j$ as scheduled by ant $k$
            $job_n = j$
            $start_j := max(release_j, end_i + setup\_time(i, j))$
        od
        if $feasible(T^k)$ then $l_k := makespan(T^k)$
        else begin $l_k := \infty$; $T^k := nil$ end
    od
    $ib := argmin_k(l_k)$          /* best tour index
    if $l_{ib} < l_{gb}$ then begin $T^{gb} := T^{ib}$; $l_{gb} := l_{ib}$ end
    /* evaporate and reward
    $\forall (i, j) \in T^{gb} : \tau_{i,j} := (1 - \rho)\tau_{i,j} + Q \cdot l_{gb}^{-1}$
od.
```

Fig. 5. Basic Ant Colony System (ACS) for Machine Scheduling

Each ant slightly reduces the amount of pheromone associated with each decision as soon as it makes this decision. In consequence the same decision is less likely to be repeated by subsequent ants and thus solution diversity is increased. Local evaporation is restricted to an interpolation between the current pheromone value and an initialization value τ_0, so that each choice always has a non-negligible probability of being selected:

$$\tau_{i,j} := (1 - \rho) \cdot \tau_{i,j} + \rho \cdot t_0$$

A detailed description of ACS adapted to machine scheduling is given in Figure 5.

3.3 ACSsr – ACS with Stochastic Ranking

ACS, as an elitist method, can be interpreted as applying a ranking based on the objective value and rewarding only the first ranked solution. We can easily modify the algorithm to reinforce the first solution in the ranking generated by SR instead of using an objective-based ranking. We will call this algorithm ACSsr.

```
$\forall i, j : \tau_{i,j} := \tau_0$                              /* initialize pheromone matrix
$\forall i, j : \eta_{i,j} := setup\_time(i, j)^{-1}$              /* initialize heuristic function
$job_0 := 0$
$\forall j : \tau_{0,j} := \tau_0 \wedge \eta_{0,j} := 1$          /* virtual start job 0
$T^{fb} := nil$; $T^{ifb} := nil$                                  /* init feasible, infeasible best
for $t := 1$ to max_iterations do
    for $k := 1$ to number_of_ants do
        $T^k := nil$                                               /* intialize tour of ant $k$ as empty
        mark all jobs as unscheduled by ant $k$
        for $n := 1$ to number_of_jobs do
            $C :=$ set of jobs not yet scheduled by ant $k$
            $i := job_{n-1}$
            if $random() > p$ then choose next job $j \in C$ to be scheduled
                by ant $k$ with probability $p_j := \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_{j \in C} \tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}$
            else choose $j = argmax_{j \in C}(\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta})$
            $T^k := append(T^k, (i, j))$
            $\tau_{i,j} := (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \tau_0$          /* local evaporation
            mark job $j$ as scheduled by ant $k$
            $job_n = j$
            $start_j := max(release_j, end_i + setup\_time(i, j))$
        od
        if $feasible(T^k)$ then begin
            if $makespan(T^k) < makespan(T^{fb})$ then $T^{fb} := T^k$
        end else
            if $random() < p_f$ and $makespan(T^k) < makespan(T^{ifb})$
               or $tardiness(T^k) < tardiness(T^{ifb})$ then
                $T^{ifb} := T^k$
    od
    if $T^{fb} \neq nil$ then begin $x_1 := T^{fb}$; $x_2 := T^{ifb}$; $i = 2$ end
    else begin $x_1 := T^{ifb}$; $i = 1$ end
    $\forall j \in \{1, \ldots, number\_of\_ants\} : x_{i+j} := T^j$
    $sr\text{-}rank()$;   /* using $f(\cdot) = makespan(\cdot)$ and $g(\cdot) = tardiness(\cdot)$
    $T^{sgb} := x_{s_1}$                          /* stochastically best ranked tour
    /* evaporate and reward
    $\forall (i, j) \in T^{sgb} : \tau_{i,j} := (1 - \rho)\tau_{i,j} + Q \cdot makespan(T^{sgb})^{-1}$
od.
```

Fig. 6. ACSsr – Stochastic Ranking in ACO

More precisely, ACSsr keeps a memory of the best feasible solution as well as the best infeasible solution found so-far throughout the search. In each iteration the solutions found by all ants in this iteration are ranked *together* with the so-far best feasible and the so-far best infeasible according to the stochastic ranking algorithm. Only the highest ranking solution is rewarded. This integration, which emerged as promising from pilot studies, is fundamentally based on the same ideas that underpin stochastic ranking. The full algorithm for ACSsr is given in Figure 6.

As ACSsr only rewards the best solution, the full ranking is not required. Instead we only need to find the top ranking solution. This could be done more simply using a probabilistic comparison of each infeasible solution to the best feasible solution according to makespan. Let $p_b(k, s)$ be the probability of an infeasible solution with the best makespan “bubbling” in s sweeps to the top of the ranking through k solutions that are better according to the over-penalized ranking:

$$p_b(k,s) = \begin{cases} 0 & \textit{if } k > s = 0 \\ 1 & \textit{if } k = s = 0 \\ p_f^{(k+1)} \cdot p_b(0, s-1) + & \\ \sum_{i=1}^{k+1} \left(p_f^{k+1-i} \cdot (1-p_f) \cdot p_b(i, s-1) \right) & \textit{otherwise} \end{cases}$$

To bubble up k positions the solution needs to "survive" $(k+1)$ comparisons (one "from below" to stay in place and a further k to climb). Note that the otherwise case accounts for all possibilities including the case where an element looses the first comparison, sinks down one position, and subsequently has to climb $k+1$ positions.

We could obtain an *approximation* of the original SR behavior by comparing each infeasible solution with probability

$$\frac{\sum_{i=1}^{n+m} \; p_b\left(\frac{i}{n+m}\left(n + \lceil \frac{m}{2} \rceil\right), s\right)}{n+m}$$

to the feasible solution with the best makespan, where n is the number of feasible solutions and m the number of infeasible solutions in the pool.

Pilot studies [29] have shown, however, that such an approximation truly differs form the original SR schema and does not achieve the same performance when used in ACSsr. While the behaviour of SR is not yet fully understood, it appears that this difference is mainly due to situations where all solutions generated are infeasible.

3.4 ACSsr Benchmarking

We perform benchmarks of ACSsr for various problem sizes and within those groups for problems of varying tightness (Table 1). The test data sets were drawn from two different sources.[4] In all cases the initial letters indicate the problem type. This is followed by the number of jobs and further letters or numbers identifying the particular data set. The first source of data sets is based on an application in wine bottling in the Australian wine industry. Setup times are taken from actual change-over times at a bottling plant. There are no release times as the wine is always available in the tanks. Due dates are selected randomly from one of seven possible spreads over the planning period (representing, for example, the end of each day over a week). Processing times are generated in a slightly more complicated manner to create problems with various degrees of tightness. Let μ be the average processing time available per job (calculated by removing n times the average setup time from the last due date). The duration for each job is chosen randomly in the interval $[\frac{1}{2\beta}\mu, \frac{3}{2\beta}\mu]$

[4] All test data are available from the author by request.

Table 1. Constraint ACS Benchmarks

Problem	ACSsr, pf=0.7					Multiphase ACS			
	Best	Avg	StdDev	Failed	p	Best	Avg	StdDev	Failed
W8.1	8321	8321	0	0 %	n/a	8321	8321	0	0 %
W8.2	5818	5818	0	0 %	n/a	5818	5818	0	0 %
W8.3	4245	4245	0	0 %	n/a	4245	4245	0	0 %
W20.1	**8504**	**8526**	23.23	0 %	7.8e-4	8564	8611	55.74	0 %
W20.2	**5062**	**5110**	26.16	0 %	0.017	5102	5136	18.92	0 %
W20.3	**4332**	**4369**	24.39	0 %	0.001	4372	4408	22.95	0 %
W30.1	**8067**	8536	418.77	**0** %	n/a	8232	**8310**	54.85	60 %
W30.2	**4612**	**4703**	83.36	0 %	1.3e-4	4787	4865	60.15	0 %
W30.3	4148	4304	86.37	0 %	0.289	4283	4341	60.42	0 %
RBG10.a	3840	3840	0	**0** %	n/a	3840	3840	0	10 %
RBG16.a	2596	2596	0	**0** %	n/a	2596	2596	0	40 %
RBG16.b	2094	2094	0	**0** %	0.107	2094	2115	28.76	30 %
RBG21.9	4522	4550	19.00	0 %	0.110	4546	4565	21.84	0 %
RBG27.a.3	1725	1738	9.48	0 %	0.302	1715	1733	11.48	0 %
RBG27.a.15	**1479**	**1495**	12.15	**0** %	n/a	1507	n/a	n/a	90 %
RBG27.a.27	n/a	n/a	n/a	100 %	n/a	n/a	n/a	n/a	100 %
BR17.a.3	1525	1553	23.99	0 %	0.709	1527	1549	23.25	0 %
BR17.a.10	1383	1420	20.75	**0** %	0.237	1400	1431	16.33	30 %
BR17.a.17	1057	1057	0	**0** %	n/a	1057	1057	0	70 %

for $\beta = 1.2, 2$ or 3 for problem class 1, 2 or 3. Hence problem W8.1 is the tightest problem involving eight jobs, while W8.3 is the loosest problem.

The remaining data sets are taken from the ATSP-TW literature [2] based on an application involving scheduling a stacker crane in a warehouse. In these data sets only some of the jobs have time window constraints (both release and due dates). The number of jobs with non-trivial time windows is indicated by the last set of digits in the problem name. Thus in RBG27.a.27 each of the 27 jobs has a time window, while RBG27.a.3 is identical except that the time window constraints for all but three of the jobs have been removed.

Our comparator is a multi-phase ACS that emerged as the best basic ACS from pilot experiments and outperformed all static penalty versions tried [36]. Basic penalty ACS failed to find feasible solutions for most of the medium to highly constrained problems for sizes $n \geq 16$ when reinforcement was purely based on the objective measure (makespan) and shortest setup time (SST) was used to define the heuristics $\eta_{i,j}$. SST is effective for minimizing the makespan, but is not a good guide for finding feasible solutions. Here, earliest due date (EDD) is much more effective. Therefore, the ACS version used here works in two phases. EDD ($\eta_{i,j} = due_j^{-1}$) is used and in the first phase the reward is inverse proportional to the degree of constraint violation (tardiness: $\Sigma_i max(0, end_i - due_i)$). This forces the search to first construct feasible solutions regardless of their quality. Once feasible solutions are found, we switch to the second phase which only reinforces feasible solutions with an amount of reinforcement proportional to the inverse of the makespan.

Table 1 shows averages and best over 10 runs, (ACS/ACSsr with 10 ants, 5000 iterations, $Q, \alpha, \beta = 1, \rho = 0.05, p = 0.5, \tau_0 = Q/(20 \cdot N \cdot \rho \cdot \hat{l})$, where N is the number of jobs and $\hat{l}$ is an estimate on the average makespan, i.e. the sum

Table 2. Bootstrap Test (100 runs, 95 % Confidence Intervals)

Problem	ACSsr		ACS		
	Avg	Failed	Avg	Failed	p
W20.1	**[8524, 8527]**	0 %	[8611, 8617]	0 %	[7e-4, 1e-3]
W20.2	**[5108, 5111]**	0 %	[5136, 5138]	0 %	[0.02, 0.05]
W20.3	**[4367, 4369]**	0 %	[4405, 4408]	0 %	[0.005-0.02]
W30.1	[8497, 8550]	**0 %**	**[8309, 8318]**	60 %	n/a
W30.2	**[4699, 4709]**	0 %	[4861, 4868]	0 %	[2e-4, 1e-3]
RBG10.a	[3840, 3840]	**0 %**	[3840, 3840]	10 %	n/a
RBG16.a	[2596, 2596]	**0 %**	[2596, 2596]	40 %	n/a
RBG16.b	**[2094, 2094]**	**0 %**	[2113, 2117]	30 %	n/a
RBG27.a.15	**[1494, 1495]**	**0 %**	[1507, 1507]	90 %	n/a
BR17.a.10	**[1418, 1421]**	**0 %**	[1430, 1432]	30 %	n/a
BR17.a.17	[1057, 1057]	**0 %**	[1057, 1057]	70 %	n/a

of processing times plus $(N-1)$ times the average setup time). A parametric search for optimal values of ρ, p, τ_0 shows that the algorithm is robust against parameter changes within reasonable limits, so that these standard values from the literature can be used for comparison. "Failed" gives the percentage of runs for ACS and ACSsr that did not produce any feasible solutions. We also give the p-values of a two-sided t-test (unequal variance) for the null-hypothesis that the ACSsr results and the ACS results have the same mean, i.e. our confidence that the results for ACSsr and ACS are truly different with statistical significance. p-values are not given where the ACS failure rate is too high for them to be meaningful.

We can ask whether the number of runs used in these experiments is sufficient to firmly establish significance. Simply increasing the number of runs would, of course, still leave us in the dark about the reliability of our tests. To obtain an answer to this question, we selectively perform a 100 run bootstrap test for each case in which Table 1 indicates a significant difference. The nonparametric bootstrap allows us to establish confidence intervals for our test statistics without making any assumption about the underlying distribution of the data [24]. The results are summarized in Table 2. They reconfirm the observations made in Table 1.

Best results are marked in bold face in Table 1 and in Table 2. Three different groups of problems emerge from the test data sets. The first group consists of small problems (W8.1-W8.3), which both algorithms easily solve to optimality. The second group consists of larger problems (W20.1-W30.2). In these cases ACSsr exhibits a substantially improved performance, generally finding better solutions than ACS. Arguably the most interesting class is made up of the more highly constrained real world problems (RBG10.a–BR17.a.17). In most of these cases the multiphase ACS is not able to reliably generate solutions, exhibiting failure rates of up to 90 %. ACSsr significantly improves on this performance, reducing the failure rates to 0% in all cases with the exception of the most highly constrained problem, which is also of significant size (RBG27.a.27). This clearly demonstrates the superior constraint solving ability of ACSsr over ACS. We will thus only use ACSsr for further comparison and for integration with CP in the loose coupling.

The performance of ACSsr is weakly sensitive to the setting of pf. This is not surprising and has also been observed in studies of the original SR approach [25]. In ACSsr the sensitivity is limited and ACSsr generally outperforms the two-phase ACS for all $pf > 0.5$. The optimum setting for pf was found by conducting all experiments for pf values from 0 to 1 in steps of 0.1 and from 0.65 to 0.75 in steps of 0.01. As expected the performance for $pf = 0$ is the same as for the basic ACS. The best overall performance (measured as the greatest average performance gain across all test problems) was found at $pf = 0.7$. These are the results reported above. The optimal pf value found deserves a second look. At a first glance $pf = 0.7$ appears very high, particularly considering that in the original SR technique very low pf values are used. The reason why ACSsr requires higher pf values is that we use an elitist reward: Unless a solution "bubbles" to the top of the ranking, the stochastic change in the ranking will not effect any difference. Thus, for a solution to be rewarded it needs to win a larger number of comparisons than in the original SR approach. In fact, values of $0.7 \leq pf \leq 0.90$ perform almost on par for most test problems.

Finally, we note that both ACO versions might perform better with other pheromone models [8]. We would, however, expect their relative performance to remain approximately stable as such a change should impact on both algorithms in similar ways.

4 Loose Coupling Versus Tight Coupling

To compare the ACSsr results obtained in the previous section with the basic CP solution, it remains to fully define the constraint model.

4.1 Constraint Model

Our problem specification contains the following data: n tasks, labelled $1 \dots n$ have to be sequenced in n sequence positions. Each task is identified by a task number $i \in \{1 \dots n\}$, has a release time $\overline{release_i}$ and a due time $\overline{due_i}$ as well as a fixed duration $\overline{duration_i}$. We also have sequence-dependent setup times $\overline{setup_{i,j}}$ specified for all pairs of tasks.

The most direct way to build a constraint model for a scheduling problem is to exploit the high-level scheduling constraints that modern constraint programming languages provide. This gives access to efficient global propagation methods specialized for scheduling. We use Sicstus Prolog CLP(FD) [10] for our reference implementation, which like most other CP languages provides a global constraint *serialized* which implements scheduling directly. It models the problem via decision variables for start time and end time of each task and is initialized with a complete problem specification containing release dates, due dates, durations and setup times. We use $\widetilde{start}_i$ and $\widetilde{end}_i$ to represent the start and end times of the task with id i. From our perspective, the global constraint *serialized* has to be treated as a black-box with $\widetilde{start}_i$ and $\widetilde{end}_i$ as its interface.

While this provides a very basic functioning CP model, it is clearly unsuitable for an ACO-like approach, which must construct candidate solutions by sequentially extending partial solutions. As our primary goal is to integrate the CP solution with the ACO approach, we must define a model that is also suitable for ACO. The basic choice is whether to assign jobs to sequence positions (in sequential order) or whether to assign sequence positions to jobs (in some order of jobs). This corresponds to either associating decision variables with sequence positions (such that the i-th variable indicates the task number to be executed in the i-th sequence position), or to associating decision variables with task numbers (such that the i-th variable indicates the position of task i in the sequence). The alternative of using successor variables is not explored here but used in e.g. [45].

In-line with the pure ACO method introduced above we use the first approach. This is a sensible choice because we are essentially optimizing for sequence-dependent set-up times. We use variables job_n where $job_n = i$ if the task with id number i is executed as the n-th job in sequence. We constrain:

$$\forall i : job_i \in \{1, \ldots, n\} \quad \text{and} \quad \mathit{all\_different}(job_i)$$

Three further constraint variables per task model the temporal aspects: $start_n$ is the scheduled start time, end_n the end time, and $setup_n$ the setup time for the n-th task in the sequence.

We now have introduced two different complementary models, which obviously introduces redundancy and increases the problem size. However, it is necessary to do so to make the model suitable for an ACO-like approach and to use the global propagation mechanism accessible through the serialized constraint at the same time. The gain in propagation effectiveness through the global constraint makes up for increased model size. Of course, the two models have to be coupled for cross-propagation:

$$\forall n, j : job_n = j \Rightarrow start_n = \widetilde{start}_j \wedge end_n = \widetilde{end}_j$$
$$\forall n, j : \widetilde{start}_j < start_n \vee \widetilde{start}_j > start_n \Rightarrow job_n \neq j$$

For technical reasons we also use a set of auxiliary variables: $duration_n$, $release_n$, due_n, $setup_n$ to capture the respective times for the n-th task in the sequence (Note that this is different from $\overline{duration}_j$ etc. which give the respective times for the task with id j). These are coupled to the data and each other via:

$$\begin{aligned}
\forall n &: end_n = start_n + duration_n \\
\forall n &: start_n \geq release_n \wedge end_n \leq due_n \\
\forall n > 1 &: start_n \geq end_{n-1} + setup_n \\
\forall n > 1 &: setup_n \in \{\overline{setup}_{i,j} \mid i,j \in \{1 \ldots n\}\} \\
\forall n &: duration_n \in \{\overline{duration}_j \mid j = 1 \ldots n\} \\
\forall n &: release_n \in \{\overline{release}_j \mid j = 1 \ldots n\} \\
\forall n &: due_n \in \{\overline{due}_j \mid j = 1 \ldots n\}
\end{aligned}$$

We use reified constraints [33] to bind the auxiliary variables to the data as soon as the id of the n-th job in sequence is known. Note that these can only propagate once the pre-condition is fulfilled:

$$\begin{aligned}
&\forall i > 1, l, m : job_{i-1} = l \wedge job_i = m \Rightarrow setup_i = \overline{setup}_{l,m}. \\
&\forall n, j : job_n = j \Rightarrow duration_n = \overline{duration}_j \wedge release_n = \overline{release}_j \wedge \\
&\qquad due_n = \overline{due}_j \wedge start_n = min(\overline{release}_j, end_{n-1} + setup_n). \\
&\forall i, j : end_i > \overline{due}_j \Rightarrow job_i \neq j.
\end{aligned}$$

As we always assign the earliest possible start time to a scheduled task, it is clear that a valuation of job_n together with the data completely determines all other variables. Evidently this CP model corresponds to a typical ACO approach if we label job_n in the order of increasing n.

Table 3 shows the results for CP with this model limited to 1,000,000 labelling steps and repeats the ACSsr results from above for comparison.[5] The column “Steps” indicates the number of labelling steps until the best solution was found. Column “Finished” gives the number of labelling steps after which the complete CP search has exhausted the search space (if less than 1,000,000).

As can be seen from Table 3 the performance of the basic CP model is (unsurprisingly) not impressive. We can clearly distinguish three groups of results: (1) Small problems (W8.1–W8.3, RBG10.a) are solved to optimality by both approaches. (2) Some of the larger problems (RBG27.a.27, BR17.a.10) are sufficiently highly constrained for CP to solve them to optimality, while ACSsr does not solve them to optimality (or, in the instance of RBG 27.a.27, does not find any feasible solutions). This is a clear indication of the influence of search space pruning through propagation. Finally, (3) ACSsr yields the better solutions for all of the more loosely constrained problems of larger size (W20.1–W30.3, RBG27.a.3, RBG27.a.15). This is a clear indication of the influence of the search heuristics that ACSsr learns during the search.

4.2 Loose Coupling

The possibility to combine the strengths of both methods in a unified approach is obvious. The simplest possible combination is a very loose coupling

[5] This setting results in a comparable number of labelling steps in CP and ACSsr.

Table 3. Loose Coupling Benchmarks

Problem	CP			ACSsr, pf=0.7			CP&ACSsr			
	Best	Step	Finished	Best	Avg	Failed	Best	Avg	Step	Finished
W8.1	8321	870	13,555	8321	8321	0 %	8321	8321	**344**	**5,229**
W8.2	5818	6,667	15,707	5818	5818	0 %	5818	5818	**2,598**	**5,616**
W8.3	4245	36,237	66,975	4245	4245	0 %	4245	4245	**13,358**	**25,227**
W20.1	8779	15,239	n/a	**8504**	**8526**	0 %	**8504**	8530	n/a	n/a
W20.2	5747	16,110	n/a	**5062**	**5110**	0 %	5102	5120	n/a	n/a
W20.3	4842	433,794	n/a	**4332**	4369	0 %	4342	**4367**	n/a	n/a
W30.1	8817	201,693	n/a	**8067**	8536	0 %	8052	**8469**	n/a	n/a
W30.2	5457	999,877	n/a	4612	4703	0 %	**4582**	**4680**	n/a	n/a
W30.3	5129	547,046	n/a	**4148**	**4304**	0 %	4268	4334	n/a	n/a
RBG10.a	3840	10	9,191	3840	3840	0 %	3840	3840	n/a	**3**
RBG16.a	2596	16	52	2596	2596	0 %	2596	2596	16	52
RBG16.b	2120	8,807	n/a	**2094**	**2094**	0 %	**2094**	**2094**	n/a	n/a
RBG21.9	6524	346,879	n/a	4522	4550	0 %	**4511**	**4543**	n/a	n/a
RBG27.a.3	1984	147,616	n/a	1725	**1738**	0 %	**1695**	1725	n/a	n/a
RBG27.a.15	1569	16,878	n/a	1479	**1495**	0 %	**1459**	1496	n/a	n/a
RBG27.a.27	**1076**	27	113	n/a	n/a	100 %	**1076**	**1076**	27	113
BR17.a.3	1663	552,611	n/a	**1525**	1553	0 %	1529	**1548**	n/a	n/a
BR17.a.10	**1031**	650,120	n/a	1383	1420	0 %	**1031**	**1031**	**28,427**	**138,707**
BR17.a.17	1057	17	95	1057	1057	0 %	1057	1057	17	95

that works in the following way: Both algorithms run in parallel exchanging information on candidate solutions and objective bounds in both directions. The rationale of this coupling is clear: CP solutions can be used by ACSsr to better focus the search on promising regions and if a better solution is found by ACSsr in these regions, CP pruning can be improved by a tighter objective bound. In addition to being handled by CP in the usual way, every feasible solution that CP generates is sent to a queue that connects CP with ACSsr. When ACSsr finishes one iteration for all ants, the solutions in the queue are read, the queue is cleared and the solutions are treated by ACSsr like normal ant-generated solutions. They are ranked together with all ant generated solutions and have the same chance of being reinforced. In the opposite direction, every feasible solution generated by ACSsr that improves on the best solution found so-far is sent to CP and is used within CP as an objective bound in the same way as CP-generated solutions:

$$\forall n : end_n < l_{gb} \wedge \widetilde{end}_n < l_{gb}$$

where l_{gb} is the makespan of the new solution.

The experimental results for the loose coupling are given in Table 3 in the column "CP&ACSsr". The column "Step" for "CP&ACSsr" gives the number of labelling steps in the CP component needed for generating the best solution, in the cases where the best solution is generated by the CP component. As expected, the combined method exhibits the best overall performance. This is unsurprising, as it basically runs the previous two algorithms in parallel. Somewhat disappointingly, the combined method does not seem to leverage much from the coupling. In the cases where CP previously gave the superior solutions, it is also the dominant component in CP&ACSsr and the latter shows the same solution quality as CP. Conversely, for problems where previously ACSsr was superior, this is the dominant component in CP&ACSsr and both exhibit the same performance. It is important to note that a two-sided *t*-test (unequal variance) clearly indicates that there is no statistically significant difference in the result distributions obtained from ACSsr and CP&ACSsr in the cases where ACSsr is the dominant component.

However, the table also shows that there is some (marginal) advantage to the coupling in the form of a minor speed-up in cases where ACSsr is able to provide a reasonably good objective bound early on in the search. This is visible in the case of the smaller problems (W8.1-W8.3, RBG10.a) and for BR17.a.10 where the coupling achieves a substantial speed-up, making it possible to finish an exhaustive search in fewer steps than needed by CP alone to find the best solution and thus ascertaining optimality of this solution. In all of these cases the runs exhibit the same characteristics: The first few solutions of reasonably good quality are found by ACSsr and the bounds supplied in this way drive the CP component. It is rare for ACSsr to improve on CP-generated solutions, which suggests that ACSsr does not sufficiently re-focus the search region when receiving an improved solution from CP.

4.3 Tight Coupling

The results of the loose coupling indicate that simply rewarding candidate solutions supplied by CP is not sufficient to improve the overall performance of the ACO approach. A tighter integration in which the CP component directly interacts with the solution construction phase of ACO may provide a better approach to the problem. The basic idea is simple. Consider the structure of the pure ACO approach, which constructs the solution step by step, in our case by selecting the next job to be executed. This is exactly what the labelling phase in the basic CP solution does. The advantage of CP over the basic ACO approach is that candidate jobs that cannot lead to an improved solution are ruled out by the propagation component and are not considered in the solution construction. In contrast, the standard ACO construction may select an inferior component and only discover that this does not lead to a good solution after the construction of the whole solution has been finished. The CP approach can thus supply additional information to the construction phase, which could be interpreted as a “look-ahead”. On the other hand, the CP approach does not learn during the search (except via objective bounds), while ACO attempts to learn about good regions in the search space via reinforcement. Both approaches can be combined by using constraint propagation during the candidate selection.

The tight ACO-CP integration that we introduce is, in a sense, most closely related to a combination of a partial repair technique with death penalty: The CP component assists ACO in the construction of feasible solutions. The consequence is that a significantly larger number of feasible solutions will be generated (the repair aspect). However, CP does not guarantee that a feasible solution is generated as the propagation is necessarily incomplete: A construction attempt may stop in a dead end when the current partial solution cannot be completed into a feasible solution. In this case it will simply not be rewarded (the death-penalty aspect).

We use the basic Ant Colony System as the point of departure. Note that ACSsr, despite its improved performance, would be unsuitable, because it is based on including infeasible solutions in the search.

Consider ACS for Single Machine Job Scheduling as described in Figure 5. The most obvious way to integrate CP is during the candidate selection in line 11 where it can prune the candidate list by domain reduction of C. The basic idea is for C to become a constraint variable so that the list of candidates is simply $domain(C)$. This requires a separate C_i^k for each decision step i and each ant k. When a candidate j is chosen, $C_i^k = j$ is posted to the solver, which can then propagate to reduce the domains of the remaining C_i^ks. This effectively results in lookahead as to which candidate selections will not be able to be completed into a feasible solution.

Of course, the remaining parts of the constraint model also need to be copied, so that each ant can maintain a separate version of the current state of solution construction. This amounts to the following full model in which the superscript k indexes the decision variables for the k-th ant:

- Basic Constraint

$$\forall k, i : job_i^k \in \{1, \ldots, n\} \quad \text{and} \quad \forall k : all\_different(job_i^k)$$

- Auxiliary Variables

$$\begin{aligned}
\forall k, n &: end_n^k = start_n^k + duration_n^k \\
\forall k, n &: start_n^k \geq release_n^k \wedge end_n^k \leq due_n^k \\
\forall n > 1, k &: start_n^k \geq end_{n-1}^k + setup_n^k \\
\forall n > 1, k &: setup_n^k \in \{\overline{setup}_{i,j} \mid i, j \in \{1 \ldots n\}\} \\
\forall k, n &: duration_n^k \in \{\overline{duration}_j \mid j = 1 \ldots n\} \\
\forall k, n &: release_n^k \in \{\overline{release}_j \mid j = 1 \ldots n\} \\
\forall k, n &: due_n^k \in \{\overline{due}_j \mid j = 1 \ldots n\}
\end{aligned}$$

- Data Bindings

$$\begin{aligned}
&\forall i > 1, k, l, m : job_{i-1}^k = l \wedge job_i^k = m \Rightarrow setup_i^k = \overline{setup}_{l,m}. \\
&\forall k, n, j : job_n^k = j \Rightarrow duration_n^k = \overline{duration}_j \wedge release_n^k = \overline{release}_j \wedge \\
&\qquad\qquad due_n^k = \overline{due}_j \wedge start_n^k = min(\overline{release}_j, end_{n-1}^k + setup_n^k). \\
&\forall i, j, k : end_i^k > \overline{due}_j \Rightarrow job_i^k \neq j.
\end{aligned}$$

- Cross-propagation

$$\forall k, n, j : job_n^k = j \Rightarrow start_n^k = \widetilde{start_j^k} \wedge end_n^k = \widetilde{end_j^k}$$
$$\forall k, n, j : \widetilde{start_j^k} < start_n^k \vee \widetilde{start_j^k} > start_n^k \Rightarrow job_n^k \neq j$$

- Objective Bound
$$\forall k, n : end_n^k < l_{gb} \wedge \widetilde{end_n}^k < l_{gb}$$

A more complex way to integrate CP would be to re-order the preferences for the candidate selection through the heuristic bias factor $\eta_{i,j}$. Additional information from the constraint solver, such as a bound on the objective, can be used to dynamically adjust $\eta_{i,j}$. Here, we will focus on the first approach and modify the candidate list only.

4.4 CPACS

Depending on the model and the solver this propagation may not be strong enough. With the model and the propagation algorithms detailed above, the fact that a particular assignment $C_i^k = j$ cannot lead to a feasible solution is sometimes only discovered once $C_i^k = j$ is posted. Consider a problem consisting of J_1 with duration 20 and time window $[0, 100]$, J_2 with duration 10 and time window $[0, 25]$, and some further jobs. Even though a human observer can immediately see that J_2 has to be scheduled before J_1, the propagation solver may discover this only after $C_i^k = J_1$ is posted (the domain of J_2's start time is now empty). The same effect could obviously occur through a much longer chain of dependencies. Using the ACO-CP coupling outlined above the fact that the tour cannot be completed would be discovered as soon as the constraint $C_i^k = J_1$ is posted. However, the simple coupling would not exploit this, as the ant would just be terminated. Effectively, nothing would be learned from this failed attempt, and no information would be retained for subsequent tour construction attempts.

To better exploit the propagation, we modify the coupling by using *single level backtracking* for the selection of the next candidate. The ant makes the usual probabilistic selection which job j to assign to C_i^k based on the reduced domain of C_i^k and posts the constraint $C_i^k = j$. If the solver signals failure the ant backtracks one step and posts $C_i^k \neq j$ instead. This removes j from the domain of C_i^k and potentially triggers other propagation steps. The ant then tries to bind C_i^k to another value from its further reduced domain. This process continues until either one of the assignments is accepted by the propagation solver or until $domain(C_i^k) = \emptyset$. In the first case the tour construction proceeds, while it fails in the second case. The complete algorithm for this coupling, which we term CPACS, is given in Figure 7.

Of course, the generate-and-test step emulated by single level backtracking simply strengthens the constraint propagation. If we would not consider the

```
initialize solver; post initial constraints; $s_0$ := solver_state();
$\forall i, j : \tau_{i,j} := \tau_0$                          /* initialize pheromone matrix
$\forall i, j : \eta_{i,j} := setup\_time(i, j)^{-1}$          /* initialize heuristic function
$job_0 := 0$
$\forall j : \tau_{0,j} := \tau_0 \wedge \eta_{0,j} := 1$          /* virtual start job 0
$l_{gb} := +\infty$; $T^{gb} := nil$;    /* initialize global best
for $t$ := 1 to max_iterations do
    restore initial solver state $s_0$
    for $k$ := 1 to number_of_ants do
        $T^k := nil$           /* intialize tour of ant $k$ as empty
        mark all jobs as unscheduled by ant $k$
        $n := 0$; $feasible := true$;
        while $n <$ number_of_jobs and $feasible$ do begin
            $n := n + 1$
            $i := job^k_{n-1}$
            do
                $C := domain(job^k_n)$
                if $random() > p$ then choose next job $j \in C$ to be sched. by ant $k$ with
                    probability $p_j := \frac{\tau^{\alpha}_{i,j} \cdot \eta^{\beta}_{i,j}}{\sum_{j \in C} \tau^{\alpha}_{i,j} \cdot \eta^{\beta}_{i,j}}$
                else choose $j = argmax_{j \in C}(\tau^{\alpha}_{i,j} \cdot \eta^{\beta}_{i,j})$
                $feasible := post(job^k_n = j) \wedge$
                        $post((start_j := max(release_j, end_i + setup\_time(i, j)))$
                if not($feasible$) then $post(job^k_n \neq j)$
            until $feasible \vee C = \emptyset$
            if $feasible$ then begin
                $T^k := append(T^k, (i, j))$
                $\tau_{i,j} := (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \tau_0$     /* local evaporation
                mark job $j$ as scheduled by ant $k$
            end
        end
        if $feasible(T^k)$ then $l_k := length(T^k)$
        else begin $l_k := \infty$; $T^k := nil$ end
    end
    $ib := argmin_k(l_k)$          /* best tour index
    if $l_{ib} < l_{gb}$ then begin $T^{gb} := T^{ib}$; $l_{gb} := l_{ib}$ end
    $\forall (i, j) \in T^{gb} : \tau_{i,j} := (1 - \rho)\tau_{i,j} + Q \cdot l^{-1}_{gb}$   /* evaporate and reward
end.
```

Fig. 7. Hybrid Constraint Propagation + Ant Colony System (CPACS)

propagation method as fixed, it could (and probably should) to the same effect be directly embedded into the constraint solver instead of into CPACS.

We compare the performance of CPACS with the loosely coupled version in Table 4. To provide context we also repeat the results of ACSsr. Recall that the performance differences between ACSsr and CP&ACSsr are not statistically significant except for in the tightly constrained cases where the results are obtained by the CP component (RBG27.a.27, BR17.a.10). The same three groups of results emerge: For small problems (W8.1-W8.3) all algorithms exhibit the same performance. For some larger problems that are only loosely constrained (W20.1, W20.3) CP&ACSsr exhibits a marginally better overall performance.

The most interesting group of problems are the more tightly constrained problems of intermediate to larger sizes where CPACS shows a significantly improved performance over the other algorithms (RBG21.9 to RBG27.a.15,

Table 4. Tight Coupling Benchmarks

Problem	ACSsr, *pf*=0.7				CP&ACSsr				CPACS			
	Best	Avg	StdDev	Failed	Best	Avg	StdDev	Failed	Best	Avg	StdDev	Failed
W8.1	8321	8321	**0**	0 %	8321	8321	**0**	0 %	8321	8322	2.10	0 %
W8.2	5818	5818	**0**	0 %	5818	5818	**0**	0 %	5818	5819	2.41	0 %
W8.3	4245	4245	**0**	0 %	4245	4245	**0**	0 %	4245	4246	1.58	0 %
W20.1	**8504**	**8526**	23.23	0 %	**8504**	8530	**22.95**	0 %	8564	8604	38.72	0 %
W20.2	**5062**	**5110**	26.17	0 %	5102	5120	**17.49**	0 %	**5062**	5117	26.03	0 %
W20.3	**4332**	4369	24.39	0 %	4342	**4367**	**15.37**	0 %	4352	4385	21.99	0 %
W30.1	8067	8536	418.77	**0** %	**8052**	8469	297.46	**0** %	8142	**8182**	**28.28**	80 %
W30.2	4612	4703	83.36	0 %	**4582**	**4680**	82.60	0 %	4695	4743	**40.74**	0 %
W30.3	**4148**	**4304**	86.37	0 %	4268	4334	55.37	0 %	4288	4337	**41.99**	0 %
RBG10.a	3840	3840	0	0 %	3840	3840	0	0 %	3840	3840	0	0 %
RBG16.a	2596	2596	0	0 %	2596	2596	0	0 %	2596	2596	0	0 %
RBG16.b	2094	2094	0	0 %	2094	2094	0	0 %	2094	2094	0	0 %
RBG21.9	4522	4550	19.00	0 %	4511	4543	19.40	0 %	**4481**	**4489**	**7.84**	0 %
RBG27.a.3	1725	1738	**9.48**	0 %	1695	1725	17.16	0 %	**927**	**940**	12.79	0 %
RBG27.a.15	1479	1495	12.15	0 %	1459	1496	16.98	0 %	**1068**	**1068**	**0**	0 %
RBG27.a.27	n/a	n/a	n/a	100 %	**1076**	**1076**	**0**	**0** %	**1076**	**1076**	**0**	**0** %
BR17.a.3	1525	1553	23.99	0 %	1529	1548	16.08	0 %	**1003**	**1003**	**0**	0 %
BR17.a.10	1383	1420	20.75	0 %	**1031**	**1031**	**0**	0 %	**1031**	**1031**	**0**	0 %
BR17.a.17	1057	1057	0	0 %	1057	1057	0	0 %	1057	1057	0	0 %

BR17.a.3). This is a clear indication of having learned a successful labelling strategy.

Table 5. Bootstrap Test (100 runs, 95 % Confidence Intervals)

Problem	CP&ACSsr		CPACS		
	Avg	Failed	Avg	Failed	*p*
W20.1	**[8528, 8532]**	0 %	[8600, 8605]	0 %	[1e-4, 3e-4]
W20.2	[5116, 5118]	0 %	[5114, 5117]	0 %	[0.37, 0.48]
W20.3	[4364, 4365]	0 %	[4384, 4387]	0 %	[0.039, 0.074]
W30.1	[8402, 8436]	**0 %**	**[8161, 8164]**	80 %	n/a
W30.2	[4674, 4684]	0 %	[4742, 4747]	0 %	[0.07, 0.12]
W30.3	[4331, 4337]	0 %	[4336, 4341]	0 %	[0.46, 0.57]
RBG21.9	[4540, 4542]	0 %	**[4488, 4489]**	0 %	[1e-5, 2e-5]
RBG27.a.3	[1725, 1727]	0 %	**[937, 938]**	0 %	[8e-19, 4e-18]
RBG27.a.15	[1494, 1496]	0 %	**[1068, 1068]**	0 %	[8e-14, 2e-13]
RBG27.a.27	[1076, 1076]	0 %	[1076, 1076]	0 %	[1.0, 1.0]
BR17.a.3	[1550, 1551]	0 %	**[1003, 1003]**	0 %	[2e-17, 3e-17]

As before we back up the test cases that we have isolated as interesting in Table 4 with a 100 run bootstrap test. Recall that no significant performance differences were found between CP&ACSsr and ACSsr/CP, respectively, so we restrict the comparison to CP&ACSsr. The results, which are summarized in Table 5, confirm our observations.

5 Discussion

Our expectation for the hybrid approach to show its strength mainly for problems of intermediate tightness is confirmed by our experiments.

The loose coupling shows only minor performance improvements over the two pure algorithms. In the few cases where the loosely coupled hybrid outperforms both basic algorithms, the improvements are in runtime but not in solution quality: The hybrid algorithm finds the solutions faster, and in some cases finishes the exhaustive search earlier. These runs have a particular characteristic in our experiments: ACO finds reasonably good solutions early in the search, providing the CP component with good objective bounds that enable it to prune more efficiently.

The tight coupling, in contrast, outperforms the other algorithms for larger problems that are reasonably tightly constrained, taking advantage of the more effective solution construction made possible by the CP component.

The empirical observation that CPACS is more effective in some more tightly constrained cases is in agreement with theoretical considerations. The introduction of the CP component into the construction phase can be interpreted as a partial look-ahead or as equivalent to an (incomplete) repair method. As a consequence of this, the proportion of feasible solutions that is sampled is significantly higher for tightly constrained problems. This, of course, already speeds up the algorithm simply on the basis of a higher sampling rate. Importantly, it also has a significant influence on the convergence dynamic of the underlying ACO. From the theoretical analysis of the convergence dynamics of (Hypercube) AS it is known that the expected value of the solution quality is strictly increasing, if all ants construct a feasible solution in every attempt (as is the case for unconstrained problems) [5]. This is not the case when ants can get stuck in infeasible solutions caused by hard constraints, such as time windows or incomplete networks for TSPs. In such cases, the expected value can decrease indicating that the algorithm converges on inferior solutions [34]. Generating fewer infeasible (rejected) samples makes this situation less likely to occur and thus gives a higher probability of converging on the optimum. This argument lends additional theoretical support to the suggestion that CPACS should have a performance superior to ACS.

It is important to keep the limitations of the tight coupling in mind. Firstly, CPACS is not a complete method any more: It does not perform an exhaustive search. Thus, if we need to know whether a solution is optimal, the loose coupling is the only possibility. Secondly, in the case of larger problems that are only loosely constrained the performance advantage of CPACS vanishes, as CP cannot contribute significantly to the solution construction any more. In these cases the loose coupling can be at a runtime advantage. This is because it can rely on the performance of the ACSsr component, which does not have to perform any propagation and can thus sample significantly faster.

The main factor that slows CPACS in its basic form is that it does not utilize trailing [33] for efficient propagation. We expect a significant speed-up from extensions of the algorithm that partly exploit trailing. These are currently under investigation.

It is important to acknowledge that for any specific and sufficiently narrow class of problems we could probably fare better with a highly customized search algorithm (at the expense of having to design and implement this). Here we are, however, concerned with a generic way of problem solving using a declarative problem model, and thus with an approach that is easily and safely applied to new problem domains.

6 Estimation of Distribution Algorithms and CP

We have discussed the coupling of constructive metaheuristics and constraint programming using ACO as an example. Our integration gives us a way to fine-tune the function of the metaheuristics in a systematic way based on a declarative problem model. This generalizes and subsumes the concepts of repair techniques and feasibility-maintaining solution construction. It is important to note that this particular hybridization is only one example of a much broader class of algorithms. Many other metaheuristics can benefit from the integration with CP in a similar way. The primary criteria that render a metaheuristic, such as ACO, suitable for our type of integration are that it is (a) constructive and (b) model-based.

We call a metaheuristic *constructive* if it builds candidate solutions incrementally by adding solution components to a partial solution until the candidate solution is complete. This obviously corresponds well to how the labelling phase in CP proceeds. Any other constructive metaheuristic can in principle be integrated with CP in the same way, such that the propagation solver provides an extended lookahead to increase the efficiency of the search.

Non-constructive metaheuristics are (typically) based on local search. A local search algorithm starts from some initial candidate solution and repeatedly tries to replace this solution by a better one chosen from an appropriately defined neighbourhood of the current solution [7]. Simulated Annealing is a prototypical example of a local search algorithm. Whether or not a local-search metaheuristic is suitable for our kind of CP integration depends on how exactly the neighbourhood move to a new candidate solution is performed. A metaheuristic that performs this move by incrementally exchanging components of the current candidate solution could almost be regarded as constructive. It may be able to benefit from an integration with CP in the same way as a purely constructive metaheuristic, by using the propagation solver to restrict possible neighbourhood moves. A local search that simply “jumps” from one complete candidate solution to another complete candidate solution in a non-constructive fashion is, however, not suitable for our framework, because it only requires feasibility *checking* and cannot effectively exploit an extended lookahead.

The second reason to select ACO was its model-based nature or, more precisely, its model-building characteristics. *Model-based* refers to the fact that the selection of solution components in the construction phase is based on an explicit problem model. In ACO the problem model is captured in the pheromone matrix. It is a probabilistic model and represents the likelihood for each solution component to belong to a good solution. Our tight coupling effectively extends the ACO problem model by placing additional constraints on its decision variables. Being model-based and being constructive obviously goes hand in hand.

Model-building refers to the fact that ACO attempts to improve its problem model during the search by a reinforcement learning mechanism. ACO achieves this by estimating (the equivalent of) a conditional probability distribution during the search. The model does not just represent an independent probability for each solution component to belong to a good solution, but also captures dependencies (linkages) between different parts of a solution (i.e. between decision variables). In an ACO-style model the dependencies the algorithm can learn are predetermined by the way reinforcement is given, i.e. by the interpretation of the pheromone matrix. For example, in our implementation of the JSP, $\tau_{i,j}$ stands for the preference to execute task j immediately after task i, and so these are the only dependencies the algorithm can learn.

In ACO the model is not directly expressed in the language of conditional probabilities, and it is thus questionable whether ACO provides the best framework to understand probabilistic problem models.[6]

Luckily, a different class of metaheuristics, called Estimation of Distribution Algorithms (EDAs [39, 30, 32]), provides us with the right tools and is directly framed in terms of conditional probability models.[7]

EDAs are a close relative of genetic algorithms.[8] Like these, they are based on a sampling and elimination process. In contrast to these, they dispense with the explicit population representation and do not manipulate individual population members directly. Instead, they represent a whole population implicitly via joint probability distributions, making the desirability of solution components and their linkages explicit.[9]

ACO is also closely related to some forms of EDAs [15, 58], specifically to the univariate marginal distribution algorithm (UMDA) [39, 38] and to

[6] In fairness to ACO it has to be said that it was not designed for this purpose.

[7] Similar frameworks are also known as Iterated Density Estimation Algorithms (IDEAs [9]) and Probabilistic Model Building Genetic Algorithms (PMBGAs [43]).

[8] Note that GAs themselves are neither model-based nor constructive and thus generally not directly suitable for the integration that we have proposed. This, however, does not limit our approach as EDAs allow us to generalize GAs and are suitable for the integration.

[9] In a sense EDAs are essentially GAs stripped from their biological inspiration and recast into a probabilistic modelling framework. From the probabilistic modelling perspective they could, somewhat flippantly, be regarded as “sanitized GAs”.

population-based incremental learning (PBIL) [3]. The main difference here is that the approximation of probabilities by pheromone levels is replaced by explicit probability distributions.

In this sense, EDAs allow us to generalize both GAs and ACO and to explicitly recast these metaheuristics in the language of probability theory. This is exactly what we need to generalize our integration to more complex problem models. The general EDA procedure is outlined in Figure 8. Any EDA that performs the sampling step in line 7 in a constructive fashion is also suitable for the framework that we have presented.

```
(1)   $t := 0$;
(2)   generate initial population $P_0$;
(3)   $best$ := best solution in $P_0$;
(4)   while (termination condition not reached) {
(5)       select subset of "good" solutions $S_t \subset P_t$;
(6)       construct probability model $M_t$ for $S_t$;
(7)       sample $M_t$ generating population $O_t$ of new solutions;
(8)       $P_{t+1} := merge(P_t, O_t)$;
(9)       $t := t + 1$;
(10)      $tmp$ := best solution in $P_t$;
(11)      if $f(tmp) < f(best)$ then $best := tmp$;
(12)  }
```

Fig. 8. Generic Estimation of Distribution Algorithm

The structure of the model M, regardless of whether it is framed in terms of probabilities or pheromones, determines the types of dependencies the algorithm can learn. This is obviously a crucial factor for the performance of any model-based search algorithm, an effect that has been demonstrated empirically, for example in [8] for ACO.

In ACO and ACO-like EDAs the structure of the model is predetermined by the implementation. While we can decide to capture any dependency that is deemed important *a priori*, the model structure cannot change during the search. Ideally, we would like our algorithm to be able not only to learn the probability distributions in the model, but also to refine the structure of the model, i.e. to learn new types of dependencies that are only discovered during the search.

Learning an appropriate model structure is exactly the core idea of the Bayesian Optimization Algorithm (BOA [42]), a specific form of EDA. BOA models the population using Bayesian networks and learns both the structure of an appropriate network as well as its conditional probability tables during the search. We thus believe that BOA is a particularly promising second candidate for the type of hybridization we have suggested.

In our hybrids we maintain two different but coupled problem models: On the EDA side we are learning a probabilistic model during the search, whereas on the CP side we have a crisp relational model.[10] The core question for an effective EDA-CP hybrid is how these two models are best coupled.

On one hand, dependencies between two variables in the CP model, such as $X \leq 2Y + 5$, can be extracted by a static analysis of the model, and can be used to initialize an adequate structure for the probability model (here by introducing a link between X and Y). Further such dependencies can be found dynamically during the search by the propagation mechanism. On the other hand, not all constraints should be used like this. The simplest example are constraints that clearly express functional dependencies, such as $X = 2Y + 5$. Modelling these in the probability model would only introduce redundancy and incur additional computational costs.

Information can also flow in the opposite direction. We could term this "stochastic constraint discovery". For example, if the probability of $p(X = n \mid Y = n)$ is found to be very close to zero in the EDA model, we may consider to post $X \neq Y$ dynamically as a constraint to make this information available to the propagation solver and thus to further strengthen the lookahead.[11]

Due to the different nature of the Bayesian model and the constraint model, it is far from trivial to decide what the best way is to exchange information between these. We are currently investigating both static aspects (model analysis and refinement before the search) as well as dynamic aspects (information exchange during the search) for BOA-CP hybrids.

Another important question concerns the trade-off between model-building and sampling. Learning a good Bayesian network is computationally expensive[12] and only worthwhile if it provides significant guidance to the sampling process. Directing the search for a good network structure by an analysis of the constraint network, may help to improve this trade-off in the hybrid algorithms.

We are confident that the extension of our approach to the richer class of probabilistic models used in EDAs in general and specifically in BOA will lend significantly more power to this framework.

7 Conclusions

We have presented and evaluated two ways of hybridizing constructive metaheuristics with Constraint Programming, using Ant Colony Optimization as our example. Our loose coupling is based on exchanging candidate solutions and objective bounds between ACO and CP. The tight coupling is based on

[10] In our basic coupling the CP model appears to be static. However, this is not necessarily the case if we consider the learning of nogoods etc.

[11] It has to be taken into account that this would turn the probabilistic information on $X \neq Y$ into crisp information reducing $p(X = n \mid Y = n)$ to exactly zero.

[12] For most metrics, learning an optimal Bayesian network for a given sample is $\mathcal{NP}$-hard [12]. However, polynomial-time approximations often work well in practice.

interleaving the constraint propagation with the solution construction phase of the metaheuristic component.

The main rationale for constructing these and other hybrids of exact methods with stochastic metaheuristics is to combine the complementary advantages of these two frameworks for combinatorial optimization problems. While CP is aimed at problems that are highly constrained so that the propagation component can prune the search space very effectively, stochastic metaheuristics, such as ACO, are aimed at problems for which the feasible space is very large so that the search in feasible regions is the dominant component influencing the performance. The expectation for a hybridization is that it shows its main performance advantage in the intermediate range between these two extremes. The empirical evaluation of our case study bears this out.

Our hybridization framework introduces declarative problem models into metaheuristics and the automatic learning of search strategies into CP. We consider both as important steps in bringing these two methodologies closer together, to make them easier to use, and to render them more readily applicable to new problem domains. We hope that such hybridizations will ultimately provide the basis for the inclusion of metaheuristic toolboxes into high-level modelling languages, such as OPL [55], in order to equip these with improved automatic search strategies.

References

1. A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239, 1999.
2. N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001.
3. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Int. Conf. Machine Learning (ML-95)*, 1995.
4. A. Bauer, B. Bullnheimer, R. F. Hartl, and C. Strauss. An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the Congress on Evolutionary Computation*, Washington/DC, July 1999.
5. C. Blum, November 2003. Personal Communication.
6. C. Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373, 2005.
7. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
8. C. Blum and M. Sampels. When model bias is stronger than selection pressure. In *Parallel Problem Solving From Nature (PPSN-VII)*, Granada, September 2002.
9. P. A. Bosman and D. Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In *Genetic and Evolutionary Computation Conference - GECCO*, pages 197–200, Las Vegas, July 2000.
10. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Proc. PLILP'97 Programming Languages: Implementations, Logics, and Programs*, Southampton, September 1997.

11. Y. Caseau and F. Laburthe. Improved CLP scheduling with task intervals. In *International Conference on Logic Programming*, Santa Margherita Ligure, Italy, June 1994.
12. D. M. Chickering, D. Geiger, and D. Heckerman. Learning bayesian networks is NP-hard. Technical report, Microsoft Research, Redmont, WA, 1994. MSR-TR-94-17.
13. C. Coello and A. Carlos. A survey of constraint handling techniques used with evolutionary algorithms. Technical report, Laboratorio Nacional de Informtica Avanzada, 1999. Technical Report Lania-RI-9904.
14. C. A. Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, 2002.
15. O. Cordón, F. Herrera, and T. Stützle. A review on the ant colony optimization metaheuristic: Basis, models and new trends. *Mathware and Soft Computing*, 9(2–3):141–175, 2002.
16. P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.
17. R. Dechter. *Constraint Processing.* Morgan Kaufmann Publishers, San Francisco, CA, 2003.
18. M. den Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In *Parallel Problem Solving from Nature - PPSN VI*, Paris, France, September 2000.
19. M. Dorigo, G. D. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.
20. M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. Technical Report TR/IRIDIA/1996-5, Universite Libre de Bruxelles, 1996.
21. M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
22. M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, 2004.
23. M. Dorigo, M. Zlocin, N. Meuleau, and M. Birattari. Updating ACO pheromones using stochastic gradient ascent and cross-entropy methods. In *Proceedings of the Evo Workshops*, Kinsale, Ireland, April 2002.
24. B. Efron. *The Jackknife, the bootstrap and other resampling plans.* SIAM, 1982.
25. R. Farmani and J. A. Wright. Self-adaptive fitness formulation for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 7(5):445—455, 2003.
26. F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In F. Glover and G. Kochenberger, editors, *Handbook of metaheuristics.* Kluwer, Boston/MA, 2003.
27. F. Focacci, A. Lodi, and M. Milano. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing*, 14(4):403–417, 2003.
28. M. Gravel, W. L. Price, and C. Gagné. Scheduling continuous casting of aluminum using a multiple objective ant colony optimization metaheuristic. *European Journal of Operational Research*, 143(1):218–229, 2002.
29. M. Held. Analysis and improvement of constraint handling in ant colony algorithms, November 2005. BCS Honours Thesis, Monash University.

30. P. Larrañaga and J. A. Lozano (eds.). *Estimation of distribution algorithms: a new tool for evolutionary computation*. Kluwer, Boston, 2002.
31. 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*, pages 445–522. North Holland, Amsterdam, Netherlands, 1993.
32. J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea (eds.). *Towards a New Evolutionary Computation*. Springer-Verlag, 2006.
33. K. Marriott and P. Stuckey. *Programming With Constraints*. MIT Press, Cambridge, MA, 1998.
34. B. Meyer. On the convergence behaviour of ant colony search. In *Asia-Pacific Conference on Complex Systems*, Cairns, December 2004.
35. B. Meyer. Constraint handling and stochastic ranking in ACO. In *IEEE CEC – Congress on Evolutionary Computation*, Edinburgh, September 2005.
36. B. Meyer and A. Ernst. Integrating ACO and constraint propagation. In *Ant Colony Optimization and Swarm Intelligence (ANTS 2004)*, Brussels, September 2004.
37. Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, Berlin, 2000.
38. H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5:303–346, 1998.
39. H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. binary parameters. In *Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, Berlin, September 1996.
40. W. Nuijten and C. Le Pape. Constraint-based job scheduling with ILOG scheduler. *Journal of Heuristics*, 3:271–286, 1998.
41. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover Publications Inc., Mineola, NY, 2nd edition, 1998.
42. M. Pelikan. *Hierarchial Bayesian Optimization Algorithm*. Springer-Verlag, Berlin, 2005.
43. M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.
44. G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5(3):255–279, 1999.
45. G. Pesant, M. Gendreau, J.-Y. Potvinand, and J.-M. Rousseau. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32(1):12–29, 1998.
46. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira and J. R. Alvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*. Springer-Verlag, 2005.
47. J.-F. Puget. Constraint programming next challenge: Simplicity of use. In *Principles and Practice of Constraint Programming—CP'04*, Toronto, September 2004.
48. R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, Berlin, 2004.

49. T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284—294, 2000.
50. H.-P. Schwefel. *Evolution and Optimum Seeking.* Wiley, New York, 1995.
51. H. J. Shin, C.-O. Kim, and S. S. Kim. A tabu search algorithm for single machine scheduling with release times, due dates, and sequence-dependent set-up times. *International Journal of Advanced Manufacturing Technology*, 19(11):859–866, 2002.
52. K. Socha, J. Knowels, and M. Sampels. A MAX-MIN ant system for the university course timetabling problem. In *International Workshop on Ant Algorithms (ANTS 2002)*, Brussels, September 2002.
53. K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003)*, April 2003.
54. T. Stützle and H. H. Hoos. MAX-MIN ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.
55. P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, 1999.
56. P. Van Hentenryck and L. Michel. Synthesis of constraint-based local search algorithms from high-level models. In *AAAI-07*, Vancouver, July 2007.
57. V. C. S. Wiers. A review of the applicability of OR and AI scheduling techniques in practice. *Omega*, 25(2), 1997.
58. M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131:373–395, 2004.

Hybrid Metaheuristics for Packing Problems

Toshihide Ibaraki[1], Shinji Imahori[2] and Mutsunori Yagiura[3]

[1] School of Science and Technology
Kwansei Gakuin University, Sanda, Japan
ibaraki@kwansei.ac.jp
[2] Graduate School of Information Science and Technology
University of Tokyo, Tokyo, Japan
imahori@mist.i.u-tokyo.ac.jp
[3] Graduate School of Information Science
Nagoya University, Nagoya, Japan
yagiura@nagoya-u.jp

Summary. Three variants of the two dimensional packing problem are considered, where the items to be packed are (a) rectangles with fixed widths and heights, (b) rectangles with adjustable widths and heights, or (c) irregular shapes. All problems are solved by hybrid metaheuristics that combine local search and mathematical programming techniques of linear, nonlinear and/or dynamic programming. Basic ideas of these algorithms are explained on a unified basis, together with some computational results. It appears to indicate that mathematical programming is a vital tool for enhancing metaheuristic algorithms.

1 Introduction

We consider in this chapter the *two-dimensional packing problem* that asks to pack a given set of *items* into a given *container* without mutual overlap. There are many variants of this problem depending upon whether the items are rectangles or have irregular shapes, and how minimization of the container is defined.

Most of these variants are NP-hard, since they contain as a special case the bin packing problem, which is already known to be NP-hard. Local search and metaheuristic algorithms have been playing major roles in obtaining good approximate solutions for practical uses. We observe that many of such algorithms contain subproblems that ask to pack given items in an optimal manner under certain constraints, and such subproblems are solvable by techniques known as dynamic, linear and nonlinear programming. Thanks to the recent progress of mathematical programming, efficient softwares are available for the cases of linear and nonlinear programming. The resulting algorithms are *hybrid metaheuristics* in the sense that they are combinations of metaheuristics and mathematical programming.

T. Ibaraki et al.: *Hybrid Metaheuristics for Packing Problems*, Studies in Computational Intelligence (SCI) **114**, 185–219 (2008)
www.springerlink.com

In this chapter, we deal with the following types of the packing problem:

(a) Items are rectangles, where the size (i.e., *width* and *height*) of each rectangle is fixed in advance.
(b) Items are soft rectangles, whose sizes can be adjusted.
(c) Items have irregular shapes, which may be neither rectangular nor convex.

In all these problems, the container is assumed to be a rectangle with width W and height H, and its size is minimized in the sense of WH (area), $2(W+H)$ (perimeter) or H (height) while fixing its width to $W = W^*$ (a given constant). The last type is called the problem of *strip packing*.

We describe a method that enables us to effectively use dynamic programming for solving (a), and nonlinear and linear programming techniques for solving (b) and (c). We also touch upon the problem of packing *rectangles with weights* under the constraints on the location of the center of gravity and their moment. Nonlinear programming is also useful for such a variant.

The organization of this chapter is as follows. Sect. 2 gives basic definitions and presents metaheuristic frameworks for the above packing problems. It then specifically discusses problem (a), in which dynamic programming techniques are employed. Sect. 3 considers problem (b), in which linear programming and nonlinear programming are used. Sect. 4 briefly mentions that a similar method can be used to pack rectangles with weights. Finally, Sect. 5 deals with problem (c), where linear programming and nonlinear programming are again used. All sections are concluded with some computational results.

2 Rectangle Packing Problem

Packing a number of rectangles, each having a fixed size, is perhaps most popular among packing problems. It is encountered in many industrial applications, such as wood, glass and steel manufactures, LSI and VLSI design, and newspaper paging. As the problem is NP-hard, various approximation algorithms have been proposed [18, 37]. Metaheuristics have also been utilized [7, 15, 29, 32, 43].

We mainly treat the strip packing problem in this section. It may have additional constraints concerning orientation of rectangles and guillotine cut restriction [21, 59]. As for the orientation of rectangles, the following three situations have been considered in the literature: (1) each rectangle can be rotated by any angle, (2) each rectangle can be rotated by 90 degrees, and (3) each rectangle has a fixed orientation. Rotation of rectangles is not allowed in newspaper paging or when the rectangles to be cut are decorated or corrugated, whereas rotation is allowed in the case of plain materials. The guillotine cut constraint signifies that the rectangles must be obtained through a sequence of edge-to-edge cuts parallel to the edges of the container, which is usually imposed by technical limitations of the automated cutting machines. In this section, we focus on case (3) without the guillotine cut constraint.

After presenting in Sect. 2.1 a mathematical programming formulation of the strip packing problem of rectangles, Sect. 2.2 describes standard coding schemes and decoding algorithms. Then the framework of local search is presented in Sect. 2.3 together with its basic components such as neighborhoods and evaluation functions. A brief explanation of metaheuristics is also given at the end of this subsection. A hybrid metaheuristic algorithm for the strip packing problem is then presented in Sect. 2.4, putting emphasis on neighborhood reductions and fast decoding algorithms by dynamic programming. Sect. 2.5 concludes this section by reporting some computational results.

2.1 Problem Formulation

We are given n items $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of rectangular shape, where each rectangle $I_i \in \mathcal{I}$ has fixed width w_i and height h_i. We are asked to pack all items orthogonally into the *strip* (container) of a fixed width $W = W^*$ (a given constant) and a variable height H so as to minimize H. "Orthogonally" means that an edge of each item is parallel to an edge of the strip.

We describe the location of an item I_i by the coordinate (x_i, y_i) of its bottom-left corner. The problem is formally described as follows.

$$\begin{array}{lll}
\text{minimize} & H & \\
\text{subject to} & 0 \le x_i \le W^* - w_i, \quad 1 \le i \le n & (1) \\
& 0 \le y_i \le H - h_i, \quad 1 \le i \le n & (2) \\
& \text{At least one of the next four inequalities} & \\
& \text{holds for every pair } I_i \text{ and } I_j \text{ of rectangles:} & \\
& x_i + w_i \le x_j, \qquad x_j + w_j \le x_i, & \\
& y_i + h_i \le y_j, \qquad y_j + h_j \le y_i. & (3)
\end{array}$$

The constraints (1) and (2) tell that every rectangle must be placed into the strip. The constraint (3) means that no two rectangles overlap; that is, each inequality describes one of the four relative locations required to avoid mutual overlap: right-of, left-of, above and below.

We call a solution of the above problem (i.e., locations of all rectangles) as a *placement*. Fig. 1 shows a placement obtained by the algorithm of this section, for the benchmark known as rp100.[1]

2.2 Coding Schemes and Decoding Algorithms

In order to design algorithms for the rectangle packing problem, coding schemes and decoding algorithms should be discussed first.

[1] Available from `http://www.simplex.t.u-tokyo.ac.jp/~ imahori/packing/instance.html`

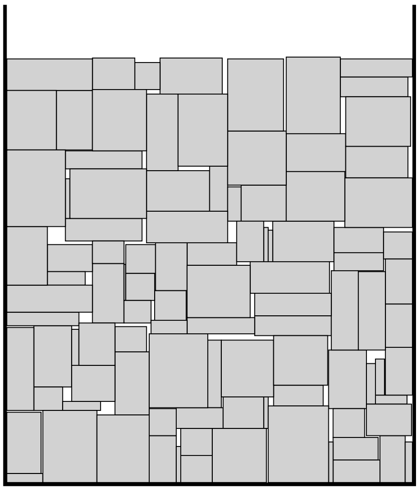


Fig. 1. An example of strip packing (rp100 with $W^* = 450$)

In the rectangle packing problem, if we consider the x and y coordinates of each rectangle explicitly, an effective search will be difficult because the number of solutions is uncountably many and elimination of mutual overlap is not easy. To overcome this difficulty, most algorithms utilize some coding schemes. A *coding scheme* consists of a set of coded solutions and a mapping from coded solutions to placements, and a *decoding algorithm* computes for a given coded solution the corresponding placement. Desirable properties of a coding scheme and a decoding algorithm are summarized as follows.

1. There is a coded solution that represents an optimal placement.
2. The number of all possible coded solutions is finite, where a smaller number is preferable provided that property 1 holds.
3. Every coded solution represents a feasible placement.
4. A fast algorithm (running in polynomial time) for decoding is available.

Some of the coding schemes in the literature satisfy all of the above four properties, but others do not.

A standard coding scheme is a permutation of n rectangles, which specifies an order of placing rectangles one by one according to a certain decoding rule. The number of all permutations is $n!$, and every permutation corresponds to a placement without mutual overlap. In order to find a good permutation among them, heuristics and metaheuristics are often used. A typical heuristics is just sorting the rectangles by some criteria; e.g., decreasing width, decreasing height, decreasing perimeter and decreasing area. In other cases, good permutations are searched by local search or metaheuristics. The corresponding placements are then computed by decoding algorithms, which are also called placement rules; e.g., first fit [18], bottom left [6], and best fit [15]

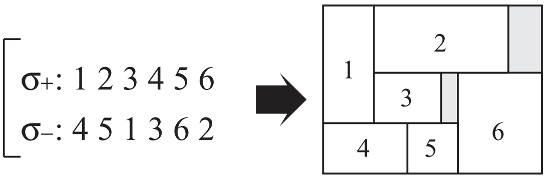


Fig. 2. A sequence pair $\sigma = (\sigma_+, \sigma_-)$ and its placement

algorithms. We should note that, for a given permutation, different decoding algorithms may give different placements. It is therefore very important to choose a good decoding algorithm.

There are other types of coding schemes. All the schemes we explain hereafter specify relative positions between each pair of rectangles I_i and I_j (i.e., one of the four inequalities of (3)). The placement for a coded solution is defined as the best one among those satisfying the constraints posed by the coded solution.

Sequence Pair Most well-known in this category is perhaps the *sequence pair* coding scheme [45]. A sequence pair is a pair of permutations $\sigma = (\sigma_+, \sigma_-)$ of $\{1, 2, \ldots, n\}$, where $\sigma_+(l) = i$ (equivalently $\sigma_+^{-1}(i) = l$) means that rectangle I_i is in the lth position of permutation σ_+. Permutation σ_- is similarly defined. A sequence pair $\sigma = (\sigma_+, \sigma_-)$ specifies which of the four conditions in (3) holds for each pair of I_i and I_j, based on the partial orders $\preceq_\sigma^x$ and $\preceq_\sigma^y$ defined by

$$\begin{aligned} i \preceq_\sigma^x j &\iff \sigma_+^{-1}(i) \leq \sigma_+^{-1}(j) \text{ and } \sigma_-^{-1}(i) \leq \sigma_-^{-1}(j), \\ i \preceq_\sigma^y j &\iff \sigma_+^{-1}(i) \geq \sigma_+^{-1}(j) \text{ and } \sigma_-^{-1}(i) \leq \sigma_-^{-1}(j). \end{aligned}$$

This says that, if i appears before j in both σ_+ and σ_-, then $i \preceq_\sigma^x j$, i.e., I_i is placed to the left of I_j, while if i appears after j in σ_+, but before j in σ_-, then $i \preceq_\sigma^y j$, i.e., I_i is placed under I_j. For example, in Fig. 2, "1 is placed before 2 in both permutations, and hence 1 is placed to the left of 2," "2 is placed before 5 in σ_+ and after 5 in σ_-, and hence 2 is placed above 5" and so on. Exactly one of $i \preceq_\sigma^x j$, $j \preceq_\sigma^x i$, $i \preceq_\sigma^y j$, $j \preceq_\sigma^y i$ always holds for any pair of I_i and I_j, and the constraints in (3) are specified as follows:

$$\begin{aligned} x_i + w_i \leq x_j \quad &\text{if} \quad i \preceq_\sigma^x j, \qquad x_j + w_j \leq x_i \quad \text{if} \quad j \preceq_\sigma^x i, \\ y_i + h_i \leq y_j \quad &\text{if} \quad i \preceq_\sigma^y j, \qquad y_j + h_j \leq y_i \quad \text{if} \quad j \preceq_\sigma^y i. \end{aligned} \tag{4}$$

Decoding Algorithms Once we are given a sequence pair, we can compute a best placement satisfying the constraints (1), (2) and (4) in polynomial time; e.g., Murata et al. [45] proposed an $O(n^2)$ time decoding algorithm, Takahashi [56] improved it to $O(n \log n)$, Tang et al. [57] further improved it to $O(n \log \log n)$. Here, we briefly explain basic ideas in these algorithms. From the definition, we can obtain a feasible placement even if we compute the x and y coordinates separately; thus, we discuss only the x direction. Let us define a set J_i for each rectangle I_i as follows:

$$J_i = \{j \mid \sigma_+^{-1}(j) < \sigma_+^{-1}(i) \text{ and } \sigma_-^{-1}(j) < \sigma_-^{-1}(i)\}.$$

Then, the horizontal coordinates of each rectangle I_i can be computed by

$$x_i = \begin{cases} 0, & \text{if } J_i = \emptyset \\ \max_{j \in J_i}\{x_j + w_j\}, & \text{otherwise.} \end{cases} \tag{5}$$

If we compute (5) naively for all i, it takes $O(n^2)$ time. But it can be reduced to $O(n \log n)$ by using a binary search tree as data structure.

Other Coding Schemes There are other coding schemes. One traditional coding scheme is to represent a solution by a binary tree with n leaves [53], which correspond to rectangles. Each internal node has a label 'h' or 'v', where h stands for horizontal and v stands for vertical. In this scheme, one of the four relative locations in (3) is retrieved for each pair of rectangles as follows: Let u be the least common ancestor of I_i and I_j. If u has label 'h' and I_i is a left descendant of u (equivalently I_j is a right descendant of u), then I_i is placed to the left of I_j. If the label of u is 'v', then I_i is placed below I_j. For a given binary tree, naive decoding algorithms can compute the best placement satisfying the above constraints in $O(n)$ time. Note however that this coding scheme can represent only slicing structures; i.e., each placement obtained by this representation always satisfies the guillotine cut constraint.

Guo et al. [27] and Chang et al. [16] also proposed coding schemes based on tree structures called O-tree and B*-tree, respectively. Their coding schemes can represent both of slicing and non-slicing structures, and compute the corresponding placement in $O(n)$ time.

Nakatake et al. [48] proposed another coding scheme called bounded slice-line grid (BSG in short). BSG has a grid structure and each pair of rooms in the grid represents one of the four relative positions. All the rectangles are assigned to rooms, where at most one rectangle can be assigned to each room. It is argued that $O(n)$ number of rooms are sufficient in practice. Thus, given a coded solution (i.e., an assignment of rectangles to rooms), the corresponding placement can be obtained in $O(n)$ time. A hybrid metaheuristic algorithm using BSG coding scheme was proposed by Imahori et al. [33].

2.3 Local Search and Simple Metaheuristics

In this section, we explain the general idea of *local search* (LS in short). LS starts from an initial solution and repeats the replacement of the current solution with a better solution in its neighborhood until no better solution is found in the neighborhood. Here we focus on the LS for the strip packing problem of rectangles, which is based on sequence pairs, though it can be easily generalized to other settings.

```
Algorithm LS
Input: Widths $w_i$ and heights $h_i,\ i = 1, 2, \ldots, n$, of rectangles
       and the width $W^*$ of the container.
Output: A placement of all rectangles.

  Step 1 (initialization): Construct an initial sequence pair $\sigma = (\sigma_+, \sigma_-)$.
      Compute the placement $\boldsymbol{v}(\sigma)$ and its objective value $z(\sigma)$ corresponding
      to $\sigma$, and let $\boldsymbol{v} := \boldsymbol{v}(\sigma)$ and $z := z(\sigma)$, where $\boldsymbol{v}$ and $z$
      denote the incumbent solution and its value, respectively.
  Step 2 (local search): Select a new $\sigma' \in N(\sigma)$, where $N(\sigma)$ denotes
      the neighborhood of $\sigma$, and compute $\boldsymbol{v}(\sigma')$ and $z(\sigma')$. If $z(\sigma') < z$
      holds, then let $\boldsymbol{v} := \boldsymbol{v}(\sigma'),\ z := z(\sigma'),\ \sigma := \sigma'$ and return to
      Step 2. If there is no new sequence pair left in $N(\sigma)$, go to Step 3;
      otherwise return to Step 2.
  Step 3 (termination): Output $\boldsymbol{v}$, and halt.
```

The solution output in Step 3 is *locally optimal* in neighborhood $N(\sigma)$.

The search space of the above LS is the set of sequence pairs, whose size is $(n!)^2$. An initial sequence pair is often generated randomly, but it is also possible to generate it by some heuristic algorithm. The quality of the solutions generated during the search is evaluated by a given *evaluation function* $z(\sigma)$. It may be equal to the objective function or may be modified from it to make the search more effective.

Standard Neighborhoods The performance of LS critically depends on how the neighborhood is designed. $N(\sigma)$ is commonly defined as the set of sequence pairs obtained from σ by applying certain local operations. Typical operations are *shift, swap* and *swap** [31, 32, 45], defined as follows.

1. Shift: This operation moves an element i in σ_+ (or σ_-) to the first position or to the next position of an element j. The shift neighborhood is defined by applying this to all pairs of i and j. If only one of σ_+ or σ_- is considered, it is the *single-shift neighborhood*, while if each shift operation is applied to both σ_+ and σ_-, then it is the *double-shift neighborhood.* The size of single-shift neighborhood is $O(n^2)$ since all pairs of i and j are considered. The size of double-shift neighborhood is $O(n^3)$ if the element j is selected in σ_+ and σ_- independently. On the other hand, if the same element j is always selected in both σ_+ and σ_-, then the size becomes $O(n^2)$. In this case, the element i is inserted before and after j, respectively, thereby examining four positions for each j. This is called the *limited double-shift neighborhood.*

2. Swap: This operation exchanges the positions of i and j in σ_+ (or in σ_-). The *single-swap neighborhood* and *double-swap neighborhood* are defined similarly to the case of shift neighborhood. The sizes of the resulting neighborhoods are $O(n^2)$.

3. Swap*: Let i and j in σ_+ satisfy $\sigma_+(\alpha) = i$ and $\sigma_+(\beta) = j$, with $\alpha < \beta$. Then for each γ with $\alpha \leq \gamma < \beta$, i and j are moved to location γ in

the manner $\sigma'_+(\gamma) = j$ and $\sigma'_+(\gamma+1) = i$, while keeping the same relative positions of other elements. This swap* operation can also be defined for σ_-. The swap* operations are usually applied to only one of σ_+ and σ_-, yielding the *swap* neighborhood*. If all combinations of i, j, γ are considered, its size becomes $O(n^3)$.

The effects of these operations may be intuitively explained as follows, where we assume for simplicity that I_i is constrained to the left of I_j by σ. A shift operation applied to σ_+ changes the relative positions of I_i and I_j to "I_j above I_i", causing side effects on relative positions between I_i and other rectangles. A single-swap operation on σ_+ (resp., σ_-) changes the relative position "I_i to the left of I_j" to "I_j above I_i" (resp., "I_i above I_j"). On the other hand, a double-swap operation exchanges only the locations of I_i and I_j, without changing the relative positions of other rectangles. A swap* operation brings I_i and I_j together to their middle locations specified by γ.

Evaluation Function As an example of defining $z(\sigma)$, we compute the placement of all rectangles by a decoding algorithm and use the height of the container

$$y_{\max} = \max_{I_i \in \mathcal{I}} \{y_i + h_i\},$$

and the maximum x coordinate,

$$x_{\max} = \max_{I_i \in \mathcal{I}} \{x_i + w_i\}.$$

For some sequence pair, it may happen that $x_{\max} > W^*$ holds, that is, there exists a rectangle that protrudes from the strip width. Penalizing such infeasibility, the following evaluation function is defined.

$$z(\sigma) = y_{\max} + M \times \max\{0,\ x_{\max} - W^*\}, \tag{6}$$

where M is a large constant.

As we may frequently encounter $z(\sigma') = z$ in Step 2 of LS, some mechanisms to break such ties are usually incorporated.

Metaheuristics In general, if LS is applied only once, many solutions of better quality may remain unvisited in the search space. To improve the situation, many variants of LS have been developed, and their frameworks are called *metaheuristics*. The *iterated local search* (ILS) [36] is one of the simplest metaheuristic approaches, in which many initial solutions are generated by slightly perturbing a good solution obtained during the search so far. In order to improve the performance of ILS, it is important to generate initial solutions which retain some features of good solutions. It may also be necessary to introduce a mechanism that avoids a cycling of solutions.

2.4 Hybrid Metaheuristics for Rectangle Packing

We now describe a hybrid metaheuristic algorithm proposed by [32] for the rectangle packing problem, which is based on the sequence pair coding scheme.

We first explain some ideas to decrease the neighborhood size, and then show efficient evaluation algorithms of dynamic programming.

Critical Paths and Neighborhood Reductions As noted in Sect. 2.3, the size of the shift neighborhood is $O(n^2)$ or $O(n^3)$. In order to reduce this size without sacrificing its effectiveness, we restrict (1) the choice of element i that will be shifted in σ, and (2) the positions of j to where the i is inserted.

For this purpose, critical paths are utilized. Critical paths are defined for both of the x (horizontal) and y (vertical) directions. We first consider the y direction. Given a placement, a directed graph $G = (V, E)$ and subsets $S, T \subseteq V$ are defined as follows:

$$\begin{aligned}
&V = \{1, 2, \ldots, n\},\\
&(i, j) \in E \Longleftrightarrow i \preceq_{\sigma}^{y} j \ \text{ and } \quad y_i + h_i = y_j,\\
&S = \{i \,|\, y_i = 0\}, \quad T = \{i \,|\, y_i + h_i = y_{\max}\}.
\end{aligned}$$

Then, a *critical path* is defined as a directed path in G, whose initial vertex s is in S and final vertex t is in T. For any placement obtained from a sequence pair σ, S and T are nonempty and there is at least one critical path. It is possible to find all rectangles in critical paths in $O(n)$ time. The definition for the x direction is similar except for the following situation: If $x_{\max} \leq W^*$, we need not to reduce $x_{\max}$, and hence critical paths of x direction are not considered. To reduce the size of shift neighborhood, we shift only those rectangles I_i in critical paths. It is easy to show that a solution is locally optimal in the original shift neighborhood if no improved solution is found in the reduced neighborhood.

In order to reduce neighborhood sizes further, the following neighborhoods with some restrictions are considered.

(1) The single-shift neighborhood, whose size is $O(n^2)$.
(2) The limited double-shift neighborhood, whose size is $O(n^2)$.
(3) As another reduced double-shift neighborhood, an element i is inserted only to the positions close to the current position of i in σ_+ and σ_-, respectively. To control its size, the distance from the original position to the new positions is restricted within $a\sqrt{n}$, where a is a parameter. The size of this neighborhood is $O(a^2 n)$ for each i. This is called the *proximal double-shift neighborhood.*

Fast Decoding by Dynamic Programming Dynamic programming can be effectively used to compute $x_{\max}$'s for all solutions in the neighborhoods as discussed above.

We first consider the double-shift neighborhood. Assume that a rectangle I_i will be shifted. Here, a shift operation is regarded as consecutive two operations: Deleting i from σ (the resulting sequence pair is denoted by $\tilde{\sigma}$), and inserting i into other positions in $\tilde{\sigma}_+$ and $\tilde{\sigma}_-$ of $\tilde{\sigma}$. For the sequence pair $\tilde{\sigma}$, the corresponding placement is computed in $O(n \log n)$ time by a decoding algorithm noted in Sect. 2.2. Let $\tilde{x}_j$ be the x coordinate of rectangle I_j and $\tilde{x}_{\max} = \max_{I_j \in \mathcal{I} \setminus \{I_i\}} \{\tilde{x}_j + w_j\}$.

Now the element i is inserted to the αth position in σ_+ and to the βth position in σ_-, respectively, and denote the resulting length of the critical path by $\tilde{x}_{\max}(\alpha, \beta)$. It is important to know whether I_i is in the horizontal critical path or not. If it is not in the critical path, then $\tilde{x}_{\max}(\alpha, \beta)$ is equal to $\tilde{x}_{\max}$. Otherwise the length of the critical path that includes rectangle I_i can be computed by dynamic programming. Let us define $\tilde{J}^{\mathrm{f}}_{\alpha,\beta}$, $\tilde{J}^{\mathrm{b}}_{\alpha,\beta}$, $\tilde{f}(\alpha, \beta)$ and $\tilde{b}(\alpha, \beta)$ for each pair of α and β such that $1 \le \alpha, \beta \le n$ as follows:

$\tilde{J}^{\mathrm{f}}_{\alpha,\beta} = \{j \mid \tilde{\sigma}_+^{-1}(j) < \alpha,\ \tilde{\sigma}_-^{-1}(j) < \beta\}$,
$\tilde{J}^{\mathrm{b}}_{\alpha,\beta} = \{j \mid \tilde{\sigma}_+^{-1}(j) \ge \alpha,\ \tilde{\sigma}_-^{-1}(j) \ge \beta\}$,
$\tilde{f}(\alpha, \beta)$: length of the critical path for the set $\{I_j \mid j \in \tilde{J}^{\mathrm{f}}_{\alpha,\beta}\}$,
$\tilde{b}(\alpha, \beta)$: length of the critical path for the set $\{I_j \mid j \in \tilde{J}^{\mathrm{b}}_{\alpha,\beta}\}$.

Based on the idea of dynamic programming, $\tilde{f}(\alpha, \beta)$ (respectively, $\tilde{b}(\alpha, \beta)$) can be computed by

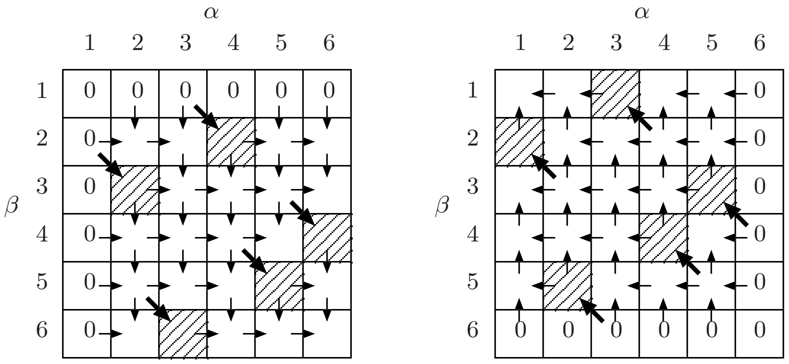
$$\tilde{f}(\alpha,\beta) = \begin{cases} 0, & \text{if } \alpha = 1 \text{ or } \beta = 1 \\ \max\{\tilde{f}(\alpha-1,\beta), \tilde{f}(\alpha,\beta-1)\}, & \text{if } \tilde{\sigma}_+(\alpha-1) \ne \tilde{\sigma}_-(\beta-1) \\ \tilde{f}(\alpha-1,\beta-1) + w_j, & \text{if } \tilde{\sigma}_+(\alpha-1) = \tilde{\sigma}_-(\beta-1) = j, \end{cases} \tag{7}$$

$$\tilde{b}(\alpha,\beta) = \begin{cases} 0, & \text{if } \alpha = n \text{ or } \beta = n \\ \max\{\tilde{b}(\alpha+1,\beta), \tilde{b}(\alpha,\beta+1)\}, & \text{if } \tilde{\sigma}_+(\alpha) \ne \tilde{\sigma}_-(\beta) \\ \tilde{b}(\alpha+1,\beta+1) + w_j, & \text{if } \tilde{\sigma}_+(\alpha) = \tilde{\sigma}_-(\beta) = j, \end{cases} \tag{8}$$

for all pairs of $\alpha = 1, 2, \ldots, n$ and $\beta = 1, 2, \ldots, n$ (resp., for all pairs of $\alpha = n, n-1, \ldots, 1$ and $\beta = n, n-1, \ldots, 1$). See Fig. 3 as an example of showing the computation order of $\tilde{f}(\alpha, \beta)$ and $\tilde{b}(\alpha, \beta)$. Each box in this figure corresponds to $\tilde{f}(\alpha, \beta)$ (resp., $\tilde{b}(\alpha, \beta)$) for each pair of α and β, and arrows show how to compute each value. For the example of $\tilde{\sigma}_+$ and $\tilde{\sigma}_-$ given in the figure, the value of each shaded box is computed by the third formula of (7) since $\tilde{\sigma}_+(\alpha-1) = \tilde{\sigma}_-(\beta-1)$ holds (resp., (8) since $\tilde{\sigma}_+(\alpha) = \tilde{\sigma}_-(\beta)$) holds). For the new placement after inserting I_i into the αth position of $\tilde{\sigma}_+$ and the βth position of $\tilde{\sigma}_-$, the critical path length that includes rectangle I_i can be computed by $\tilde{f}(\alpha, \beta) + w_i + \tilde{b}(\alpha, \beta)$. Thus,

$$\tilde{x}_{\max}(\alpha, \beta) = \max\{\tilde{x}_{\max}, \tilde{f}(\alpha, \beta) + w_i + \tilde{b}(\alpha, \beta)\}. \tag{9}$$

This algorithm takes $O(n \log n)$ time for the original decoding algorithm applied to $\tilde{\mathcal{I}}$ and $\tilde{\sigma}$. The time to compute $\tilde{f}(\alpha, \beta)$ and $\tilde{b}(\alpha, \beta)$ for all pairs of α and β by (7) and (8) is $O(n^2)$. The time to compute $\tilde{x}_{\max}(\alpha, \beta)$ by (9) is $O(1)$ for each pair of α and β, and it becomes $O(n^2)$ for all pairs of α and β. In summary, the total computation time of this algorithm, i.e., time to evaluate all solutions when rectangle I_i is shifted in the double-shift neighborhood is



(a) computation of $\tilde{f}(\alpha,\beta)$ (b) computation of $\tilde{b}(\alpha,\beta)$

($\tilde{\sigma}_+ : 1,2,3,4,5, \quad \tilde{\sigma}_- : 3,1,5,4,2.$)

Fig. 3. An example of computing $\tilde{f}(\alpha,\beta)$ and $\tilde{b}(\alpha,\beta)$

$O(n^2)$. This implies that it takes $O(1)$ amortized time to evaluate one coded solution in the double-shift neighborhood. The space complexity of this DP algorithm is $O(n^2)$.

We then consider the limited double-shift neighborhood. As in the above algorithm, an element i is deleted from σ and the placement for $\tilde{\sigma}$ is computed. In this case, the $\tilde{f}(\alpha,\beta)$ and $\tilde{b}(\alpha,\beta)$ only for necessary pairs of α and β should be computed. That is, when the element i is inserted before or after $j = \tilde{\sigma}_+(\alpha-1) = \tilde{\sigma}_-(\beta-1)$ in $\tilde{\sigma}_+$ and $\tilde{\sigma}_-$, respectively, we only need to compute $\tilde{f}(\alpha-1,\beta-1)$, $\tilde{f}(\alpha-1,\beta)$, $\tilde{f}(\alpha,\beta-1)$ or $\tilde{f}(\alpha,\beta)$. However, these can be immediately given by $\tilde{f}(\alpha-1,\beta-1) = \tilde{f}(\alpha-1,\beta) = \tilde{f}(\alpha,\beta-1) = \tilde{x}_j$ and $\tilde{f}(\alpha,\beta) = \tilde{x}_j + w_j$. Thus the computation time for each solution is $O(1)$. The case of $\tilde{b}(\alpha,\beta)$ is similar.

To evaluate solutions obtainable in the single-shift neighborhood, where i is shifted in σ_+, we compute $\tilde{f}(\alpha,\beta)$ for all $1 \le \alpha \le n$ and $\beta = \sigma_-^{-1}(i)$ by

$$\tilde{f}(\alpha,\beta) = \begin{cases} \max\{\tilde{f}(\alpha-1,\beta), \tilde{x}_{j'} + w_{j'}\}, & \text{if } \tilde{\sigma}_-^{-1}(j') \le \beta - 2 \\ \tilde{x}_{j'} + w_{j'}, & \text{if } \tilde{\sigma}_-^{-1}(j') = \beta - 1 \\ \tilde{f}(\alpha-1,\beta), & \text{otherwise,} \end{cases} \tag{10}$$

where $j' = \tilde{\sigma}_+(\alpha-1)$. Similar formula can be derived if i is shifted in σ_-. In both cases, the time to compute $\tilde{f}(\alpha,\beta)$ for all necessary α and β is $O(n)$.

To evaluate solutions in the proximal double shift neighborhood, we compute $\tilde{f}(\alpha,\beta)$ for all $\alpha_l \le \alpha \le \alpha_u$ and $\beta_l \le \beta \le \beta_u$, where $\alpha_u - \alpha_l \le 2a\sqrt{n}$ and $\beta_u - \beta_l \le 2a\sqrt{n}$ hold. We first compute $\tilde{f}(\alpha,\beta_l)$ for $1 \le \alpha \le \alpha_u$ by (10) and $\tilde{f}(\alpha_l,\beta)$ for $1 \le \beta \le \beta_u$ by the σ_- version of (10). Then, (7) is used in computing $\tilde{f}(\alpha,\beta)$ for all $\alpha_l + 1 \le \alpha \le \alpha_u$ and $\beta_l + 1 \le \beta \le \beta_u$. Therefore, $\tilde{f}(\alpha,\beta)$ for all necessary α and β can be computed in $O(a^2 n)$ time.

Table 1. Comparison of three algorithms for the rectangle packing problem

		BLFDW		SA-BLF		HM-SP	
category	n	ratio	time	ratio	time	ratio	time
C1	17	89	< 0.1	96	42	97.56	10
C2	25	84	< 0.1	94	144	93.75	15
C3	29	88	< 0.1	95	240	96.67	20
C4	49	95	< 0.1	97	1980	96.88	150
C5	73	95	< 0.1	97	6900	97.02	500
C6	97	95	< 0.1	97	22920	96.85	1000
C7	197	95	0.64	96	250860	96.55	3600
CPU		PentiumPro		PentiumPro		Pentium3	
clock freq.		200 MHz		200 MHz		1.0 GHz	

In summary, all solutions in the limited and proximal double-shift neighborhoods can be evaluated in $O(n \log n)+O(a^2 n)$ time for each shifted i. Thus the amortized computation time for one coded solution becomes $O(\log n)$.

2.5 Computational Results

We give some computational results of heuristic, metaheuristic, and hybrid metaheuristic algorithms on test instances given by Hopper and Turton [29], which is a well known benchmark used in the literature (e.g., [15, 29, 32]). There are seven different categories called C1, C2, ..., C7 with the number of rectangles ranging from 17 to 197, where each category has three instances. We compare the following three algorithms: (1) A heuristic algorithm BLF-DW (bottom left fill with decreasing width) proposed by Baker et al. [6] and implemented by Hopper and Turton [29] (denoted BLFDW), (2) a simulated annealing algorithm with BLF algorithm by Hopper and Turton [29] (denoted SA-BLF) and (3) a hybrid metaheuristic algorithm based on ILS and dynamic programming by Imahori et al. [32] (denoted HM-SP). Results are shown in Table 1. Column "ratio" shows the average of the following ratio,

$$100 \times \frac{\text{(total area of rectangles)}}{\text{(output value of } H) \times W^*},$$

(i.e., the larger the better). Column "time" shows the computation time in seconds for one instance, where the CPUs used for the computation are given in the row "CPU". Based on the benchmark results of SPECint from SPEC web page (`http://www.specbench.org/`), the Pentium3 1.0 GHz is about six times faster than the PentiumPro 200 MHz.

From the table, we observe that BLFDW (i.e., a heuristic algorithm) is much faster than others, but the quality of output solutions is slightly worse. By using (hybrid) metaheuristics, it is possible to attain better quality of solutions. We also observe that HM-SP is faster than SA-BLF to attain similar quality.

3 Packing Soft Rectangles

In this section we assume that all rectangles are soft. Namely, the width w_i and the height h_i of rectangle I_i can be adjusted within given constraints. For example, the constraints may specify their lower and upper bounds:

$$\begin{aligned} w_i^L &\leq w_i \leq w_i^U, \\ h_i^L &\leq h_i \leq h_i^U. \end{aligned} \tag{11}$$

We may also add the constraint that the *aspect ratio* h_i/w_i is bounded between its lower bound r_i^L and upper bound r_i^U:

$$r_i^L w_i \leq h_i \leq r_i^U w_i. \tag{12}$$

Another type of constraint common in applications is that each rectangle I_i must have either a given perimeter L_i or a given area A_i (or both):

$$2(w_i + h_i) \geq L_i, \tag{13}$$

$$w_i h_i \geq A_i. \tag{14}$$

In addition, we may consider that the locations (x_i, y_i) of rectangles I_i are predetermined in some intervals:

$$\begin{aligned} x_i^L &\leq x_i \leq x_i^U, \\ y_i^L &\leq y_i \leq y_i^U. \end{aligned} \tag{15}$$

To our knowledge, there is not much literature on the problem using soft rectangles, except such papers as [17, 35, 46, 60] containing algorithms and [47] containing a theoretical analysis, even though the problem has wide applications.

Applications can be found, for example, in VLSI floorplan design [17, 35, 45, 46, 60] and in resource constrained scheduling. In the VLSI design, each rectangle represents a block of logic circuits consisting of a certain number of transistors, which occupy certain area, and must have at least some perimeter length to accommodate connection lines to other blocks. The shape of each rectangle is adjustable, but is required to satisfy the constraints as stated above. In a scheduling application, each rectangle may represent a job, to be assigned to an appropriate position on the time axis (horizontal), where its width gives the processing time of the job and its height represents the amount of resource (per unit time) invested to process the job. In this case, the area of the rectangle represents the total amount of resource consumed by the job, which is again required to satisfy the above constraints.

In the following, we focus on the local search algorithm proposed by [30], which is based on the sequence pair (defined in Sect. 2.2). It is described in Sect. 3.1 that, given a sequence pair, the problem of computing the sizes and placement of rectangles is formulated as a linear programming problem or nonlinear programming problem (more exactly, convex programming problem) depending on the constraints and objective functions. Although these mathematical programming algorithms are quite efficient, it still consumes some amount of time, and it is carefully considered in Sect. 3.2 how to reduce the neighborhood sizes. Finally, in Sect. 3.3, some computational results are reported.

3.1 Problem Statement

As the objective function, we may choose to minimize the perimeter of the container, i.e.,

$$\text{minimize} \quad 2(W+H) \tag{16}$$

or to minimize its area,

$$\text{minimize} \quad WH, \tag{17}$$

where W and H are the variables that satisfy

$$\begin{aligned} x_i + w_i &\le W \quad \text{for all} \;\; i, \\ y_i + h_i &\le H \quad \text{for all} \;\; i. \end{aligned} \tag{18}$$

In the strip packing problem, the width of the container is fixed to $W = W^*$, and its height H is minimized:

$$\text{minimize} \quad H, \tag{19}$$

under the constraints

$$\begin{aligned} x_i + w_i &\le W^* \quad \text{for all} \;\; i, \\ y_i + h_i &\le H \quad \text{for all} \;\; i. \end{aligned} \tag{20}$$

Therefore, if a sequence pair σ is specified, we are required to solve the mathematical programming problem $P(\sigma)$ to minimize objective function (16), (17) or (19) under the constraints:

$$\begin{array}{ll} (11), (12), (13) \text{ (and/or (14)), (15), (18) (or (20))} & \text{for all} \;\; i, \\ (4) & \text{for all} \;\; i \text{ and } j, \\ x_i, \; y_i \ge 0 & \text{for all} \;\; i. \end{array} \tag{21}$$

It is important to note that the feasible region defined by constraints (21) is *convex*, as illustrated in Fig. 4. Therefore, if the objective function is convex, we obtain a convex programming problem, and can solve it by existing efficient algorithms (e.g., [12, 49]) (to be more precise, we need to resort to semidefinite programming to handle constraints of type (14)). This is the case when we want to minimize (16) or (19). The problem with objective function (17) is not a convex programming problem, but is a so-called multiplicative programming problem for which some efficient approaches are also known (e.g., [39]).

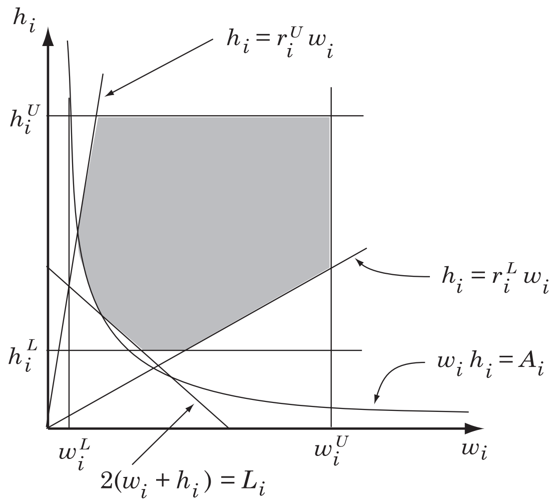


Fig. 4. Feasible region for the w_i and h_i of rectangle I_i

Two Test Problems From the above varieties, two simple problems will be discussed in the following. The first type minimizes the perimeter of the container under the perimeter constraints (13) of rectangles.

$$\begin{array}{rlll} P_{\text{peri}}(\sigma) : \text{minimize} & 2(W+H) & & \\ \text{subject to} & (11), (13), (18) & \text{for all } i & \\ & (4) & \text{for all } i \text{ and } j & (22) \\ & x_i,\ y_i \geq 0 & \text{for all } i. & \end{array}$$

This is a linear programming problem for each given sequence pair σ, and is called the *perimeter minimization problem.*

The second type is the strip packing problem under the area constraints (14) of rectangles, which is formulated as a convex programming problem.

$$\begin{array}{rlll} P_{\text{area}}(\sigma) : \text{minimize} & H + Ms & & \\ \text{subject to} & (11), (14) & \text{for all } i & \\ & x_i + w_i \leq W^* + s & \text{for all } i & \\ & y_i + h_i \leq H & \text{for all } i & \\ & (4) & \text{for all } i \text{ and } j & (23) \\ & s \geq 0 & & \\ & x_i,\ y_i \geq 0 & \text{for all } i. & \end{array}$$

Here the variable s is introduced to keep the problem always feasible, by adding penalty term Ms to the objective function (19) with a large positive constant M. This is called the *area minimization problem.*

3.2 Neighborhood Reductions

Starting from the standard neighborhoods as described in Sect. 2.3, we consider how to reduce their sizes further.

Critical Paths Given a placement $\boldsymbol{v}(\sigma)$, we consider horizontal and vertical critical paths as described in Sect. 2.4. It is often attempted (e.g., see Sect. 2.4 and [31]) to restrict I_i to be in a critical path, while I_j can be any. We call the resulting neighborhoods like *single-swap critical neighborhood, swap* critical neighborhood* and so forth.

In handling soft rectangles, a placement $\boldsymbol{v}(\sigma)$ tends to have many critical paths, since each rectangle is adjusted so that it directly touches horizontally adjacent rectangles or vertically adjacent rectangles. As a result, the restriction to critical paths is not very effective in reducing the neighborhood size. To remedy this to some extent, we define the *single-swap lower-bounding critical neighborhood* by restricting I_i to be in a critical path and to satisfy $w_i = w_i^L$ if the critical path is horizontal (or $h_i = h_i^L$ if vertical), since such a rectangle I_i cannot be shrunk any further. Similar argument applies also to other types of neighborhoods, resulting in the *single-shift lower-bounding critical neighborhood* and others.

Computational Comparison of Neighborhoods To evaluate the power of the above neighborhoods, preliminary computational experiment was conducted in [30] for the following neighborhoods, abbreviated as

`Sg-shift`: Single-shift neighborhood,
`Db-shift`: Double-shift neighborhood,
`Sg-swap`: Single-swap neighborhood,
`Db-swap`: Double-swap neighborhood,
`SgCr-shift`: Single-shift critical neighborhood (similarly for `DbCr-shift`, `SgCr-swap`, `DbCr-swap`),
`swap*`: swap* neighborhood,
`SgLb-shift`: Single-shift lower-bounding critical neighborhood (similarly for `SgLb-swap` and `Lb-swap*`),

According to the computational results, `SgLb-swap` appears to be reasonably stable and gives good results in most cases. However, it still requires rather large computation time. To shorten its time, we further restrict rectangles I_i and I_j to be swapped to those which are lower bounding (i.e., $w_i = w_i^L$ or $h_i = h_i^L$ holds depending on the direction of the critical path) and are adjacent in some critical path. The resulting neighborhood is denoted as follows.

`SgAd-swap`: Single-swap adjacent lower-bounding critical neighborhood.

The quality of the solutions obtained by `SgAd-swap` is not good, but it consumes very little time compared with others.

Further Elaborations To reduce the neighborhood sizes further while maintaining high searching power, three more modifications were added.

The first idea is to look at a rectangle which belongs to both horizontal and vertical critical paths. We call such a rectangle as a *junction* rectangle. It is expected that removing a junction rectangle will break both the horizontal and vertical critical paths, and will have a large effect of changing the current placement. Thus we apply single-shift or double-swap operations to a junction rectangle I_i with any other rectangles I_j which are not junctions (in the case of double-swap we further restrict I_j to have a smaller area than I_i). We then apply these operations to all junction rectangles I_i. If an improvement is attained in this process, we immediately move to local search with `SgLb-swap` neighborhood for attaining further improvement. This cycle of "junction removals" and "local search with `SgLb-swap`" is repeated until no further improvement is attained. The resulting algorithms are denoted `Jc(Sg-shift)`+`SgLb-swap` or `Jc(Db-swap)`+`SgLb-swap`, respectively, depending on which operation is used to move the junction rectangle.

To improve the efficiency further, it was tried to replace the `SgLb-swap` neighborhood in the above iterations with `SgAd-swap`, which was defined at the end of the previous subsection. Using this neighborhood in place of `SgLb-swap`, we obtain algorithms `Jc(Sg-shift)`+`SgAd-swap` or `Jc(Db-swap)` +`SgAd-swap`.

The experiment (partially reported in [30]) shows that these four attain similar quality, but the last one `Jc(Db-swap)`+`SgAd-swap` consumes much less computation time than others.

The last idea is to make use of vacant areas existing in a given placement. To find some of such vacant areas by a simple computation, we use the following property. Let the current sequence pair σ satisfy $i \preceq_\sigma^x j$ and there is no k such that $i \preceq_\sigma^x k \preceq_\sigma^k j$ (i.e., i is immediately to the left of j). In this case, if $x_i + w_i < x_j$ holds, there is some vacant area between i and j. We pick up the largest one among such vacant areas, in the sense of maximizing $x_j - (x_i + w_i)$. Let i^* and j^* be the resulting pair. Then we apply `Sg-swap` operations on σ^+ between those i and j such that $i \in \sigma^+$ is located in distance at most 5 from i^* (forward or backward, i.e., $|\sigma_+^{-1}(i) - \sigma_+^{-1}(i^*)| \leq 5$), and $j \in \sigma^+$ is located in distance at most 5 from j^* (forward or backward).

This neighborhood is derived by horizontal argument. Analogous argument can also be applied vertically, and we consider the local search based on the resulting two types of neighborhoods, denoted `SgVc-swap`.

We conducted an extensive preliminary experiment to find a best combination of the above ideas. This is a difficult task, and we omit here the details (some of the computational results are reported in [30]). There was no clear winner, but we have chosen the following combined neighborhood as a reasonable candidate.

Neighborhood A: Neighborhood `Jc(Db-swap)`+`SgAd-swap` with the addition of neighborhood `SgVc-swap`.

In the experiment of the next section, the two neighborhoods in A are combined in the following manner: First apply local search with `Jc(Db-swap)`+ `SgAd-swap` until a locally optimal solution is obtained, and then improve it by local search with `SgVc-swap`. The best solution obtained is then output.

3.3 Computational Results

Benchmarks and Experiment The benchmarks[2] known as ami49 and rp100 were used. They involve 49 and 100 hard rectangles, respectively, whose widths and heights are denoted w_i^0 and h_i^0. These rectangles are then softened for our experiment. In the case of the perimeter minimization problem, we set the lower and upper bounds on widths and heights as follows.

$$\begin{aligned} w_i^L &= (1-e)w_i^0,\, w_i^U = (1+e)w_i^0 \\ h_i^L &= (1-e)h_i^0,\ h_i^U = (1+e)h_i^0, \end{aligned} \tag{24}$$

where e is a constant like $0.1, 0.2$, etc. The perimeter L_i in (13) of each rectangle I_i is set to $L_i = 2(w_i^0 + h_i^0)$.

For the area minimization problem, we first set the areas A_i in constraint (14) by $A_i = w_i^0 h_i^0$ for all i, the bounds on h_i as in (24), and then the bounds on w_i by

$$w_i^L = A_i/h_i^U \quad \text{and} \quad w_i^U = A_i/h_i^L. \tag{25}$$

The algorithm was coded in C language, and run on a PC using Pentium 4 CPU, whose clock is 2.60 GHz and memory size is 780 MB. The linear and convex programming problems are solved by a proprietary software package NUOPT[3] of Mathematical Systems Inc., where the linear programming is based on the simplex method and the convex programming is based on the line search method.

Perimeter Minimization The first set of instances of the perimeter minimization problem (22) are generated from ami49 by setting constants e in (24) to $e = 0.0,\ 0.1,\ 0.2,\ 0.3$, respectively. For each e, five runs are conducted from independent random initial solutions and average data are given in Table 2. The meaning of rows is as follows: Candidates: The number of solutions σ tested, Improvements: The number of improvements attained in LS, Time: CPU time in seconds, Density: Total area of rectangles over the area of container, $2(W + H)$: Objective values. Note that Density and $2(W + H)$ are given both the average and best values in five runs.

From these results we see that the local search could obtain reasonably good solutions, except for the case of $e = 0.0$ (i.e., all rectangles are hard).

[2] See the footnote in Sect. 2.1

[3] `http://www.msi.co.jp/english/`

As we reduced the neighborhood size to a great extent, in order to make the whole computation time acceptable, the resulting size appears not sufficient for handling hard rectangles.

Table 2. Perimeter minimization problem with different e

Benchmarks		$e = 0.0$	$e = 0.1$	$e = 0.2$	$e = 0.3$
ami49	Candidates	413.8	7877.6	12400.0	11472.2
	Improvements	34.6	153.0	218.8	236.2
	Time (secs)	28.0	641.8	1142.4	897.4
	Density (avg.)	61.3	92.6	97.1	97.6
	$2(W + H)$ (avg.)	30536.8	24692.6	23439.8	22803.8
	Density (best)	66.3	97.7	98.5	98.7
	$2(W + H)$ (best)	29288.0	23807.0	23373.6	23355.8

Table 3. Area minimization problem with fixed widths W^*

Benchmarks		$W^* = 800$	$W^* = 1000$	$W^* = 1200$
ami49	Candidates	4156.2	3200.6	2195.0
	Improvements	241.2	184.2	140.4
	Time (secs)	967.2	743.2	511.6
	Density(avg.)	98.8	95.9	91.9
	H (avg.)	8152.5	6744.5	5868.5
	Density(best)	99.5	99.5	98.5
	H (best)	8097.6	6479.5	5454.4

Area Minimization The area minimization problem (23) was solved for three different W^* and $e = 0.2$, where five runs from random initial solutions were again carried out. The results are shown in Table 3, where row H gives the objective values of this problem. Although, in this case, the convex programming problems $P_{\text{area}}(\sigma)$ are used instead of the linear programming problems, the computation time does not increase much, and very dense placements are obtained in most of the tested instances.

Finally, a large benchmark rp100 with 100 rectangles was tested. Table 4 gives the results of problems (22) and (23) with $e = 0.2$ and $W^* = 450$ (in the case of (23)). The obtained result for the area minimization is shown in Fig. 5. Considering that 2–3 hours were consumed for each run, it appears difficult to handle larger instances than these with this approach.

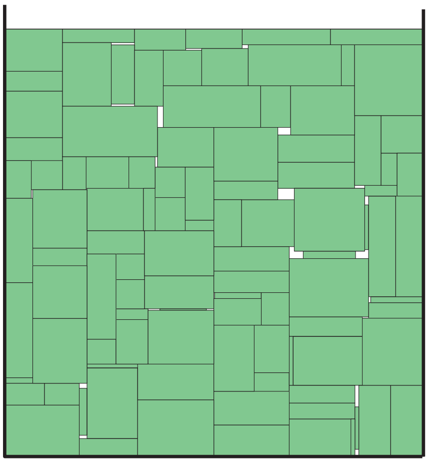


Fig. 5. Area minimization with rp100 ($W^* = 450$)

Table 4. Results with 100 rectangles

Benchmarks		Perimeter	Area
	Candidates	35012	15933
	Improvements	536	500
rp100	Time (secs)	7268.5	10101.5
	Density(%)	97.6	98.8
	Objectives	1777.6	461.4

4 Rectangles with Weights

The packing problem of rectangles with weights is also found in some applications, where each rectangle has weight d_i. In this case, the center of each rectangle is given by $(x_i + w_i/2,\ y_i + h_i/2)$, and constraints involving their center of gravity

$$(x^c, y^c) = \left(\frac{1}{D} \sum_i d_i(x_i + w_i/2),\ \frac{1}{D} \sum_i d_i(y_i + h_i/2) \right),$$

where $D = \sum_i d_i$, may be added. The objective function to minimize may be their kth moment (e.g., $k = 1$ or 2) around the center of gravity,

$$\sum_i d_i \left(\sqrt{(x_i + w_i/2 - x^c)^2 + (y_i + h_i/2 - y^c)^2} \right)^k .$$

This type of problem can also be handled in the local search framework as described in the previous section, in which the placement corresponding to a given sequence pair can be computed by nonlinear programming. An attempt in this direction is being made by [40]. Fig. 6 shows a solution obtained in their preliminary experiment, where the center of gravity is constrained to be the middle and the first moment is minimized. Note that darker rectangles represent heavier items.

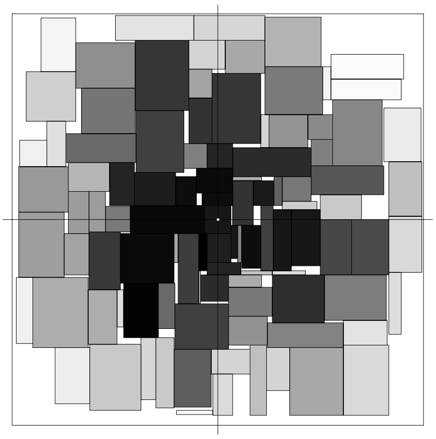


Fig. 6. Minimization of the first moment of rectangles with weights

5 Irregular Packing

In this section, we assume that given items are general polygons or arbitrary shapes that are not necessarily rectangular nor convex. Such problems are called *irregular packing* or *nesting* problems (or sometimes called *marker making*). Fig. 7 shows an example of packing nonconvex items for the benchmark known as "swim" (see Sect. 5.7). There are many practical applications,

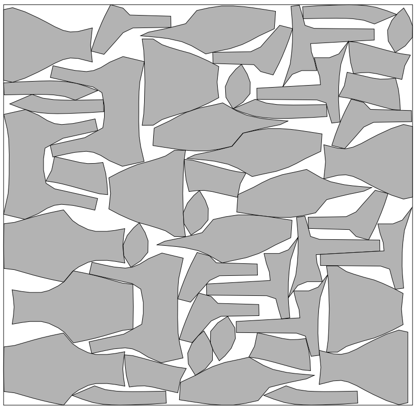


Fig. 7. An example of packing nonconvex polygons

e.g., in the garment, shoe and ship building industries, and many variants have been considered in the literature. Among them, the *irregular strip packing* problem has been extensively studied. Given n polygons and a rectangular container (i.e., a *strip*) with constant width W^* and variable height H, this problem asks to find a feasible placement of the given polygons into the strip so as to minimize the height H of the strip. As in the case of the rectangle packing problem, a placement is feasible if no polygon overlaps with any other polygon or protrudes from the strip. It has three variations with respect to the rotations of polygons: (1) rotations of any angles are allowed, (2) finite number of angles are allowed, and (3) no rotation is allowed. (Note that case (3) is a special case of (2) in which the number of given orientations of each polygon is one.) In this section, we mainly focus on case (3) for simplicity. In many practical applications such as textile industry, rotations are usually restricted to 180 degrees because textiles have the grain and may have a drawing pattern, while in such applications as glass, plastic etc., rotations of any angles are allowed. As mentioned in [28, 44], small rotations of any angles (e.g., a few degrees) in addition to 180 degrees are sometimes considered even for textiles in order to make the placement efficient.

A big difference of irregular packing from rectangle packing is that the intersection test between polygons is considerably more complex. Some heuristic algorithms use approximation of the given shapes, while many of the recent algorithms use a geometric technique called *no-fit polygons*, whose details will be explained in Sect. 5.2.

A standard way of designing heuristics is to put polygons one by one into the container according to a given sequence of all polygons. Most of the construction heuristics are based on this scheme, e.g., [3, 13, 52], which can be viewed as the irregular counterpart of the bottom-left heuristics for the rectangular case. A recent heuristic algorithm by Burke et al. [14] is quick and its solution quality is promising. It is also effective to apply local search (or metaheuristics) to find good sequences of polygons that lead to good placements when a construction algorithm is applied (i.e., a sequence is a coded solution and the construction algorithm is a decoding algorithm) [25].

Among many heuristic and metaheuristic algorithms, we focus our attention on hybrid approaches with mathematical programming. Such approaches seem to be effective in obtaining solutions of high quality especially when we have sufficient computation time.

In the following, we first define the irregular strip packing problem in Sect. 5.1, and then explain the idea of no-fit polygons in Sect. 5.2. In Sect. 5.3 we define some important subproblems, which are useful to solve the original packing problem efficiently. Then in Sects. 5.4 and 5.5 we describe how mathematical programming techniques are utilized for the irregular packing problems. In Sect. 5.6 we briefly summarize metaheuristic algorithms incorporated with mathematical programming techniques, and finally in Sect. 5.7 we give some computational results of recent hybrid metaheuristics.

5.1 Irregular Strip Packing Problem

We are given a set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of polygons, and a rectangular container (i.e., strip) $C = C(W^*, H)$ with a width $W^* \geq 0$ and a height H, where W^* is a constant and H is a nonnegative variable. We describe the location of a polygon P_i by the coordinate $\boldsymbol{v}_i = (x_i, y_i)$ of its *reference point*, where the reference point is any point of the polygon (e.g., a vertex of the polygon or the center of gravity; in the rectangular case, we set the bottom-left corner to be the reference point). The vector $\boldsymbol{v}_i = (x_i, y_i)$ is called the *translation vector* for P_i. For convenience, we regard each polygon P_i and the container C as the set of points (including both interior and boundary points), whose coordinates are determined from the reference point put at the origin $O = (0, 0)$. Then, we describe the polygon P_i placed at $\boldsymbol{v}_i$ by the Minkowski sum $P_i \oplus \boldsymbol{v}_i = \{\boldsymbol{p} + \boldsymbol{v}_i \mid \boldsymbol{p} \in P_i\}$. For a polygon S, let $\mathrm{int}(S)$ be the interior of S, ∂S be the boundary of S, $\bar{S}$ be the complement of S, and $\mathrm{cl}(S)$ be the closure of S (i.e., the smallest closed set containing S). Then the irregular strip packing problem (ISP) is formally described as follows.

$$\begin{array}{lll} \text{(ISP)} & \text{minimize} & H \\ & \text{subject to} & \mathrm{int}(P_i \oplus \boldsymbol{v}_i) \cap (P_j \oplus \boldsymbol{v}_j) = \emptyset, \quad 1 \leq i < j \leq n \\ & & (P_i \oplus \boldsymbol{v}_i) \subseteq C(W^*, H), \quad 1 \leq i \leq n \\ & & H \geq 0, \\ & & \boldsymbol{v}_i \in \mathbf{R}^2, \quad 1 \leq i \leq n. \end{array}$$

We represent a solution of problem (ISP) by an n-tuple $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$, which is the essential part of the decision variables because the minimum height H of the container is determined by

$$\begin{aligned} H_{\min}(\boldsymbol{v}) = {} & \max\{y \mid (x, y) \in P_i \oplus \boldsymbol{v}_i,\ P_i \in \mathcal{P}\} \\ & - \min\{y \mid (x, y) \in P_i \oplus \boldsymbol{v}_i,\ P_i \in \mathcal{P}\} \end{aligned}$$

and $(P_i \oplus \boldsymbol{v}_i) \subseteq C(W^*, H_{\min}(\boldsymbol{v}))$ holds for all $P_i \in \mathcal{P}$ if and only if

$$\begin{aligned} W^* \geq {} & \max\{x \mid (x, y) \in P_i \oplus \boldsymbol{v}_i,\ P_i \in \mathcal{P}\} \\ & - \min\{x \mid (x, y) \in P_i \oplus \boldsymbol{v}_i,\ P_i \in \mathcal{P}\}. \end{aligned}$$

5.2 Intersection Test and No-Fit Polygon

We consider in this section how to test the intersection between polygons. One popular idea for speeding up this test is to represent the polygons approximately. Some heuristic algorithms [5, 51] are based on *raster* (or bitmap) representation of the given polygons. Main drawback of this approach is that an appropriate choice of raster size is not easy. If the raster is rough, the intersection test is quick, but it will suffer from inaccuracy caused by the approximation inherent in the raster representation. On the other hand, if the raster is minute, the intersection test becomes expensive, and the memory space to keep the raster representation of polygons becomes huge.

Okano [50] proposed a technique that approximates polygons by a set of parallel line segments, which is called *scanline* representation. (His algorithm was designed for the two-dimensional bin packing problem, but it is applicable to various irregular packing problems including (ISP).) The number of scanlines is usually much smaller than that of pixels in a raster representation when the same resolution is required.

One of the most popular geometric techniques used for the intersection test is *no-fit polygon*. This concept was introduced by Art [4] in 1960s, who used the term “shape envelope” to describe the positions where two polygons can be placed without intersection. This technique is used in many algorithms for (ISP) [1, 3, 10, 25, 26, 52], where Albano and Sapuppo [3] seems to be the first who used the term “no-fit polygon.” This concept is also used for other problems such as robot motion planning and image analysis, and is called in various names such as Minkowski sums and configuration-space obstacle. Practical algorithms to calculate the no-fit polygon of two nonconvex polygons have been proposed, e.g., by Bennell et al. [11] and Ramkumar [54].

The no-fit polygon $\mathrm{NFP}(P_i, P_j)$ of an ordered pair of two polygons P_i and P_j is defined by

$$\mathrm{NFP}(P_i, P_j) = \mathrm{int}(P_i) \oplus (-\mathrm{int}(P_j)) = \{\boldsymbol{u} - \boldsymbol{w} \mid \boldsymbol{u} \in \mathrm{int}(P_i),\ \boldsymbol{w} \in \mathrm{int}(P_j)\}.$$

When the two polygons are clear from the context, we may simply use NFP instead of $\mathrm{NFP}(P_i, P_j)$. The no-fit polygon has the following important properties:

- $P_j \oplus \boldsymbol{v}_j$ overlaps with $P_i \oplus \boldsymbol{v}_i$ if and only if $\boldsymbol{v}_j - \boldsymbol{v}_i \in \mathrm{NFP}(P_i, P_j)$.
- $P_j \oplus \boldsymbol{v}_j$ touches $P_i \oplus \boldsymbol{v}_i$ if and only if $\boldsymbol{v}_j - \boldsymbol{v}_i \in \partial\mathrm{NFP}(P_i, P_j)$.
- $P_i \oplus \boldsymbol{v}_i$ and $P_j \oplus \boldsymbol{v}_j$ are separated if and only if $\boldsymbol{v}_j - \boldsymbol{v}_i \notin \mathrm{cl}(\mathrm{NFP}(P_i, P_j))$.

Hence the problem of checking whether two polygons overlap or not becomes an easier problem of checking whether a point is in a polygon or not. Fig. 8 shows an example of $\mathrm{NFP}(P_i, P_j)$.

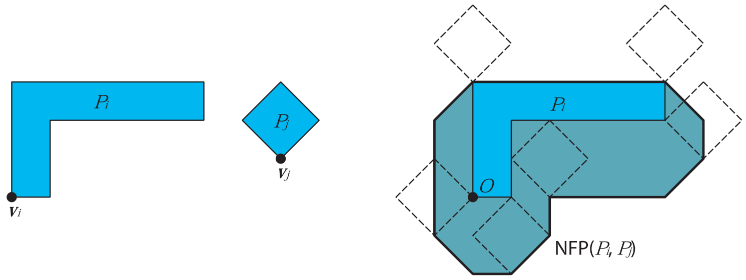


Fig. 8. An example of $\mathrm{NFP}(P_i, P_j)$, where O is the origin

When P_i and P_j are both convex, $\partial\mathrm{NFP}(P_i, P_j)$ can be computed by the following simple procedure: We first place the reference point of P_i at the origin $O = (0, 0)$, and slide P_j around P_i having it keep touching with P_i. Then the trace of the reference point of P_j is $\partial\mathrm{NFP}(P_i, P_j)$.

We can also check whether a polygon P_i protrudes from the container C or not similarly by using $\mathrm{NFP}(\bar{C}, P_i)$, which is the complement of a rectangle whose boundary is the trajectory of the reference point of P_i when we slide P_i inside C having it keep touching with C. (Gomes and Oliveira [25, 26] call $\overline{\mathrm{NFP}(\bar{C}, P_i)}$ *inner-fit rectangle.*)

5.3 Overlap Minimization, Compaction and Separation

In this section, we introduce three important subproblems of (ISP), *overlap minimization*, *translational compaction* and *separation* problems [34, 41]. Algorithms for these problems will be discussed in the next section.

Overlap Minimization Problem For this problem, infeasible placements that have overlap and/or protrusion are allowed, and the height H of the container C is a given constant (e.g., temporarily fixed in a heuristic algorithm). For a given placement $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$, let $f_{ij}(\boldsymbol{v})$ be a function that measures the amount of overlap of $P_i \oplus \boldsymbol{v}_i$ and $P_j \oplus \boldsymbol{v}_j$, and $g_i(\boldsymbol{v})$ be a function that measures the amount of protrusion of $P_i \oplus \boldsymbol{v}_i$ from the container $C(W^*, H)$. Then the objective of this problem is to find a placement $\boldsymbol{v} \in \mathbf{R}^{2n}$ that minimizes the total amount of overlap and protrusion

$$F(\boldsymbol{v}) = \sum_{1 \le i < j \le n} f_{ij}(\boldsymbol{v}) + \sum_{1 \le i \le n} g_i(\boldsymbol{v}).$$

It is not hard to see that this problem is NP-hard.

Translational Compaction Problem This problem is formulated as a two-dimensional motion planning problem. We are given a feasible placement $\boldsymbol{v}$ (i.e., $\boldsymbol{v}$ has no overlap or protrusion). The polygons and the container can move (translate) simultaneously, and the height H of the container can change. During a legal motion, the polygons cannot overlap each other nor protrude from the container. The objective is to minimize the height H of the container. See an example in Fig. 9.

Li and Milenkovic [41] showed that this problem is PSPACE-hard. They also considered more general formulation, and mentioned different possibilities of utilizing this problem; e.g., to make a big hole in the given placement by moving polygons away from a given point.

Translational Separation Problem We are given an infeasible placement $\boldsymbol{v}$ (i.e., some polygons overlap and/or protrude from the container). The problem is to find a set of translations of the polygons that eliminates all overlaps and protrusion while minimizing the total amount of translation.

Li and Milenkovic [41] showed that simple special cases of this problem are NP-hard.

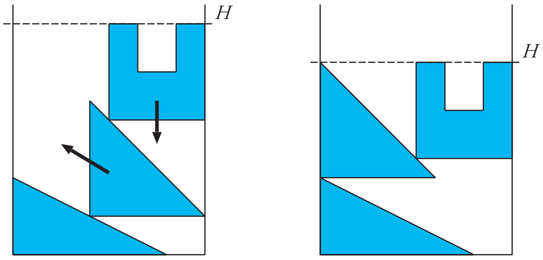


Fig. 9. An example of translational compaction

5.4 Nonlinear Programming Approach to Overlap Minimization

Imamichi et al. [34] considered the overlap minimization problem as an unconstrained nonlinear program, and incorporated a nonlinear programming technique in their heuristic algorithm. They defined the amount of overlap $f_{ij}(\boldsymbol{v})$ and protrusion $g_i(\boldsymbol{v})$ based on the following concept of *penetration depth*.

The penetration depth (also known as the intersection depth) is an important notion used in robotics, computer vision and so on [2, 19, 38]. The penetration depth $\delta(P_i \oplus \boldsymbol{v}_i, P_j \oplus \boldsymbol{v}_j)$ of two overlapping polygons P_i and P_j placed at $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, respectively, is defined to be the minimum translational distance to separate them. If two polygons do not overlap, their penetration depth is zero. The formal definition of the penetration depth is given by

$$\delta(P_i \oplus \boldsymbol{v}_i, P_j \oplus \boldsymbol{v}_j) = \min\{||\boldsymbol{u}|| \mid \mathrm{int}(P_i \oplus \boldsymbol{v}_i) \cap (P_j \oplus \boldsymbol{v}_j \oplus \boldsymbol{u}) = \emptyset, \boldsymbol{u} \in \mathbf{R}^2\},$$

where $||\cdot||$ denotes the Euclidean norm. In other words, this is the minimum distance from the point $\boldsymbol{v}_j - \boldsymbol{v}_i$ to the boundary $\partial\mathrm{NFP}(P_i, P_j)$ of the no-fit polygon. See an example in Fig. 10, where arrow $\boldsymbol{u}$ is the minimum translation vector that separates the two polygons.

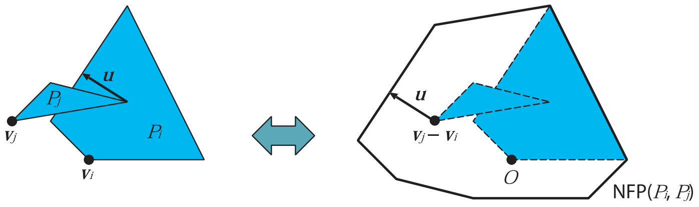


Fig. 10. The computation of penetration depth via no-fit polygon

Then the amounts of overlap and protrusion are defined by

$$\begin{aligned} f_{ij}(\boldsymbol{v}) &= \delta(P_i \oplus \boldsymbol{v}_i, P_j \oplus \boldsymbol{v}_j)^a, \quad && 1 \le i < j \le n \\ g_i(\boldsymbol{v}) &= \delta(\mathrm{cl}(\bar{C}), P_i \oplus \boldsymbol{v}_i)^a, && 1 \le i \le n, \end{aligned}$$

where $a > 0$ is a parameter. These functions are continuous, and values $f_{ij}(\boldsymbol{v})$ and $g_i(\boldsymbol{v})$ as well as $\nabla f_{ij}(\boldsymbol{v})$ and $\nabla g_i(\boldsymbol{v})$ for any placement $\boldsymbol{v}$ can be computed efficiently by using no-fit polygons. They set $a = 2$ for the following two reasons:

- At the boundary of no-fit polygons (i.e., when two polygons touch each other), $f_{ij}(\boldsymbol{v})$ and $g_i(\boldsymbol{v})$ are differentiable if and only if $a > 1$.
- The formulae of $\nabla f_{ij}(\boldsymbol{v})$ and $\nabla g_i(\boldsymbol{v})$ become the simplest when $a = 2$.

Then the problem becomes an unconstrained quadratic programming problem (e.g., [12]), for which many efficient algorithms for finding locally optimal solutions exist; e.g., quasi-Newton method, conjugate gradient method, etc.

5.5 Linear Programming Approach to Translational Compaction and Separation

Li and Milenkovic [41] proposed linear programming (LP) approaches for translational compaction and separation problems. Similar ideas are also used in [10, 26, 55]. We first explain their method for compaction.

The main idea is to restrict the search to a convex subregion of the original problem, which is realized by adding artificial linear constraints, in order to apply linear programming methods. They call the heuristic rules to add such constraints *locality heuristics*. The subregion should contain the given placement, and a larger region is preferable.

Let $\boldsymbol{v}^{(0)}$ be the given placement and $\Delta\boldsymbol{v}$ be the translation added to $\boldsymbol{v}^{(0)}$ (i.e., $\boldsymbol{v}^{(0)}$ is the given constant, $\Delta\boldsymbol{v}$ is the decision variable, and $\boldsymbol{v}^{(0)} + \Delta\boldsymbol{v}$ gives the modified placement). Among the constraints of problem (ISP), only the first one

$$\mathrm{int}(P_i \oplus (\boldsymbol{v}_i^{(0)} + \Delta\boldsymbol{v}_i)) \cap (P_j \oplus (\boldsymbol{v}_j^{(0)} + \Delta\boldsymbol{v}_j)) = \emptyset, \quad 1 \le i < j \le n,$$

which is equivalent to

$$(\boldsymbol{v}_j^{(0)} + \Delta\boldsymbol{v}_j) - (\boldsymbol{v}_i^{(0)} + \Delta\boldsymbol{v}_i) \notin \mathrm{NFP}(P_i, P_j), \quad 1 \le i < j \le n,$$

is nonconvex, and others are convex linear. The objective of the locality heuristics is to define, for each pair of P_i and P_j, a subset $S_{ij}(\boldsymbol{v}^{(0)}) \subseteq \overline{\mathrm{NFP}(P_i, P_j)}$ that has the following properties: (1) convex, (2) large, and (3) contains $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)}$. Then, for such subsets $S_{ij}(\boldsymbol{v}^{(0)})$, it is not hard to see that the following problem is a linear programming problem:

$$\begin{array}{lll}
\text{minimize} & H & \\
\text{subject to} & (\boldsymbol{v}_j^{(0)} + \Delta\boldsymbol{v}_j) - (\boldsymbol{v}_i^{(0)} + \Delta\boldsymbol{v}_i) \in S_{ij}(\boldsymbol{v}^{(0)}), & 1 \le i < j \le n \\
 & (P_i \oplus (\boldsymbol{v}_i^{(0)} + \Delta\boldsymbol{v}_i)) \subseteq C(W^*, H), & 1 \le i \le n \\
 & H \ge 0, & \\
 & \Delta\boldsymbol{v}_i \in \mathbf{R}^2, & 1 \le i \le n.
\end{array}$$

Note that Li and Milenkovic [41] used a different objective function $\sum_{P_i \in \mathcal{P}} \boldsymbol{d}_i \boldsymbol{v}_i$ in their LP formulation, where $\boldsymbol{d}_i$ is a desirable direction to move each polygon P_i, and the above formulation is due to [10, 26].

Fig. 11 illustrates a subregion outside the no-fit polygon. The left figure shows the given placement $\boldsymbol{v}_i^{(0)}$ and $\boldsymbol{v}_j^{(0)}$ of two polygons P_i and P_j, while the right figure shows the corresponding position of $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)}$ against no-fit polygon $\mathrm{NFP}(P_i, P_j)$, and a convex set $S_{ij}(\boldsymbol{v}^{(0)}) \subseteq \overline{\mathrm{NFP}(P_i, P_j)}$ that satisfies $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)} \in S_{ij}(\boldsymbol{v}^{(0)})$.

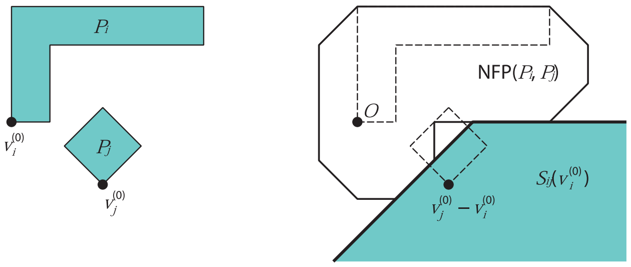


Fig. 11. A convex subregion $S_{ij}(\boldsymbol{v}^{(0)})$ outside $\mathrm{NFP}(P_i, P_j)$

The locality heuristic procedure in [41] defines the boundary of $S_{ij}(\boldsymbol{v}^{(0)})$ as follows. Starting from an edge of $\mathrm{NFP}(P_i, P_j)$ close to[4] $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)}$, it walks along $\partial\mathrm{NFP}(P_i, P_j)$ in both clockwise and counterclockwise directions (viewed from the origin), and the trace of the walk becomes the boundary $\partial S_{ij}(\boldsymbol{v}^{(0)})$. When walking clockwise, $\partial S_{ij}(\boldsymbol{v}^{(0)})$ should make only left turns to keep $S_{ij}(\boldsymbol{v}^{(0)})$ convex. If the next edge of NFP turns to the left (i.e., a concave vertex of NFP), the walk follows it; otherwise, the walk extends the current edge until it intersects with NFP. This procedure continues until it can extend the current edge infinitely. The walk to the counterclockwise direction is similar. The resulting $S_{ij}(\boldsymbol{v}^{(0)})$ can be different if the starting edge is different. Gomes and Oliveira [26] suggest to choose a convex subregion whose closest edge from $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)}$ is most distant. Intuitively, this rule has an effect of making more margin for compaction.

Given an optimal solution $\Delta\boldsymbol{v}^*$ to the above LP problem, we can repeat the same procedure from the new placement $\boldsymbol{v}^{(1)} = \boldsymbol{v}^{(0)} + \Delta\boldsymbol{v}^*$. Hence we can generate a sequence of improved placements $\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots$ until the objective values of $\boldsymbol{v}^{(k)}$ and $\boldsymbol{v}^{(k+1)}$ coincide. It is not hard to observe that any convex combination of two consecutive placements $t\boldsymbol{v}^{(l)} + (1-t)\boldsymbol{v}^{(l+1)}$ $(0 \leq$

[4] Li and Milenkovic [41] consider a special type of polygons called star-shaped, and define the origins of no-fit polygons accordingly. Then the edge of the no-fit polygon that crosses the line segment from the origin to the point $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)}$ is chosen as the starting edge. This is not necessarily the closest edge.

$t \leq 1,\ l = 0, 1, \ldots, k$) is feasible. Hence such a sequence of placements gives a (piecewise linear) legal motion to the translational compaction problem.

Here it should be noted that having the constraints $(\boldsymbol{v}_j^{(0)} + \Delta\boldsymbol{v}_j) - (\boldsymbol{v}_i^{(0)} + \Delta\boldsymbol{v}_i) \in S_{ij}(\boldsymbol{v}^{(0)})$ for all pairs of polygons is not necessary in practice, and is time consuming. Hence such constraints are usually imposed only for relatively close pairs in the current placement [10, 41].

Similar technique is applicable to the translational separation problem. For a given infeasible placement $\boldsymbol{v}^{(0)}$, we can similarly define a convex subregion $S_{ij}(\boldsymbol{v}^{(0)})$ even for overlapping polygons; e.g., by making a walk starting from an edge close to $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)}$, where the constraint $\boldsymbol{v}_j^{(0)} - \boldsymbol{v}_i^{(0)} \in S_{ij}(\boldsymbol{v}^{(0)})$ cannot hold in this case. Then the objective of the resulting LP problem, whose constraints are the same as the LP model for compaction, is to find a feasible placement that is close to $\boldsymbol{v}^{(0)}$.

If the above LP problem is infeasible, then no feasible placement is found. To deal with such situations, Bennell and Dowsland [10] relax the violated constraints slightly and solve the modified LP problem again; then repeat a limited number of such steps. Gomes and Oliveira [26] consider a modified formulation in which the objective is the sum of the penalties on violating constraints $(\boldsymbol{v}_j^{(0)} + \Delta\boldsymbol{v}_j) - (\boldsymbol{v}_i^{(0)} + \Delta\boldsymbol{v}_i) \in S_{ij}(\boldsymbol{v}^{(0)})$. This can be regarded as an algorithm for a variant of the overlap minimization problem.

5.6 Hybrid Approaches

In this section, we summarize hybrid metaheuristic approaches for the irregular strip packing problem.

Imamichi et al. [34] proposed an iterated local search (ILS) algorithm, in which the nonlinear programming technique in Sect. 5.4 is incorporated. (The framework of ILS was explained in Sect. 2.3.) The core part of their ILS is the algorithm for the overlap minimization problem. They fix the height H of the container temporarily, and solve the overlap minimization problem. If a feasible placement is found (resp., not found), they reduce (resp., increase) H slightly, and solve the overlap minimization problem again. Such iterations are repeated until some stopping criterion is met. They used the quasi-Newton method for the nonlinear programming formulation of the overlap minimization problem, explained in Sect. 5.4. Given an initial solution (to the overlap minimization problem), the quasi-Newton method is applied, which can be viewed as local search since it iteratively improves the current solution by applying slight modifications to it until a locally optimal solution is found. This local search is iterated from different initial solutions, and the entire algorithm is regarded as ILS. The perturbation for generating the next initial solution is realized by a swap of the positions of two polygons, which are found by a sophisticated algorithm based on no-fit polygons under the condition that other polygons do not move.

There are other metaheuristic algorithms based on different formulations of the overlap minimization problem. Umetani et al. [58] define the penalty of two overlapping polygons to be the minimum translational distance in a specified direction (e.g., horizontal or vertical direction) to separate them. Egeblad et al. [22] define the penalty of two overlapping polygons to be their overlapping area. In these papers, they devise efficient algorithms to find the best position of a polygon when it is translated to a specified direction, and use them to define neighborhood operations. Such efficient neighborhood search algorithms are then incorporated in the guided local search framework in both papers. They can be regarded as hybridization of metaheuristics and algorithmic techniques of computational geometry.

Compaction approaches can modify given feasible placements to more efficient ones, but cannot perform wider search. This limitation is shown in [8], which finds that the compaction of randomly generated solutions cannot compete with local search. Thus, it seems meaningful to combine metaheuristics with compaction/separation algorithms.

Bennell and Dowsland [10] proposed a hybrid approach of separation/compaction and tabu thresholding algorithm, whose original version without hybridization was given in [9]. Their algorithm basically deals with the overlap minimization problem whose objective function is similar to [58]. Their neighborhood operation is to move a polygon to another position in the container. Tabu thresholding [23] is a variant of tabu search. It consists of two phases called the improving phase and the mixed phase, and these phases are alternately repeated. The improving phase is a standard local search, while nonimproving moves are allowed in the mixed phase. In the mixed phase, the neighborhood is divided into subareas. In each move, one of the subareas is chosen and the best neighbor in it is chosen even if it is worse than the current solution. They apply separation and compaction for the locally optimal solutions obtained after improving phases, but they limit the application of separation/compaction only for promising solutions because it is computationally expensive.

Gomes and Oliveira [26] proposed a hybrid approach of separation/compaction with simulated annealing. They limit the search space to feasible placements, and allow infeasible placements only when trial solutions are generated by the neighborhood operation, where they adopt the swap neighborhood (i.e., the positions of the two polygons are exchanged). Whenever a trial solution is generated, separation and compaction algorithms are applied, and the new placement is evaluated by the height H of the container if it is feasible, and is discarded if separation fails.

5.7 Computational Results

We briefly report some computational results of algorithms which give high quality solutions. They are (1) the iterated local search incorporated with quasi-Newton method (denoted as ILSQN) proposed in [34], (2) the simulated annealing incorporated with separation and compaction (denoted as SAHA) proposed in [26], and (3) the guided local search (denoted as 2DNest) proposed in [22]. The instances are available from the ESICUP web site.[5] For these instances, rotations of fixed degrees are allowed.

Table 5. Comparison of three algorithms for the irregular strip packing problem

instance	NDP	TNP	ANV	ILSQN		SAHA		2DNest	
				EF(%)	time(s)	EF(%)	time(s)	EF(%)	time(s)
ALBANO	8	24	7.25	*88.16	1200	87.43	2257	87.44	600
DAGLI	10	30	6.30	*87.40	1200	87.15	5110	85.98	600
DIGHE1	16	16	3.87	99.89	600	*100.00	83	99.86	600
DIGHE2	10	10	4.70	99.99	600	*100.00	22	99.95	600
FU	12	12	3.58	90.67	600	90.96	296	*91.84	600
JAKOBS1	25	25	5.60	86.89	600	†78.89	332	*89.07	600
JAKOBS2	25	25	5.36	*82.51	600	77.28	454	80.41	600
MAO	9	20	9.22	83.44	1200	82.54	8245	*85.15	600
MARQUES	8	24	7.37	89.03	1200	88.14	7507	*89.17	600
SHAPES0	4	43	8.75	*68.44	1200	66.50	3914	67.09	600
SHAPES1	4	43	8.75	*73.84	1200	71.25	10314	*73.84	600
SHAPES2	7	28	6.29	*84.25	1200	83.60	2136	81.21	600
SHIRTS	8	99	6.63	*88.78	1200	86.79	10391	86.33	600
SWIM	10	48	21.90	*75.29	1200	74.37	6937	71.53	600
TROUSERS	17	64	5.06	89.79	1200	*89.96	8588	89.84	600
CPU					Xeon		Pentium4		Pentium4
clock freq.					2.8 GHz		2.4 GHz		3.0 GHz
#runs					10		20		20

Table 5 shows their efficiency in % and computation time in seconds, where the efficiency of a solution is measured by the ratio $\sum_{P_i \in \mathcal{P}}(\text{area of } P_i)/W^*H$, which is shown in column "EF." The results were taken from their original papers unless otherwise stated. (The results of ILSQN are those reported in the technical report version of [34].) Note that the stopping criteria of these algorithms are different; column "time" shows the time limit of each run for ILSQN and 2DNest, while it shows the average computation time of all runs for SAHA. Column NDP, TNP and ANV show the number of different polygons, the total number of polygons, and the average number of vertices of different polygons, respectively. The value with "†" has been corrected from the one reported in [26] according to the information sent from the authors. The best results among the three algorithms are marked with "$*$." From the table, we can observe that ILSQN is somewhat better than the others.

[5] http://paginas.fe.up.pt/~esicup/

References

1. Adamowicz M, Albano A (1976) Nesting two-dimensional shapes in rectangular modules, Computer-Aided Design 8:27-33
2. Agarwal PK, Guibas LJ, Har-Peled S, Rabinovitch A, Sharir M (2000) Penetration depth of two convex polytopes in 3D, Nordic Journal of Computing 7:227–240
3. Albano A, Sapuppo G (1980) Optimal allocation of two-dimensional irregular shapes using heuristic search methods, IEEE Transactions on Systems, Man and Cybernetics 10:242–248
4. Art Jr. RC (1966) An approach to the two-dimensional irregular cutting stock problem, Technical Report 36-Y08, IBM Cambridge Science Center
5. Babu AR, Babu NR (2001) A genetic approach for nesting of 2-D parts in 2-D sheets using genetic and heuristic algorithms, Computer-Aided Design 33:879–891
6. Baker BS, Coffman Jr. EG, Rivest RL (1980) Orthogonal packing in two dimensions, SIAM Journal on Computing 9:846–855
7. Beltrán JD, Calderón JE, Cabrera RJ, Pérez JAM, Moreno-Vega JM (2004) GRASP/VNS hybrid for the strip packing problem, In: Proceedings of the First International Workshop on Hybrid Meta-heuristics (HM04), 79–90
8. Bennell JA (1998) Incorporating problem specific knowledge into a local search framework for the irregular shape packing problem, Ph.D. thesis, European Business Management School, University of Wales, Swansea
9. Bennell JA, Dowsland KA (1999) A tabu thresholding implementation for the irregular stock cutting problem, International Journal of Production Research 37:4259–4275
10. Bennell JA, Dowsland KA (2001) Hybridising tabu search with optimisation techniques for irregular stock cutting, Management Science 47:1160–1172
11. Bennell JA, Dowsland KA, Dowsland WB (2001) The irregular cutting-stock problem—a new procedure for deriving the no-fit polygon, Computers & Operations Research 28:271–287
12. Bertsekas DP (1999) Nonlinear Programming (2nd edition), Athena Scientific.
13. Błażewicz J, Hawryluk P, Walkowiak R (1993) Using a tabu search for solving the two-dimensional irregular cutting problem, Annals of Operations Research 41:313–325
14. Burke E, Hellier R, Kendall G, Whitwell G (2006) A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem, Operations Research 54:587–601
15. Burke EK, Kendall G, Whitwell G (2004) A new placement heuristic for the orthogonal stock-cutting problem, Operations Research 52: 655–671
16. Chang YC, Chang YW, Wu GM, Wu SW (2000) B*-trees: a new representation for non-slicing floorplans, In: Proceedings of the 37th Design Automation Conference, 458–463
17. Chu CCN, Young EFY (2004) Nonrectangular shaping and sizing of soft modules for floorplan-design improvement, IEEE Transactions Computer Aided Design of Integrated Circuits and Systems 23:71–79
18. Coffman Jr. EG, Garey MR, Johnson DS, Tarjan RE (1980) Performance bounds for level-oriented two-dimensional packing algorithms, SIAM Journal on Computing 9:801–826

19. Dobkin D, Hershberger J, Kirkpatrick D, Suri S (1993) Computing the intersection-depth of polyhedra, Algorithmica 9:518–533
20. Dréo J, Pétrowski JDA, Siarry P, Taillard E (2006) Metaheuristics for Hard Optimization, Springer.
21. Dyckhoff H (1990) A typology of cutting and packing problems, European Journal of Operational Research 44:145–159
22. Egeblad J, Nielsen BK, Odgaard A (2007) Fast neighborhood search for two- and three-dimensional nesting problems, European Journal of Operational Research 183:1249–1266
23. Glover F (1995) Tabu thresholding: Improved search by nonmonotonic trajectories, ORSA Journal on Computing 7:426–442
24. Glover FW, Kochenberge GA (eds) (2003) Handbook of Metaheuristics, Springer.
25. Gomes AM, Oliveira JF (2002) A 2-exchange heuristic for nesting problems, European Journal of Operational Research 141:359–370
26. Gomes AM, Oliveira JF (2006) Solving irregular strip packing problems by hybridising simulated annealing and linear programming, European Journal of Operational Research 171:811–829
27. Guo PN, Takahashi T, Cheng CK, Yoshimura T (2001) Floorplanning using a tree representation, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 20:281–289
28. Heckmann R, Lengauer T (1995) A simulated annealing approach to the nesting problem in the textile manufacturing industry, Annals of Operations Research 57:103–133
29. Hopper E, Turton BCH (2001) An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, European Journal of Operational Research 128:34–57
30. Ibaraki T, Nakamura K (2006) Packing problems with soft rectangles, In: Almeida F, Blesa Aguilera MJ, Blum C, Vega JMM, Pérez MP, Roli A, Sampels M (eds) Hybrid Metaheuristics, Springer Lecture Notes on Computer Science 4030:13–27
31. Imahori S, Yagiura M, Ibaraki T (2003) Local search algorithms for the rectangle packing problem with general spatial costs, Mathematical Programming 97:543–569
32. Imahori S, Yagiura M, Ibaraki T (2005) Improved local search algorithms for the rectangle packing problem with general spatial costs, European Journal of Operational Research 167:48–67
33. Imahori S, Yagiura M, Ibaraki T (2005) Variable neighborhood search for the rectangle packing problem, In: Proceedings of the 6th Metaheuristics International Conference (MIC05), 532–537
34. Imamichi T, Yagiura M, Nagamochi H (2006) An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem, Technical Report 2007-009, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, February, 2007 (available at http://www.amp.i.kyoto-u.ac.jp/tecrep/); A short version is available in: Proceedings of the Third International Symposium on Scheduling (ISS06), 132–137
35. Itoga H, Kodama C, Fujiyoshi K (2005) A graph based soft module handling in floorplan, IEICE Transactions Fundamentals E88-A:3390–3397

36. Johnson DS (1990) Local optimization and the traveling salesman problem, In: Peterson MS (ed) Automata, Languages and Programming, Lecture Notes in Computer Science 443:446–461
37. Kenyon C, Rémila E (2000) A near-optimal solution to a two-dimensional cutting stock problem, Mathematics of Operations Research 25:645–656
38. Kim YJ, Lin MC, Manocha D (2004) Incremental penetration depth estimation between convex polytopes using dual-space expansion, IEEE Transactions on Visualization and Computer Graphics 10:152–163
39. Konno H, Kuno T (1995) Multiplicative programming problems, In: Horst R, Pardalos PM (eds) Handbook of Global Optimization, Kluwer Academic Publishers, 369–406
40. Kurebe Y, Miwa H, Ibaraki T (2007) Weighted module placement based on rectangle packing, 4th ESICUP meeting (EURO Special Interest Group on Cutting and Packing).
41. Li Z, Milenkovic V (1995) Compaction and separation algorithms for nonconvex polygons and their applications, European Journal of Operational Research 84:539–561
42. Lodi A, Martello S, Monaci M (2002) Two-dimensional packing problems: A survey, European Journal of Operational Research 141:241–252
43. Lodi A, Martello S, Vigo D (1999) Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11:345–357
44. Milenkovic VJ (1998) Rotational polygon overlap minimization and compaction, Computational Geometry 10:305–318
45. Murata H, Fujiyoshi K, Nakatake S, Kajitani Y (1996) VLSI module placement based on rectangle-packing by the sequence-pair, IEEE Transactions on Computer Aided Design 15:1518–1524
46. Murata H, Kuh ES (1998) Sequence-pair based placement method for hard/soft/preplaced modules, In: Proceedings of International Symposium on Physical Design, 167–172
47. Nagamochi H (2005) Packing soft rectangles, International Journal of Foundations of Computer Science 17:1165–1178
48. Nakatake S, Fujiyoshi K, Murata H, Kajitani Y (1998) Module packing based on the BSG-structure and IC layout applications, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 17:519–530
49. Nesterov Y, Nemirovskii A (1994) Interior Point Polynomial Algorithms in Convex Programming, SIAM Pub.
50. Okano H (2002) A scanline-based algorithm for the 2D free-form bin packing problem, Journal of the Operations Research Society of Japan 45:145–161
51. Oliveira JF, Ferreira JS (1993) Algorithms for nesting problems, In: Vidal RVV (ed) Applied Simulated Annealing. Lecture Notes in Economics and Mathematical Systems 396, Springer-Verlag, 255–274
52. Oliveira JF, Gomes AM, Ferreira JS (2000) TOPOS—a new constructive algorithm for nesting problems, OR Spektrum 22:263–284
53. Preas BT, van Cleemput WM (1979) Placement algorithms for arbitrarily shaped blocks, In: Proceedings of the ACM/IEEE Design Automation Conference, 474–480
54. Ramkumar GD (1996) An algorithm to compute the Minkowski sum outer-face of two simple polygons, In: Proceedings of the Twelfth Annual Symposium on Computational Geometry (SCG96), 234–241

55. Stoyan YG, Novozhilova MV, Kartashov AV (1996) Mathematical model and method of searching for a local extremum for the non-convex oriented polygons allocation problem, European Journal of Operational Research 92:193–210
56. Takahashi T (1996) An algorithm for finding a maximum-weight decreasing sequence in a permutation, motivated by rectangle packing problem (in Japanese), Technical Report of IEICE VLD96-30, 31–35
57. Tang X, Tian R, Wong DF (2001) Fast evaluation of sequence pair in block placement by longest common subsequence computation, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 20:1406–1413
58. Umetani S, Yagiura M, Imamichi T, Imahori S, Nonobe K, Ibaraki T (2006) A guided local search algorithm based on a fast neighborhood search for the irregular strip packing problem, In: Proceedings of the Third International Symposium on Scheduling (ISS06), 126–131
59. Wäscher G, Haußner H, Schumann H (2007) An improved typology of cutting and packing problems, European Journal of Operational Research 183:1109–1130
60. Young FY, Chu CCN, Luk WL, Wong YC (2001) Handling soft modules in general nonslicing floorplan using Lagrangean relaxation, IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems 20: 687–692

Hybrid Metaheuristics for Multi-objective Combinatorial Optimization

Matthias Ehrgott and Xavier Gandibleux

Laboratoire d'Informatique de Nantes Atlantique FRE CNRS 2729
Université de Nantes, Nantes, France
{Matthias.Ehrgott,Xavier.Gandibleux}@univ-nantes.fr

Summary. Many real-world optimization problems can be modelled as combinatorial optimization problems. Often, these problems are characterized by their large size and the presence of multiple, conflicting objectives. Despite progress in solving multi-objective combinatorial optimization problems exactly, the large size often means that heuristics are required for their solution in acceptable time. Since the middle of the nineties the trend is towards heuristics that "pick and choose" elements from several of the established metaheuristic schemes. Such hybrid approximation techniques may even combine exact and heuristic approaches. In this chapter we give an overview over approximation methods in multi-objective combinatorial optimization. We briefly summarize "classical" metaheuristics and focus on recent approaches, where metaheuristics are hybridized and/or combined with exact methods.

1 Introduction

The last two or three decades have seen the development and the improvement of approximate solution methods – usually called *heuristics* and *metaheuristics*. The success of metaheuristics, e.g., simulated annealing, tabu search, genetic algorithms, on hard single objective optimization problems is well recognized today. Although combinatorial optimization models have been successfully used in a vast number of applications, these models often neglect the fact that many real-life problems require taking into account several conflicting points of view corresponding to multiple objectives. Here are some examples.

- In portfolio optimization risk and return are the criteria that have generally been considered. Recently the classical Markowitz model has been criticized and other criteria have been mentioned: ratings by agencies, dividend, long-term performance, etc., see, e.g., Ehrgott et al. [32].
- In airline operations, scheduling technical and cabin crew has a major effect on cost and small percentage improvements may translate to multi-million

M. Ehrgott and X. Gandibleux: *Hybrid Metaheuristics for Multi-objective Combinatorial Optimization*, Studies in Computational Intelligence (SCI) **114**, 221–259 (2008)
www.springerlink.com

Euro savings. However, cost is not the only concern in airline operations. Robust solutions are desired, which avoid the propagation of delays due to crew changing aircraft, see Ehrgott and Ryan [33].

- In railway transportation, the planning of railway network infrastructure capacity has the goals of maximizing the number of trains that can use the infrastructure element (e.g. a station) and to maximize robustness of the solution to disruptions in operation. This problem can be modelled as a large scale set packing problem with two objectives [20].
- In radiation therapy planning for cancer treatment conflicting goals are to achieve a high uniform dose in the tumour whereas the dose absorbed by healthy tissue is to be limited. For anatomical and physical reasons these objectives cannot be achieved simultaneously. A multi-criteria model is described in Hamacher and Küfer [62].
- In computer networks, internet traffic routing may be enhanced if based on a multi-objective routing procedure to prevent network congestion. Multi-objective shortest paths between one router and all the other routers of the network must be computed in real-time, by simultaneously optimizing linear objectives (cost, delay) and bottleneck ones (quality, bandwidth), see Randriamasy et al. [44, 107].

Multiple objective optimization differs from traditional single objective optimization in several ways.

- The usual meaning of the optimum makes no sense in the multiple objective case because a solution optimizing all objectives simultaneously does in general not exist. Instead, a search is launched for a feasible solution yielding the best compromise among objectives on a set of so-called efficient (Pareto optimal, non-dominated) solutions.
- The identification of a best compromise solution requires the preferences expressed by the decision maker to be taken into account.
- The multiple objectives encountered in real-life problems can often be expressed as mathematical functions of a variety of forms, i.e., not only do we deal with conflicting objectives, but with objectives of different structures.

1.1 Multi-objective Optimization

A multi-objective optimization problem is defined as

$$\min \{(z_1(x), \ldots, z_p(x)) : x \in X\}, \tag{MOP}$$

where $X \subset \mathbb{R}^n$ is a feasible set and $z : \mathbb{R}^n \to \mathbb{R}^p$ is a vector valued objective function. By $Y = z(X) := \{z(x) : x \in X\} \subset \mathbb{R}^p$ we denote the image of the feasible set in the objective space. We consider optimal solutions of (MOP) in the sense of efficiency, i.e., a feasible solution $x \in X$ is called efficient if there does not exist $x' \in X$ such that $z_k(x') \leqq z_k(x)$ for all $k = 1, \ldots, p$ and

$z_j(x') < z_j(x)$ for some j. In other words, no solution is at least as good as x for all objectives, and strictly better for at least one.

Efficiency refers to solutions x in decision space. In terms of the objective space, with objective vectors $z(x) \in \mathbb{R}^p$ we use the notion of non-dominance. If x is efficient then $z(x) = (z_1(x), \ldots, z_p(x))$ is called non-dominated. The set of efficient solutions is X_E, the set of non-dominated points is Y_N. We may also refer to Y_N as the non-dominated frontier. For $y^1, y^2 \in \mathbb{R}^p$ we shall use the notation $y^1 \leqq y$ if $y^1_k \leqq y^2_k$ for all $k = 1, \ldots, p$; $y^1 \leq y^2$ if $y^1 \leqq y^2$ and $y^1 \neq y^2$; and $y^1 < y^2$ if $y^1_k < y^2_k$ for all $k = 1, \ldots, p$. $\mathbb{R}^p_{\geqq}$ denotes the non-negative orthant $\{y \in \mathbb{R}^p : y \geqq 0\}$, $\mathbb{R}^p_{\geq}0$ and $\mathbb{R}^p_{>}0$ are defined analogously.

To solve a multi-objective optimization problem means to find the set of efficient solutions or, in case of multiple x mapping to the same non-dominated point, for each $y \in Y_N$ to find an $x \in X_E$ with $z(x) = y$. This concept of a set of efficient solutions is the major challenge of multi-objective optimization. Most methods require the repeated solution of single objective problems which are in some sense related to the multi-objective problem, see e.g., Miettinen [91] or Ehrgott and Wiecek [35]. Many references on the state-of-the-art in multi-objective optimization are found in [29].

1.2 Multi-objective Combinatorial Optimization

In this chapter we focus on *multi-objective combinatorial optimization* problems formulated as

$$\min \{Cx : Ax \geqq b, x \in \mathbb{Z}^n\}. \qquad \text{(MOCO)}$$

Here C is a $p \times n$ objective function matrix, where c^k denotes the k-th row of C. A is an $m \times n$ matrix of constraint coefficients and $b \in \mathbb{R}^m$. Usually the entries of C, A and b are integers. The feasible set $X = \{Ax \geqq b, x \in \mathbb{Z}^n\}$ may describe a combinatorial structure such as, e.g., spanning trees of a graph, paths, matchings etc. We shall assume that X is a finite set. By $Y = CX := \{Cx : x \in X\}$ we denote the image of X under C in $\mathbb{R}^p$, the feasible set in objective space.

MOCO has become a very active area of research since the 1990s as demonstrated in the bibliographies by Ehrgott and Gandibleux [27, 28].

The biggest additional challenge in solving MOCOs as compared to multi-objective linear programs (MOLPs) $\min\{Cx : Ax \geqq b, x \geqq 0\}$ results from the existence of efficient solutions which are *not* optimal for any scalarization using weighted sums

$$\min \left\{ \sum_{k=1}^{p} \lambda_k z_k(x) : x \in X \right\}, \qquad (1)$$

called non-supported efficient solutions X_{NE}. Those that are optimal for some weighted sum problem (1) are called supported efficient solutions X_{SE}.

Based on this distinction between supported and non-supported efficient solutions, the so-called two-phase method has first been developed by Ulungu and Teghem [130, 131]. The method computes supported (in Phase 1) and non-supported (in Phase 2) efficient solutions. In Phase 2 information obtained in Phase 1 is exploited. Both phases rely on efficient algorithms for the solution of single objective problems. The method has been applied to a number of problems such as network flow [82], spanning tree [106], assignment [104, 131] and knapsack problems [135].

Notably, all these applications are for bi-objective problems and the first generalization to three objectives is very recent. Przybylski, Gandibleux and Ehrgott [102, 103] use a decomposition of the weight set $\left\{\lambda \in \mathbb{R}^p_{>} : \mathbf{1}^T\lambda = 1\right\}$ (where $\mathbf{1}$ is a vector of ones) in Phase 1 and upper bound sets (see Sect. 2.1) and ranking algorithms in Phase 2 in an application to the three-objective assignment problem.

Many methods generalizing single objective algorithms for the use with multiple objectives have been developed, including dynamic programming [11] and branch and bound [88]. For more details we refer again to the bibliographies by Ehrgott and Gandibleux [27, 28] and references therein.

From a methodological point of view non-supported efficient solution are the main reason why the computation of X_E, respectively Y_N is hard, from a more theoretical point of view the computational complexity is another. Most MOCO problems are NP-hard (finding an efficient solution is hard) as well as #P-hard (counting efficient solutions is hard), see Ehrgott [26]. The best illustration of this fact is perhaps the unconstrained problem

$$\min\left\{\left(\sum_{i=1}^{n} c_i^1 x_i, \sum_{i=1}^{n} c_i^2 x_i\right) : x \in \{0,1\}^n\right\},$$

with $c_i^1, c_i^2 \geqq 0$ for $i = 1, \dots, n$ which is trivial with only one objective. Moreover, MOCO problems are often intractable, i.e., there may exist exponentially many non-dominated points and efficient solutions.

With increasing interest in multi-objective models for real world applications and the difficulty of solving multi-objective combinatorial optimization problems exactly, interest in approximate methods for solving MOP/MOCO problems arose. In this paper we present an overview of approximation methods for solving multi-objective combinatorial optimization problems, focussing on hybrid metaheuristics for multi-objective combinatorial optimization.

The chapter is organized as follows. Sect. 2 introduces approximation methods for MOCO problems. Bound sets and quality of approximation are discussed in some detail. In Sect. 3, we present classical evolutionary algorithms and neighbourhood search metaheuristics. New methods based on ant colony optimization and particle swarm optimization are also briefly presented. In Sect. 4 we develop the hybrid methods and organize the ideas in six classes.

2 Approximation Methods for MOCO

Approximation methods for multi-objective optimization include both approximation algorithms which have a guaranteed quality of approximation, i.e., polynomial time approximation algorithms, and *multiple objective (meta)heuristics*, *MO(M)H* for short. A MO(M)H is a method which finds either sets of locally potentially efficient solutions that are later merged to form a set of potentially efficient solutions – the approximation denoted by X_{PE} – or globally potentially efficient solutions according to the current approximation X_{PE}.

The interest in approximation methods for multi-objective optimization is relatively recent. The first polynomial time approximation algorithm with performance guarantee is due to Warburton for the shortest path problem [136]. In the last five years this field has been growing, and such algorithms for, e.g., knapsack [38], travelling salesman [3, 85] and scheduling problems [58] are now known. Papadimitriou and Yannakakis [96] give a general result on the existence of polynomial time approximation algorithms.

In this chapter, however, we concentrate on metaheuristics. The first MOMH algorithm is a genetic algorithm developed 1984 by Schaffer [109]. In 1992, the work of Serafini [111] started a stream of research on multiple objective extensions of local search based metaheuristics. While the first adaptation of metaheuristic techniques for the solution of multi-objective optimization problems has been introduced more than 20 years ago, the MO(M)H field has clearly mushroomed over the last ten years. The pioneer methods have three characteristics.

- They are inspired either by *evolutionary algorithms (EA)* or by *neighbourhood search algorithms (NSA)*.
- The early methods are direct derivations of single objective metaheuristics, incorporating small adaptations to integrate the concept of efficient solution for optimizing multiple objectives.
- Almost all methods were designed as a solution concept according to the principle of metaheuristics. In fact, problem specific heuristics that are so ubiquitous in combinatorial optimization are rare, even descriptions of local search procedures appeared only recently [30, 99]. Paquete and Stützle describe local search heuristics for the TSP and QAP in [97] and [98], respectively. [115] develops a local search for the optimization of mobile phone keymaps.

2.1 Bound Sets

The quality of a solution of a combinatorial optimization problem can be estimated by comparing lower and upper bounds on the optimal objective function value. In analogy to moving from the optimal value to a set of nondominated points, the concept of bounds has to be extended to bound sets in multi-objective optimization. It is clear that the ideal and nadir point y^I and y^N defined by

$$\begin{aligned} y^I_k &= \min\{z_k(x) : x \in X\} \text{ for } k = 1, \dots, p \text{ and} \\ y^N_k &= \max\{z_k(x) : x \in X_E\} \text{ for } k = 1, \dots, p, \end{aligned}$$

respectively, are lower/upper bounds for Y_N. We sometimes refer to a utopian point $y^U = y^I - \varepsilon\mathbf{1}$, where $\mathbf{1}$ is a vector of all ones and ε is a small positive number. However, the ideal and nadir points are usually far away from non-dominated points and do not provide a good estimate of the non-dominated set. In addition, the nadir point is hard to compute for problems with more than two objectives, see [34]. Ehrgott and Gandibleux [31] present definitions, some general procedures and report results on lower and upper bound sets for the bi-objective assignment, knapsack, travelling salesman, set covering, and set packing problems.

Fernández and Puerto [39] use bound sets in their exact and heuristic methods to solve the multi-objective uncapacitated facility location problem. Spanjaard and Sourd [116] use bound sets in a branch and bound algorithm for the bi-objective spanning tree problem.

2.2 The Quality of Approximation

MO(M)H algorithms do compute a (usually feasible) set of solutions to a multi-objective optimization problem. According to the definition of Ehrgott and Gandibleux [31] these define an upper bound set. But what is a *good* approximation of the non-dominated set of a MOP? This is an ongoing discussion in the literature and there is no sign of a consensus at this point in time (see Figs. 1 and 2).

Kim et al. [10] propose the integrated preference functional (IFP), which relies on a weight density function provided by the decision maker, to compare the quality of algorithms for MOCO problems with two objectives. Sayın [108] proposes the criteria of coverage, uniformity, and cardinality to measure how well subsets of the non-dominated set represent the whole non-dominated set. Although developed for continuous problems the ideas may be interesting for MOCO problems. However, the methods proposed in [108] can be efficiently implemented for linear problems only.

Viana and Pinho de Sousa [134] propose distance based measures and visual comparisons of the generated approximations. The latter are restricted to bi-objective problems. Jaszkiewicz [70] also distinguishes between cardinal and geometric quality measures. He gives further references and suggests preference-based evaluation of approximations of the non-dominated set using outperformance relations. Collette and Siarry [15] mention the proportion of X_{PE} among all solutions generated in one iteration, the variance of the distance between points in objective space, and a metric to measure the speed of convergence. They also talk about the aesthetic of solution sets in the bi-objective case. Tenfelde-Podehl [126] proposes volume based measures. The

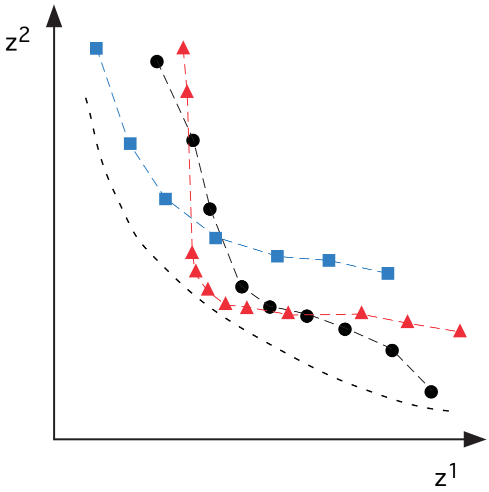


Fig. 1. Three approximations (bullet, triangle, square) and the non-dominated frontier (dotted line) are plotted. Square (better for z^1) and triangle (better for z^2) are complementary. Same conclusion for square and bullet. Triangle is better than bullet in the central part, and reversely. None of the three approximations is dominated. They are incomparable.

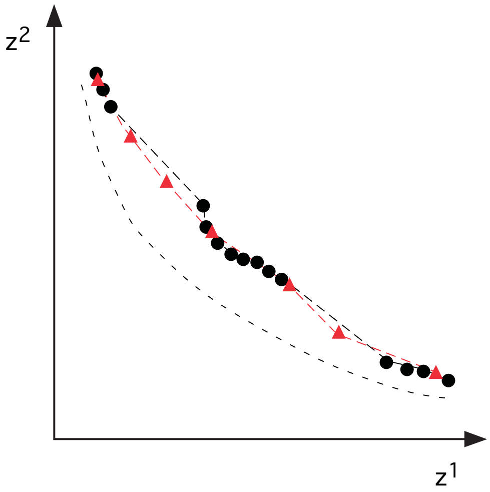


Fig. 2. Two approximations (bullet, triangle) and the non-dominated frontier (dotted line) are plotted. Density of points in the bullet approximation is higher than for the triangle one. However, points appear in clusters with the consequence of reporting gaps in the approximated frontier. Points in triangle approximation are better distributed along the frontier, but report a sparse approximated frontier. None of the two approximations dominates the other, they are incomparable.

hyper-volume measure introduced by Ziztler and Thiele [139] has become a popular means of comparing approximations because it has some important properties such as monotonicity with respect to set inclusion.

Zitzler et al. [140] present a review of performance measures, but none have been universally adopted in the multi-objective optimization literature, and further research is clearly needed.

3 Generation 1: Population Based Versus Neighbourhood Search MOMHs

3.1 Population Based Algorithms

Population based algorithms (see also Chap. 1 of this book) manage a "population" $\mathcal{P}$ of solutions rather than a single feasible solution. In general, they start with an initial population and combine principles of self adaptation, i.e., independent evolution (such as the mutation strategy in genetic algorithms), and cooperation, i.e., the exchange of information between individuals (such as the "pheromone" used in ant colony systems), to improve approximation quality. Because the whole population contributes to the evolutionary process, the generation mechanism is parallel along the non-dominated frontier, and thus these methods are also called *global convergence-based methods.* This characteristic makes population-based methods very attractive for solving multi-objective problems.

For a long time, the problems investigated with these methods were often unconstrained bi-objective problems with continuous variables and non-linear functions. Population based algorithms are appreciated by the engineering community, which could explain the large number of multi-objective evolutionary algorithm (MOEA) applications to real world problems (in mechanical design or electronics, for example). Many of these applications are characterized by long computation times for the evaluation of a single solutions. Surprisingly, MOCO problems have hardly been attacked by population based methods until recently.

3.2 The Pioneer: VEGA by Schaffer, 1984

In 1984 Schaffer [109, 110] introduced the *Vector Evaluated Genetic Algorithm (VEGA)*, which is an extension of Grefenstette's GENESIS program [60] to include multiple objective functions. The vector extension concerns only the selection procedure.

For each generation in VEGA, three stages are performed (Algorithm 17). The selection procedure is performed independently for each objective. In the first stage, the population is divided into p subpopulations S^k according to their performance in objective k (routine `pickIndividuals`). Each subpopulation is entrusted with the optimization of a single objective. In the next stage, subpopulations are shuffled to create a mixed population (routine `shuffle`). In the final stage, genetic operators, such as mutation and crossover, are applied to produce new potentially efficient individuals (routine `evolution`). This process is repeated for N_{gen} iterations.

Algorithm 17 VEGA, Vector Evaluated Genetic Algorithm

```
input:   $pop$,            the population size
         $N_{gen}$,          the generation limit
         $parameters$,     the crossover probability and mutation rate
output:  $X_{PE}$,          the set of potentially efficient solutions

begin VEGA
--| Generate an initial population of pop individuals
    $\mathcal{P}_0 \leftarrow$ initialization($pop$)
--| Generation process
    for n in $1, \ldots, N_{gen}$ loop
      --| 1. Elaborate $p$ subpopulations of size $pop/p$ using each objective $k$
      --| in turn
         $S^k \leftarrow$ pickIndividuals( $pop/p$ , $k$ , $\mathcal{P}_{n-1}$), $\forall k = 1, \ldots, p$
      --| 2. Set population of size pop shuffling together
      --| the $p$ subpopulations $S^k$
         $S \leftarrow$ shuffle($\cup_{k=1,\ldots,p} S^k$)
      --| 3. Apply genetic operators
         $\mathcal{P}_n \leftarrow$ evolution($S, parameters$)
    endLoop
    $X_{PE} \leftarrow \mathcal{P}_{N_{gen}}$
end VEGA
```

As VEGA selects individuals who excel in one performance dimension without looking at the other dimensions, the speciation problem can arise with that method. This implies that individuals with a balanced performance on all objectives will not survive under this selection mechanism. Speciation is undesirable because it is opposed to the generation of compromise solutions. Due to this characteristic VEGA is termed a non-Pareto approach [13]. Additional heuristics were developed (like crossbreeding among the species) and studied to overcome this tendency.

3.3 Other Evolutionary Algorithms

Since VEGA many MOEAs have been developed. Significant progress concerns corrections of shortcomings observed in the first algorithms introduced and propositions of new algorithmic primitives to generate a better approximation of X_E. Modern MOEAs are characterized according to the way they handle population structure, archiving, selection/elitism mechanisms, and fitness functions.

Two central questions motivate the research about MOEAs: (1) how to accomplish both fitness assignment and selection in order to guide the search toward the non-dominated frontier and (2) how to maintain a diversified population in order to avoid premature convergence and find a uniform distribution of points along the non-dominated frontier?

For the first question MOEAs are distinguished by the way the performance of individuals is evaluated in the selection. If the objectives are considered separately, the selection of individuals is performed by considering each objective independently (Schaffer [110]), or the selection is based on a comparison procedure according to a predefined (or random) order on the objectives (Fourman [42]), or the selection takes into account probabilities assigned to each objective in order to determine a predominant objective (Kursawe [78]). If the objectives are aggregated into a single parameterized objective function, the parameters of the function are systematically updated during the same runs (at random or using a particular weight combination) taking advantage of information collected on the population of individuals (Hajela and Lin [61] and Murata and Ishibuchi [94]). Each aggregation defines a search direction in the objective space and the idea is to optimize in multiple directions simultaneously. If, on the other hand, the concept of efficiency is directly used, (non-domination ranking) the fitness of an individual (i.e. a solution) is calculated on the basis of the dominance definition. The idea is to take advantage of information carried by the population of solutions using the notion of domination for selection. This is the most common approach and has led to several Pareto-based fitness assignment schemes, see [40, 55, 67, 117, 139], etc.

The majority of the other components of a MOEA deal with the second question. Fitness sharing based on a principle of niches is the most frequently used technique and most MOEAs are implementing it, e.g., [40, 67, 117, 138]. Niches are solution neighbourhoods with a radius σ_{sh} in objective space, centered on candidate points. Based on the number of solutions in these niches, the selection of individuals can be influenced to generate more in areas where niches are sparsely populated, with the goal of greater distribution uniformity along the non-dominated frontier. A *sharing function*, which measures the distance $d(i,j)$ between a candidate point i and a neighbour j, is defined by

$$\phi(d(i,j)) = 1 - \left(\frac{d(i,j)}{\sigma_{sh}}\right)^{\alpha}$$

if $d(i,j) < \sigma_{sh}$ and 0 otherwise. The parameter α amplifies ($\alpha > 1$) or attenuates ($\alpha < 1$) the sharing value computed. Thus the shared fitness f_{s_i} of candidate i is

$$f_{s_i} = \frac{f_i}{\sum_{j=1}^{N} \phi(d(i,j))}$$

such that the shared fitness of candidates increases the fitness f_i if ϕ values are small, i.e. the distance of neighbours from i is close to σ_{sh}.

A number of important implementations of MOEAs have been published in recent years. There are even a number of surveys on the topic (see [13, 14, 41, 72]). The most outstanding among the pioneer MOEAs are mentioned briefly below:

- Vector Evaluated Genetic Algorithm (VEGA) by Schaffer, 1984 [109].
- Multiple Objective Genetic Algorithm (MOGA93) by Fonseca and Fleming, 1993 [40]. MOGA93 uses a ranking procedure in which the rank of an individual is equal to the number of solutions which dominate this individual.
- Non-dominated Sorting Genetic Algorithm (NSGA) by Srinivas and Deb, 1994 [117]. NSGA implements Goldberg's ranking idea in which the rank of an individual is equal to its domination layer, computed by ranking the population on the basis of domination.
- Niched Pareto Genetic Algorithm (NPGA) by Horn, Nafpliotis and Goldberg, 1994 [67]. NPGA combines the Pareto dominance principle and a Pareto tournament selection where two competing individuals and a set of individuals are compared to determine the winner of the tournament.
- Strength Pareto Evolutionary Algorithm (SPEA) by Zitzler and Thiele, 1998 [138]. SPEA takes the best features of previous MOEAs and combines them to create a single algorithm. The multi-objective multi-constraint knapsack problem has been used as a benchmark to evaluate the method [139].
- Pareto Archived Evolution Strategy (PAES) by Knowles and Corne, 1999 [77]. PAES is an evolutionary strategy that employs local search to generate new candidate solutions and a reference archive to compute solution quality.

3.4 Other Population Based Methods

Ant colony optimization (ACO) based heuristics are population based methods imitating the foraging behaviour of ants. The principle is based on the positive reinforcement of elementary decisions via the use of pheromone trails, which are left by the ants. The cooperation between ants is accomplished by pheromone trails which act as a common memory of the colony.

Roughly speaking, in the case of a combinatorial optimisation problem, the pheromone trails induce a probability distribution over the search space. An ACO algorithm consists in an iterative process where, at each iteration the ants build solutions by exploiting the pheromone trails. Hence, they have the ability of learning "good and bad" decisions when building solutions. Once a solution is completed, pheromone trails are updated according to the quality of some of the generated solutions. Recently, ACO algorithms have been proposed for MOPs, introducing a novel stream of population based algorithms. We summarize some of the published papers.

- The MOAQ algorithm of Mariano and Morales [86, 87] uses one ant colony for each objective function. All colonies have the same number of ants. (Partial) solutions of each colony are used in the next colony. The algorithm is applied to two literature problems (and compared with VEGA [109]). The research is motivated by the real world problem of designing a water distribution network for irrigation to minimise network cost and maximise profit.
- McMullen [89] addresses a just-in-time sequencing problem with the objective of minimizing the number of setups and minimizing the usage rates of raw materials. The problem is reformulated as a TSP by spatialising the data and applying a standard single objective ant colony algorithm.
- Iredi and Middendorf [68] propose a number of ant colony optimization algorithms to solve bi-criteria combinatorial problems, including ones with single and multiple colonies. Various methods for pheromone update and weight assignment (in order to browse the whole non-dominated frontier) are proposed and tested on a single machine scheduling problem to minimise total tardiness and changeover cost. Numerical tests are presented.
- Doerner et al. [21, 22, 23, 24] work on a multi-objective portfolio selection problem. They consider a rather large number of objectives $p = (B+R)T$, where B is the number of benefit categories, R is the number of resources, and T is the planning period. They use one colony and a random selection of weights of objectives for each ant. The global update considers the best and second best solutions for each objective found in the current iteration. The results on test problems are compared with NSGA [117], PSA [16], and the true efficient set (for small problems).
- Doerner et al. [25] solve a special case of the pickup and delivery problem with the linearly combined objectives of total number of vehicles and empty vehicle movements by a multi-colony approach. The colonies use different heuristic information, and their sizes change during the algorithm.
- Gravel et al. [59] consider the problem of sequencing orders for the casting of aluminium. Four objectives are considered in a lexicographic sense. A distance function based on penalties for bad performance is used to translate the problem into a TSP setting. Global pheromone update considers only the primary objective.

Particle swarm optimization (PSO) is a population based metaheuristic inspired by the social behaviour of bird flocks and fish schools (swarms) searching for food. Each individual determines its velocity based on its experience and information obtained from interacting with other members. In the optimization context, the individual members of the swarm (particles) “travel” through solution space. Each particle is characterized by three vectors x, v and p, where x is the current position (solution), v is the velocity, and p is the best position occupied so far. In each iteration k, the position and velocity of the particles are updated: $v^{k+1} := f(v^k, p^k, x^k, \hat{p}^k)$ and $x^{k+1} = x^k + v^{k+1}$. The update function f depends not only on the particles own experience but also on $\hat{p}$, the best position of any particle so far, thus modelling the interaction in the swarms behaviour.

The PSO method has been introduced by Kennedy and Eberhart [76] in 1995 and applications in multi-objective optimization date back to [92]. Implementations usually use an archive of potentially efficient solutions. Very few applications in the MOCO area exist.

In [105] PSO is applied to a permutation flowshop problem to minimize weighted mean completion time and weighted mean tardiness. Yapicioglu et al. [137] use it to solve a bi-objective formulation of the semi-desirable facility location problem. The algorithm uses local search to improve solutions obtained by the basic PSO mechanism.

The *scatter search* principle has been proposed by Glover, Laguna, and Martí [54]. It uses a population of solutions called the reference set. The method forms combinations of solutions which are subsequently improved, before updating the reference set. A scatter search algorithm consists of

1. a diversification method to generate a diverse set of solutions,
2. an improvement method to be applied to solutions,
3. a reference set update method,
4. a subset generation method to select solutions from the reference set that is used for creating combinations,
5. and a solution combination method.

An adaptation to multi-objective optimization appeared shortly after the original method [8]. The only MOCO papers using scatter search we are aware of are [56] and [57], who apply it to the bi-objective knapsack problem. The former paper uses exact solutions of the LP relaxation as diversification, simple heuristics to obtain feasible and improved solutions, a clustering method for reference set update, selects pairs of consecutive solutions from the reference set, and uses path relinking for combining solutions. The latter is a hybrid method that improves some of the methods used in [56] and includes the exact solution of small residual problems in the improvement method.

3.5 Neighbourhood Search Algorithms

In *neighbourhood search algorithms (NSA)* the generation of solutions relies upon one individual, a current solution x^n, and its neighbours $x \in \mathcal{N}(x^n)$. Using a local aggregation mechanism for the objectives (often based on a weighted sum), a weight vector $\lambda \in \Lambda$, and an initial solution x^0, the procedure iteratively projects the neighbours into the objective space in a search direction λ by optimizing the corresponding parametric single objective problem. A local approximation of the non-dominated frontier is obtained using archives of the successive potentially efficient solutions detected. This generation mechanism is sequential along the frontier, producing a local convergence to the non-dominated frontier, and so such methods are called *local* *convergence-based methods*. The principle is repeated for diverse search directions to completely approximate the non-dominated frontier. NSAs are well-known for their ability to locate the non-dominated frontier, but they require more effort in diversification than EAs in order to cover the non-dominated frontier completely.

The first approximation methods proposed for MOCO problems were "pure" NSA strategies and were straightforward extensions of well-known metaheuristics for dealing with the notion of non-dominated points. Simulated annealing (the MOSA method [132]), tabu search (the MOTS method [46], the method of Sun [119]), or GRASP (the VO-GRASP method [52]) are examples.

3.6 The Pioneer: MOSA by Ulungu, 1992

In 1992 (EURO XII conference, Helsinki), Ulungu introduced Multi-objective Simulated Annealing, MOSA [130], a direct derivation of the simulated annealing principle for handling multiple objectives (see Algorithm 18). Starting from an initial, randomly generated solution x^0 and a neighbourhood structure $\mathcal{N}(x^n)$, MOSA computes a neighbour $x \in \mathcal{N}(\cdot)$ using a set of weights Λ that define search directions $\lambda \in \Lambda$. The comparison of x with x^n according to p objectives $z_k(x)$, $k = 1, \ldots, p$ gives rise to three possible cases. If $\Delta z_k = z_k(x) - z_k(x^n)$ is the difference between solution x and x^n in the objective k:

(a) $\Delta z_k \leqq 0$ for all k: x improves all objectives. x (weakly) dominates x^n.
(b) $\Delta z_k < 0$ and $\Delta z_{k'} > 0$ for some k and k': Improvement and deterioration occur simultaneously for different objectives. Both solutions x and x^n are potentially efficient.
(c) $\Delta z_k \geqq 0$ for all k: All objectives deteriorate with at least one strict inequality. Solution x is dominated by x^n.

A neighbour x is always accepted if it dominates x^n (a). When x is dominated (c), it can be accepted with decreasing probability, depending on the current "temperature" of the cooling schedule (Routine `isAccepted`). In the initial version of MOSA, a neighbour in situation (b) was also always accepted (Routine `isBetter`). This acceptance principle has been revised in a later version of the method to include the search direction in the decision.

To measure the degradation in the routine `isAccepted`, the values are aggregated using a scalarizing function $S(z(x), \lambda)$. Such a function makes a "local aggregation" of the objectives which allows the computation of the "weighted distance" $\Delta s = S(z(x), \lambda) - S(z(x^n), \lambda)$ between $z(x)$ and $z(x^n)$.

If a neighbour is accepted, the set of potentially efficient solutions $X_{PE\lambda}$ in direction λ is updated. The search stops after a certain number of iterations, or when a predetermined temperature is reached (Routine `isFinished`). At the end, MOSA combines the sequential processes in the objective space Y in a set X_{PE} by merging the sets PE_λ (Routine `merge`). The outline of MOSA for maximizing objectives is given in Algorithm 18.

Algorithm 18 MOSA, multi-objective Simulated Annealing

```
  input:   $\Lambda$,                                          set of weights
           $T, \alpha, N_{step}, T_{stop}, N_{stop}$,   SA parameters
  output:  $X_{PE}$,                                        set of potentially efficient solutions

begin MOSA
  $X_{PE} \leftarrow \emptyset$
  for all $\lambda \in \Lambda$ loop
    $T_0 \leftarrow T$ ;   $N_{count} \leftarrow 0$ ;   $n \leftarrow 0$
    randomly draw $x^n \in X$ ; $X_{PE\lambda} \leftarrow \{x^n\}$
    repeat      randomly draw $x \in \mathcal{N}(x^n)$
                if isBetter($x, x^n$) or else isAccepted($x, x^n, n, T_n, \lambda$) then
                   $X_{PE\lambda} \leftarrow$ archive($X_{PE\lambda}, x$);   $x^{n+1} \leftarrow x$ ;   $N_{count} \leftarrow 0$
                else
                   $x^{n+1} \leftarrow x^n$;   $N_{count} + +$
                endIf
                $n + +$ ;  updateParameters($\alpha, n, T_n$)
    until isFinished($N_{count}, T_n$)
  endLoop
  $X_{PE} \leftarrow$ merge($X_{PE\lambda}$)
end MOSA
```

3.7 Other Neighbourhood Search Methods

In 1996 (MOPGP 96 conference, Torremolinos), Gandibleux et al. introduced the first TS-based method called MOTS for multi-objective tabu search [46], designed to compute a set of potentially efficient solutions. Using a scalarizing function and a reference point, the method performs a series of tabu processes guided automatically in the objective space by the current approximation of the non-dominated frontier. Intensification, diversification and tabu daemon (usually called aspiration criteria) are designed for the multi-objective case. Two tabu lists are used, one on the decision space $TmemX$, the second on the objective space $TmemY$. The former is an attribute-based tabu list preventing a return to already visited solutions during a tabu process. The latter is related to the objectives and based on an improvement measure of each objective. It is used for updating weights between two consecutive tabu processes.

The MOTS search strategy is encapsulated in a *tabu process*, which is composed of a series of iterations. Let us consider, at the n^{th} iteration, the current solution x^n and its (sub)neighbourhood $\mathcal{N}(x^n)$ obtained according to a suitable move $x^n \to x$ defined according to the structure of the feasible set X (routine `exploreNeighbourhood`). The successor $\bar{x}$ of x^n for the next iteration is selected from the list of neighbour solutions $\mathcal{L} = \{x \in \mathcal{N}(x^n)\}$ as the best according to a weighted scalarizing function $S(z(x), y^U, \lambda)$

$$S(z(x), y^U, \lambda) = \max_{1 \leq k \leq p} \left\{ \lambda_k \left(y_k^U - z_k(x) \right) \right\} + \rho \sum_{k=1}^{p} \lambda_k \left(y_k^U - z_k(x) \right),$$

with $\rho > 0$. The number of candidates in list $\mathcal{L}$ is limited to K solutions. The value of this parameter depends on the neighbourhood size $(1 \leq K \leq |\mathcal{N}(x^n)|)$. The reference point y^U in the scalarizing function is the locally determined utopian point $y^U = (y_1^U, \ldots, y_p^U)$ over $\mathcal{L}$, where $y_k^U < \inf\{z_k(x) : x \in \mathcal{L}\}$. This point dominates the ideal point given by the lowest objective function value on each objective among the solutions in the neighbourhood of the current solution. The tabu list $TmemX$ is used to avoid cycling. The selected solution $\bar{x} \in \mathcal{L}$, which minimizes $S(z(x), y^U, \lambda)$ over $\mathcal{L}$ such that the move $x^n \to \bar{x}$ is not tabu, becomes the new current solution x^{n+1}. The *tabu daemon* overrides the tabu status of a solution $x' \in \mathcal{N}(x^n)$ if $s(z(x'), \lambda) \leq s(z(\bar{x}), \lambda) - \Delta$, with Δ being a static or dynamic threshold value. As $\mathcal{L}$ is generally a finite subset of X, the successor solution x^{n+1} can be found easily. However, the time complexity depends on the size of the neighbourhood $\mathcal{N}(x^n)$. Each iteration ends with the identification of the potentially efficient solutions in $\mathcal{L}$, which represents a local approximation of the non-dominated frontier (routine `archive`). More details about MOTS are available in [30, 46]. This is a generic method, rather than a ready-to-use technique. All of its primitives need to be stated in a suitable manner, according to the MOCO problem to be solved.

Few implementations have been developed in this stream. It can be explained by the fact that the advantage of NSA algorithms hybridized with other techniques has been recognized early, giving birth to the hybrid MOMH wave. Among the “pure NSA” algorithms discussed, we find:

- Multi-objective tabu search (MOTS) by Gandibleux, Mezdaoui and Fréville, 1996 [46].
- Sun's Method, 1997 [119]. An interactive procedure using tabu search for general multiple objective combinatorial optimization problems, the procedure is similar to the combined Tchebycheff/aspiration criterion vector method [118]. The tabu search is used to solve subproblems in order to find potentially efficient solutions. The principles used for designing the TS search strategy are similar to those defined for MOTS. This method has been used for facility location planning [1].
- Adaptations of metaheuristics, such as the greedy randomized adaptive search procedure GRASP [52].
- Other simulated annealing-based methods. Nam and Park's method, 2000 [95] is another simulated annealing-based method. The authors report

good results in comparison with MOEA. Applications include aircrew rostering problems [84], assembly line balancing problems with parallel workstations [90], and analogue filter tuning [127].

4 Generation 2: Hybrid MOMHs

The methods that followed the pioneer ones, designed to be more efficient algorithms in the MOCO context, have been influenced by two important observations.

The first observation is that on the one hand, NSAs focus on convergence to efficient solutions, but must be guided along the non-dominated frontier. On the other hand, EAs are very well able to maintain a population of solutions along the non-dominated frontier (in terms of diversity, coverage, etc.), but often converge too slowly to the non-dominated frontier. Naturally, methods have been proposed that try to take advantage of both EA and NSA features by combining components of both approaches, introducing *hybrid algorithms* for MOPs.

The second observation is that MOCO problems contain information deriving from their specific combinatorial structure, which can be advantageously exploited by the approximation process. Single objective combinatorial optimization is a very active field of research. Many combinatorial structures are very well understood. Thus combinatorial optimization represents a useful source of knowledge to be used in multi-objective optimization. This knowledge (e.g., cuts for reducing the search space) are more and more taken into account when designing a very efficient approximation method for a particular MOCO. It is not surprising to see an evolutionary algorithm – for global convergence – coupled with a tabu search algorithm – for the exploitation of the combinatorial structure – within one approximation method.

Modern MOMH's for MOCO problems appear more and more as problem-oriented techniques, i.e., selections of components that are advantageously combined to create an algorithm which can tackle the problem in the most efficient way. By nature the algorithm is hybrid, including evolutionary components, neighbourhood search components, and problem specific components.

This section gives an overview of the literature on hybrid MOMH's classified in three main categories: metaheuristic with metaheuristic (Sects. 4.1, 4.2 and 4.3), metaheuristic with other techniques (Sect. 4.4) and metaheuristic with preferences provided by a decision-maker (Sect. 4.5). An overview of the trends in the design of hybrid multi-objective metaheuristics closes this section (Sect. 4.6).

4.1 Hybridization to Make a Method More Aggressive

To enhance the aggressiveness of the approximation procedure, the first step of hybridization has been the inclusion of a local search inside a MOEA. In that scheme, EA drives the search procedure, and activates an NSA, with the aim to improve as far as possible the promising solutions resulting from the evolutionary operators. The NSA can simply be a depth first search method, or a more sophisticated method like a (truncated) tabu search exploiting advantageously the combinatorial structure of the optimization problem.

Figs. 3 and 4 report the results obtained for a bi-objective knapsack problem with two constraints and 250 items. The exact frontier has been computed with CPLEX. Approximations are the output of a single run of five MOMHs. The approximations obtained with three MOEAs without NSA (VEGA, NSGA, SPEA) are concentrated in the middle part of the non-dominated frontier (Fig. 3). Notice the gap of efficiency between VEGA, the pioneer method, and the two well-known methods, NSGA and SPEA, which is well visible on this example. The two MOEAs with a NSA (MOGLS and MOGTS) compute comparable approximations, both are well spread along the non-dominated frontier (Fig. 4).

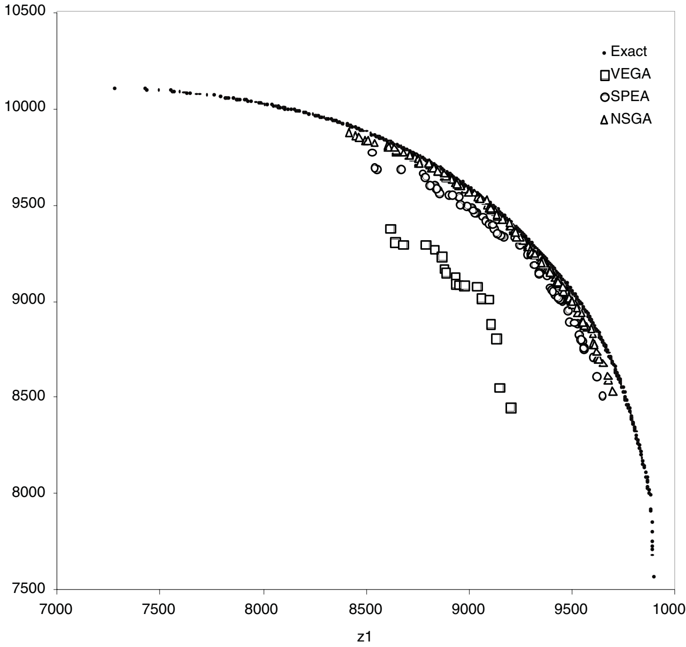


Fig. 3. Multi-objective multi-constrained knapsack problem (250 items, 2 objectives, and 2 constraints), solved with VEGA, NSGA, and SPEA.

The following references give an overview of methods falling in this category.

- Multiple Objective Genetic Algorithm (MOGA) by Murata and Ishibuchi, 1995 [94]. This method is not based on the Pareto ranking principle but

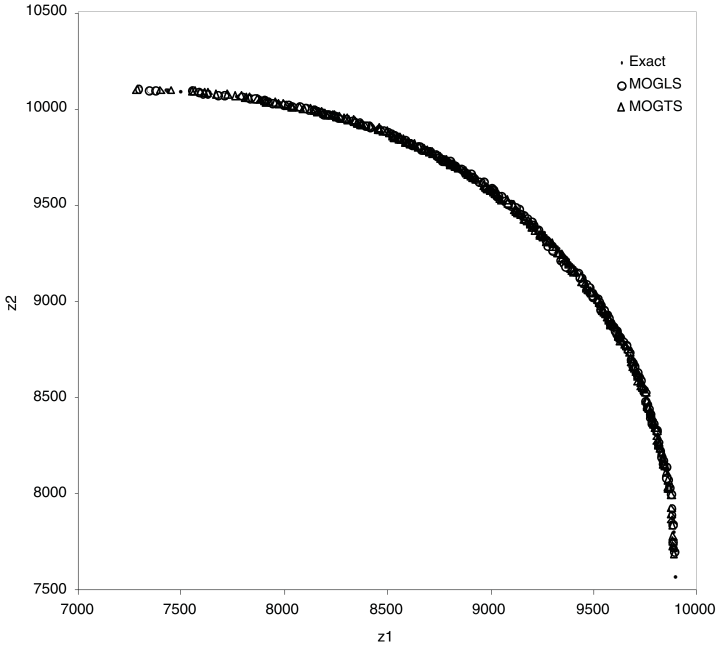


Fig. 4. Multi-objective multi-constrained knapsack problem (250 items, 2 objectives, and 2 constraints), solved with MOGLS and MOGTS.

on a weighted sum of objective functions, combining them into a scalar fitness function that uses randomly generated weight values in each iteration. Later, the authors coupled a local search with a genetic algorithm, introducing the memetic algorithm principle for MOPs.

- Method of Morita et al. (MGK) by Morita, Gandibleux and Katoh, 1998 [47]. Seeding solutions, either greedy or supported efficient, are put in the initial population in order to initialise the algorithm with good genetic information. The bi-objective knapsack problem is used to validate the principle. This method becomes a memetic algorithm when a local search has been performed on each new potentially efficient solution [48].
- Pareto Archived Evolution Strategy (PAES) by Knowles and Corne, 1999 [77]. PAES is an evolutionary strategy that employs local search to generate new candidate solutions and a reference archive to compute solution quality.
- Multiple Objective Genetic Local Search (MOGLS) by Jaszkiewicz, 2001 [69]. This method hybridizes recombination operators with local improvement heuristics. A scalarizing function is drawn at random for selecting solutions, which are then recombined, and the offspring of the recombination are improved using heuristics.

- Multiple Objective Genetic Tabu Search (MOGTS) by Barichard and Hao, 2002 [5]. This is another hybrid method in which a genetic algorithm is coupled with a tabu search. MOGTS has been evaluated on the multi-constraint knapsack problem.
- Multi-colony Ant System (MACS) by Gambardella, Taillard and Agazzi, 1999 [43]. A bi-criteria vehicle routing problem with time windows with the lexicographically sorted objectives of minimizing the number of vehicles and minimizing total travel distance is solved. MACS uses one colony for each objective and local as well as global pheromone update. The colonies cooperate through the use of the global best solution for the global pheromone update. Local search is applied to improve the quality of each solution found.
- T'Kindt et al., 2002 [128]. A (single) ant colony optimization approach is proposed for a two machine bi-criteria flowshop problem to minimize makespan and total flowtime in a lexicographic sense. The solutions produced by the ants are improved by local search.

Other references in this most popular category include [123, 124, 122], all of which apply an evolutionary algorithm combined with local search heuristics to variants of the vehicle routing problem. In [71] this idea is applied to the bi-objective set covering problem and [79] deals with the bi-objective arc routing problem following this strategy and [74] apply it to the TSP with profits.

4.2 Hybridization to Drive a Method

One fundamental question for "pure" NSA based MOMHs, like MOSA or MOTS, is the guidance mechanism along the non-dominated frontier. Influenced by the success of MOGA, authors introduced the principle of deriving search directions from a population of individuals. Here, global information about the current approximation is deduced and *drives local search processes* in order to "guarantee" a good coverage of the non-dominated frontier.

Using, for example, mechanisms based on notions of repulsion between non-dominated points, the search is guided toward subareas of the frontier containing (i) a high density of solutions or (ii) areas not yet explored.

This is the principle of the PSA [17] and the TAMOCO [63] methods. Dealing with the same question, Engrand's revised method [36, 100], and Shelokar et al. [114] use the non-domination definition to avoid the management of search directions.

- Pareto Simulated Annealing (PSA) by Czyzak and Jaszkiewicz, 1995 [16, 17, 18]. This method combines simulated annealing and genetic algorithm principles. The main ideas concern the management of weights and the consideration of a set of current solutions. A sample $S \subset X$ of $\#S$ solutions is determined and used as initial solutions. Each solution in this set is "optimized" iteratively, i.e., by generating neighbouring solutions that may be accepted according to a probabilistic strategy. For a given solution $\bar{x} \in S$ the weights are changed in order to increase the probability of moving it away from its closest neighbour in S denoted by $\bar{x}'$. Solutions in S play the role of agents working almost independently but exchanging information about their positions in the objective space. Thus, the interaction between solutions guides the generation process through the values of λ.

- Multi-objective Tabu Search (TAMOCO) by Hansen, 1997 [64]. This method uses a set of "generation solutions", each with its own tabu list. These solutions are dispersed throughout the objective space in order to allow searches in different areas of the non-dominated frontier (see Fig. 5). Weights are defined for each solution with the aim of forcing the search into a certain direction of the non-dominated frontier and away from other current solutions that are efficient with respect to it. Diversification is ensured by a set of generation solutions and a drift criterion. Results for the knapsack problem and for a resource constrained project scheduling problem are available in [134].

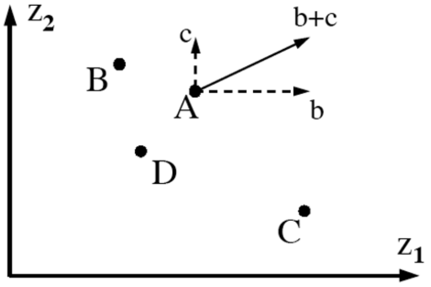


Fig. 5. The positions of four solutions A, B, C and D in objective space are shown (for a maximization problem). Solution A should be improved to move towards the non-dominated frontier but at the same time it should move away from other current solutions, which are non-dominated with respect to A (solutions B and C). Solution B pushes solution A away and this is shown by an optimization influence in the direction of vector b. Likewise does solution C influence solution A to move away from it in direction c. The final optimization direction for solution A is found by adding these weighted influence vectors. (The figure is reproduced from [63].)

- Engrand's method, 1997 [36, 37]. This is a hybridization of simulated annealing principles with genetic algorithms originally applied to a nuclear fuel management problem. Engrand's method has been revised later by Park and Suppapitnarm [100, 120, 121] and applied to the pressurized water reactor reload core design optimization problem. The main characteristic of this revised version is its ability to work without search directions, using a population of individuals to ensure the exploration of the complete trade-off surface. Each objective is considered separately. The method uses only the non-domination definition to select potentially efficient solutions, thus avoiding the management of search directions and aggregation mechanisms. Advanced strategies use the population of potentially efficient solutions to drive the approximation mechanism, thus ensuring the detection of the whole non-dominated frontier.

- The method of Shelokar et al., 2000 [112, 113, 114]. An ant algorithm for multi-objective continuous optimization is proposed. An interesting feature is that it combines the ant system methodology with the strength Pareto fitness assignment of [138] and clustering methods. The algorithm is applied to reliability engineering problems in [112] and to the optimization of reactor regenerator systems in [113].
- Armentano and Arroyo's method [4] is based on the tabu search principle. However, they work on a set of solutions following several paths, each having its own tabu-list. To diversify the search they apply a clustering technique to the set of potentially efficient solutions in any given iteration and use the centroids of clusters to define search directions. They report experiments on a bi-objective flowshop scheduling problem, obtaining good results compared to exact methods, local search, and tabu search.

4.3 Hybridization for Exploiting Complementary Strengths

The idea of using the complementary forces of metaheuristics was a natural way for the emergence of hybrid methods. A MOMH is often designed as a two step method, switching from method A to method B, with a communication process of solutions from A to B. The couple EA+NSA aims to take the advantage of, for example, a GA-based algorithm for building a set of good solutions in a first step followed by an aggressive search method, e.g., a TS-based algorithm, in a second step. The couple NSA+GA makes sense for example when an efficient algorithm is known for the single objective version of the optimization problem. The first step aims to poke around the non-dominated frontier, providing a sample of very good solutions covering very well the whole non-dominated frontier, while the second step has to fill out the approximation in terms of distribution. The following references illustrate three MOMHs falling in this category.

- Ben Abdelaziz et al.'s hybrid method, 1999 [9]. The authors present a hybrid algorithm using both EA and NSA independently. The goal of the EA (a genetic algorithm) is to produce a first diversified approximation, which is then improved by the NSA (a tabu search algorithm). Results have been reported on the multi-objective knapsack problem.
- Delorme et al. [19] design a scheme based on an NSA interfaced with an EA for solving the bi-objective set packing problem. The idea is to take advantage of an efficient heuristic known for the single objective problem in order to compute an initial set of very good solutions $\mathcal{P}_0$ in a first phase. The heuristic (a GRASP algorithm) is encapsulated in a basic generation

procedure, for example using a convex combination of the objectives: λ-GRASP. The second phase works on a population of individuals $\mathcal{P}$ derived from the initial set $\mathcal{P}_0$, and performs an EA (a modified version of SPEA dealing with all potential efficient solutions and integrating a local search: A-SPEA) in order to consolidate the approximation of the non-dominated frontier (see Fig. 6).

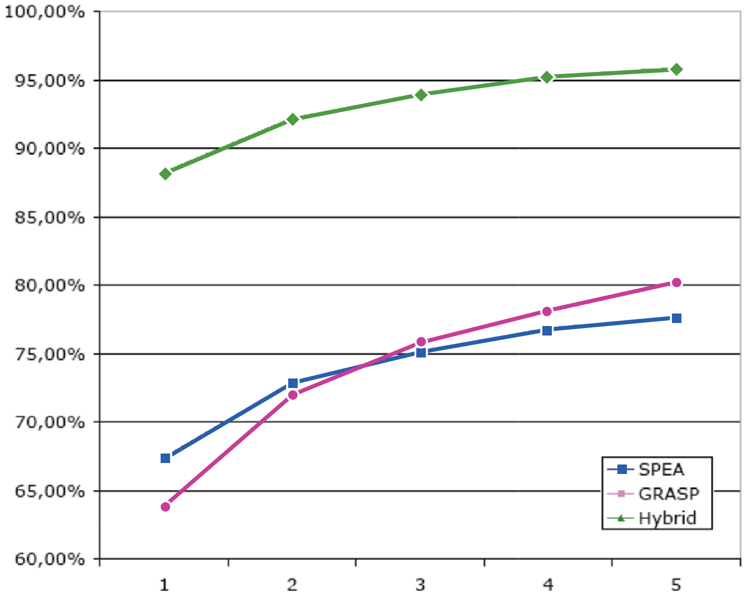


Fig. 6. The figure illustrates the average percentage of exact solutions found using λ-GRASP, A-SPEA, and the hybrid for the set packing problem with two objectives, when all three methods are allowed the same computational effort. (The figure is reproduced from [19].)

- López-Ibáñez, Paquete and Stützle [83] compare hybridizations of a multi-objective ant colony optimization algorithm and the evolutionary algorithm SPEA with short runs of a tabu search or a simple iterative improvement algorithm. The goal of the hybridization is to achieve an acceptable trade-off between solutions speed and solution quality. Their experimental study on the bi-objective quadratic assignment problem shows that characteristics of the instance have a strong influence on the performance of the variants.

4.4 Hybridization with Other Techniques

Embedding (meta)heuristics in an exact solution method or reversely is common in single objective optimization. More broadly speaking, the integration of techniques from several fields as operations research, artificial intelligence, and constraint programming has lead to interesting results on large and complex single objective problems. But it is astonishing to observe that this way to proceed is marginal in the multi-objective context. Four successful cases of that kind of hybridization are reported.

- Gandibleux and Fréville [45] propose a procedure for the bi-objective knapsack problem combining an exact procedure for reducing the search space with a tabu search process for identifying the potentially efficient solutions. The reduction principle is based on cuts which eliminate parts of the decision space where (provably) no exact efficient solution exists (see Fig. 7). It uses an additional constraint on the cardinality of an optimal solution for computing a utopian reference point and an approximation set for verifying if the reference point is dominated. The tabu search is triggered on the reduced space and dynamically updates the bounds in order to guarantee the tightest value at any time. Here the cuts help the metaheuristic.

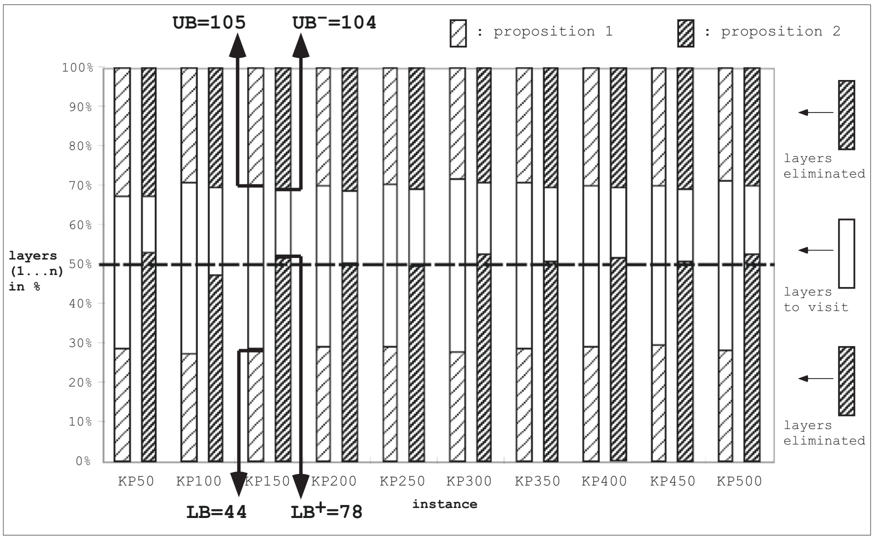


Fig. 7. The figure shows how two pre-processing procedures reduce the search area in decsision space by proving that solutions with small or large cardinality can not be efficient. E.g., for KP 150, the first procedure establishes that all efficient solutions must have between 44 and 105 items in the knapsack, whereas the second procedure shows they must have between 78 and 104 items. The vertical axis is relative because the results are for instances of size 50 to 500. (The figure is reproduced from [45].)

- Przybylski et al. [104] introduced the *seek and cut method* for solving the bi-objective assignment problem. The "seek" computes a local approximation of the non-dominated frontier (i.e., bounds are computed by a population-based algorithm coupled with path relinking) which is then used for "cutting" the search space of an implicit enumeration scheme (see Fig. 8). Here the metaheuristic helps the enumeration scheme in providing bounds of good quality for generating the exact non-dominated points.

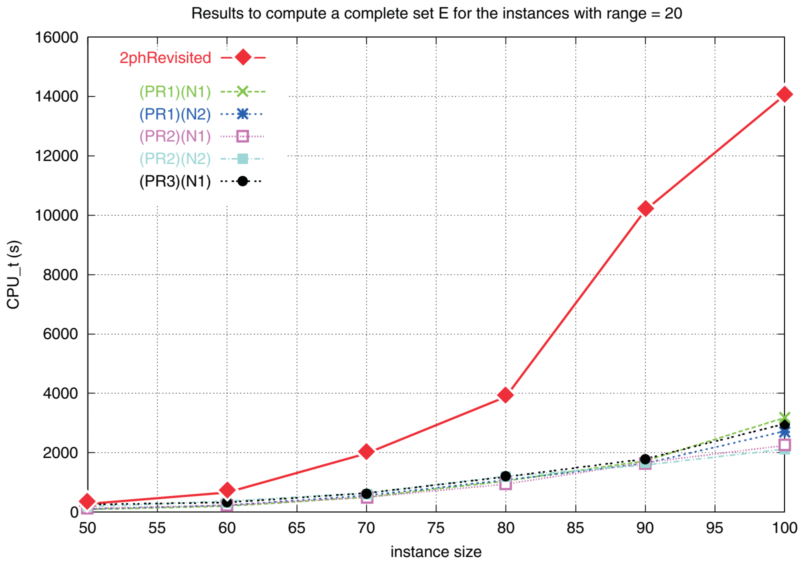


Fig. 8. CPU time used by an exact method for solving the assignment problem with two objectives without (the upper curve) and with (the lower curves) the use of approximate solutions for the pruning test inside the method of [104].

- Barichard introduced constraint programming techniques in the solution procedure of MOP problems with the PICPA method [6, 7]. The main goal of CP in PICPA is to build a sharp bound around the non-dominated frontier in the decision space using value propagation mechanisms over variables.
- Jozefowiez proposed Target Aiming Pareto Search (TAPaS) [73], a MOMH where the search directions of the procedure are given by the current set X_{PE}, similar to the principle of almost all the tabu search adaptations for MOP [46, 64, 119]. A series of goals is deduced from X_{PE} and a scalarizing function is used for guiding an NSA, defining a two phase strategy. For the covering tour problem, the method is coupled with an EA plus a branch and cut algorithm specifically designed for the single objective version of the problem.

We mention a few more papers with combinations of metaheuristics with exact methods for MOCO. In [57] scatter search is combined with the exact solution of small residual problems. [75] combine an evolutionary algorithm with a branch and cut algorithm to solve subproblems. The approach by [80] adaptively constructs right hand side values for the ε-constraint scalarization and solves the single objective subproblems, e.g., by an evolutionary algorithm.

Chen et al. [12] integrate simulation to model uncertainty and a genetic algorithm to solve a bi-level multi-objective stochastic program arising in built-operate-transfer network design problems.

4.5 Hybridization with Preferences Provided by a Decision-Maker

A decision-maker can be invited to play a role in a resolution process of an MOP. Information provided by the decision maker often concerns his preferences. Here, it is usual to distinguish three modes following the role of the decision maker in the resolution process. In “a priori mode”, all the preferences are known at the beginning of the decision making process. The techniques used seek for a solution on the basis of these parameters. In “a posteriori mode” the set of all efficient solutions is generated for the considered problem. At the end, this set is analyzed according to the decision maker’s preferences. Many MOMHs are designed following this solution mode. In the “interactive mode”, the preferences are introduced by the decision maker during the solution process. The methods involve a series of computing steps alternating with dialogue steps and can be viewed as the interactive determination of a satisfying compromise for the decision maker. Thus they require a high participation level on the part of the decision maker. Practical problems are often solved according to the interactive mode.

The interactive mode defines a kind of hybrid MOMH. The following references give an overview of some procedures proposed in that context.

- Alves and Clímaco [2] propose a general interactive method for solving 0-1 multi-objective problems where simulated annealing and tabu search work as two alternative and complementary computing procedures. It is a progressive and selective search of potentially efficient solutions by focusing the search on a subregion delimited by information for the objective function values specified by the decision-maker. Computational results for multiple-constraint knapsack problems with two objectives are reported.
- Pamuk and Köksalan proposed an evolutionary metaheuristic that interacts with the decision maker to guide the search effort towards his or her preferred solutions.
- PSA (see Sect. 4.2) has been coupled with an interactive procedure (light beam search) in order to organize an interactive search in [65].
- An interactive version of MOSA (see Sect. 3.6) has been introduced. In [125], a simulation with a fictitious decision maker is reported for the knapsack problem with four objectives and the assignment problem with three objectives. In [133] the interactive version is used for solving a real situation, the problem of homogeneous grouping of nuclear fuel.

4.6 Conclusions and Discussion

Literature overviews published these last 20 years about progress in the field of MOMH, show clearly that the methods have quickly transcended the perimeter of principles endogenous of metaheuristics. Various ideas and techniques have been progressively integrated for making the method more aggressive, for better tackling the specificities and difficulties of multi-objective optimization problems, for reusing well-established results in the single objective case, etc. MOMHs became a blend of heterogeneous components, hybrid methods by nature. Another important change relates to the abandonment of the idea to design a single universal method (solver). Methods are more and more specific, strongly related to the optimization problem to be solved. This double observation is particularly true when the problem to be solved is a MOCO problem.

The research field now has a significant background which allows to measure the strengths and weaknesses of ideas which were introduced. Postulating that an approximation method in multi-objective optimization is specialized for the problem to be solved, a MOMH appears today as collection of various techniques available in a library, which have to be combined together and instantiated on a given problem. Without being exhaustive, among the relevant techniques who can be embedded in components available for a MOMH designer, we find: a population of solutions, evolutionary operators, strategies of ranking/guiding/clustering, a neighbourhood structure, an exploration strategy (partial, exhaustive), a scalarizing function, etc. Three components have been recently underlined as important for the efficiency of a MOMH designed for solving MOCO problems:

- **The initial solutions.** In [81] Zitzler has underlined the significant role played by elite (potentially efficient) solutions in a MOEA. However, the initial population is often composed of feasible solutions built randomly. For MOCO problems, subsets of exact solutions or approximate solutions computed with a greedy or an ad-hoc algorithm can be advantageously exploited. The role of elite solutions in the generation of the non-dominated frontier has been investigated by Gandibleux et al. [48, 93] for the knapsack problem. Using greedy solutions, or efficient supported solutions, in the initial population makes the algorithm more apt to generate quickly efficient solutions. Haubelt et al. [66] obtained the same conclusion for a MOCO problem in the field of embedded system synthesis. Gandibleux et al. [51] compute a complete set of exact supported solutions for the bi-objective assignment problem. Pasia et al. [101] use a single objective ACO algorithm for generating individuals objective by objective for a flowshop problem.
- **Lower and upper bounds on the non-dominated frontier.** A local search is usually an expensive procedure in terms of computing time. Triggering a local search from a solution, while it has little chance of generating new potentially efficient solutions is a waste of time. A solution

can be a candidate for performing a local search basically if it close to the non-dominated frontier. Knowing a lower bound on the non-dominated frontier can help to implement such a strategy. Bounds and bound sets are discussed by Ehrgott and Gandibleux [31]. The principle has been implemented successfully by Gandibleux et al. [51] and Paisa et al. [101].

- **A path relinking operator.** In Gandibleux et al. [50], similarities in solutions and subsets of exact solutions are used advantageously by the components of an evolutionary method. Here, interesting performance results are measured with a *path relinking* operator [53], given a subset of optimal solutions (or approximations) in the initial population. Path relinking generates new solutions by exploring the trajectories that connect good solutions. A path relinking operation starts by randomly selecting I_A (the initiating solution) and I_B (the guiding solution), two individuals from the current population (Fig. 9). The path relinking operation generates a path $I_A(= I_0), I_1, \ldots, I_B$ through the neighbourhood space, such that the distance between I_i and I_B decreases monotonically in i, where the distance is defined as the number of positions for which different values are assigned in I_i and I_B.
  Although many such paths may possibly exist, one path is chosen using, for example, random moves based on a swap operator. Such randomness introduces a form of diversity to the solutions generated along the path. For every intermediate solution I_i, a single solution is generated in the neighbourhood (Fig. 9). Introduced for the first time in 2003 in multi-objective optimization [49], path relinking has shown clearly its impact for the approximation of efficient solutions. This principle has been successfully implemented for computing the approximation of the complete non-dominated frontier of assignment and knapsack problems with two objectives [51], and recently on a bi-objective flowshop problem [101]. Fig. 10 provides a sample of these results. Using the same numerical instances, this population-based method based on specific operators outperforms MOSA [129].

Despite all the work done, many hot topics are open in the field of hybrid metaheuristics for multi-objective combinatorial optimization. Among them, we mention the scalability problem, the reduction of (decision and objective) spaces, the impact of objective functions, or the solution of problems with more than two objectives.

Answers may come from effectively reusing 50 years of knowledge in (single objective) optimization, the coupling with others techniques like constraint programming, or again the design of new efficient components for combinatorial problems, like the path relinking. These challenges promise many future papers.

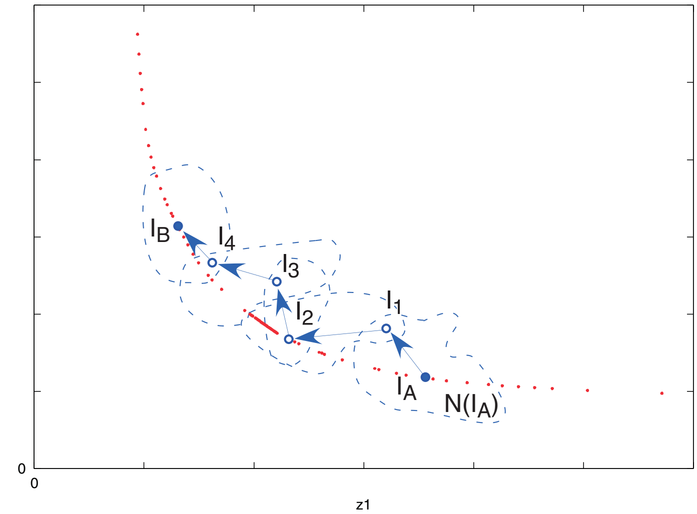


Fig. 9. Illustration of a possible path construction (see [50]). I_A and I_B are two individuals randomly selected from the current elite population (small bullets). I_A is the initiating solution, and I_B is the guiding solution. $\mathcal{N}(I_A)$ is the feasible neighbourhood according to the move defined. $I_A - I_1 - I_2 - I_3 - I_4 - I_B$ is the path that is built.

References

1. P. Agrell, M. Sun, and A. Stam. A tabu search multi-criteria decision model for facility location planning. In *Proceedings of the 1997 DSI Annual Meeting, San Diego, California*, volume 2, pages 908–910. Decision Sciences Institute, Atlanta, GA, 1997.
2. M. J. Alves and J. Climaco. An interactive method for 0-1 multiobjective problems using simulated annealing and tabu search. *Journal of Heuristics*, 6(3):385–403, 2000.
3. E. Angel, E. Bampis, and L. Gourvès. Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoretical Computer Science*, 310:135–146, 2004.
4. V. A. Armentano and J. E. C. Arroyo. An application of a multi-objective tabu search algorithm to a bicriteria flowshop problem. *Journal of Heuristics*, 10:463–481, 2004.
5. V. Barichard and J.-K. Hao. Un algorithme hybride pour le problème de sac à dos multi-objectifs. In *Huitièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets JNPC'2002 Proceedings*, 2002. Nice, France, 27–29 May.
6. V. Barichard and J.-K. Hao. A population and interval constraint propagation algorithm. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8-11, 2003. Proceedings*, volume 2632 of *Lecture Notes in Computer Science*, pages 88–101. Springer-Verlag, Berlin, Germany, 2003.

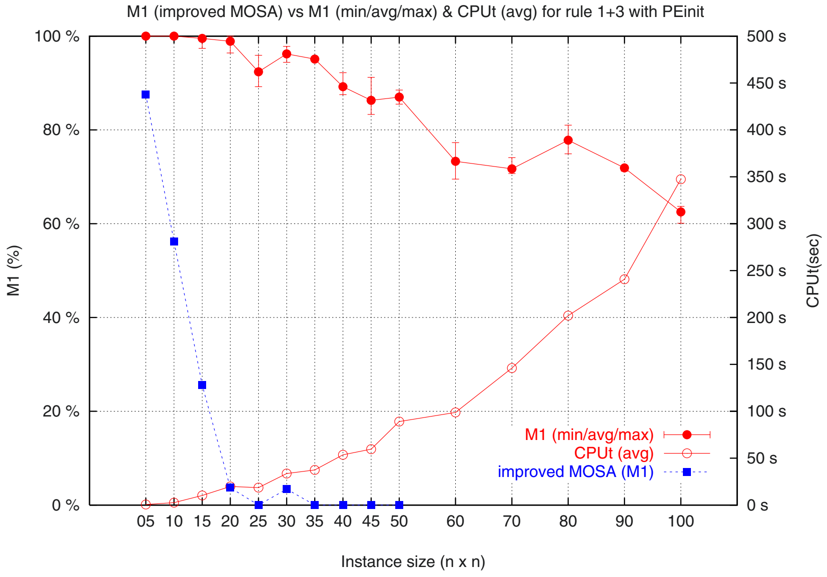


Fig. 10. Approximations obtained with the MOSA method and a population-based method for the assignment problem with two objectives. The comparison is based on the performance measure M_1 [130], which measures the percentage of exact solutions included in the final approximation set.

7. V. Barichard and J.-K. Hao. A population and interval constraint propagation algorithm for mulitobjective optimization. In *Proceedings of The Fifth Metaheuristics International Conference MIC'03*, paper ID MIC03–04. CD ROM, 2003.
8. R. Beausoleil. Multiple criteria scatter search. In J. P. de Sousa, editor, *MIC 2001 Proceedings of the 4th Metaheuristics International Conference, Porto, July 16-20, 2001*, volume 2, pages 539–543, 2001.
9. F. Ben Abdelaziz, J. Chaouachi, and S. Krichen. A hybrid heuristic for multiobjective knapsack problems. In S. Voß, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 205–212. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.
10. W. M. Carlyle, J. W. Fowler, E. S. Gel, and B. Kim. Quantitative comparison of approximate solution sets for bi-criteria optimization problems. *Decision Sciences*, 34(1):63–82, 2003.
11. R. L. Carraway, T. L. Morin, and H. Moskovitz. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research*, 44:95–104, 1990.
12. A. Chen, K. Subprasom, and Z. Ji. A simulation-based multi-objective genetic algorithm (SMOGA) procedure for BOT network design. *Optimization and Engineering*, 7:225–247, 2006.
13. C. A. Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

14. C. A. Coello. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.
15. Y. Collette and P. Siarry. Three new metrics to measure the convergence of metaheuristics towards the Pareto frontier and the aesthetic of a set of solutions in biobjcetive optimization. *Computers and Operations Research*, 32:773–792, 2005.
16. P. Czyżak and A. Jaszkiewicz. A multiobjective metaheuristic approach to the localization of a chain of petrol stations by the capital budgeting model. *Control and Cybernetics*, 25(1):177–187, 1996.
17. P. Czyżak and A. Jaszkiewicz. Pareto simulated annealing. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making. Proceedings of the XIIth International Conference, Hagen (Germany)*, volume 448 of *Lecture Notes in Economics and Mathematical Systems*, pages 297–307. Springer-Verlag, Berlin, Germany, 1997.
18. P. Czyżak and A. Jaszkiewicz. Pareto simulated annealing – A metaheuristic technique for multiple objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.
19. X. Delorme, X. Gandibleux, and F. Degoutin. Evolutionary, constructive and hybrid procedures for the biobjective set packing problem. September 2005. In revision (European Journal of Operational Research) Research report EMSE 2005-500-011, Ecole des Mines de Saint-Etienne, 2005.
20. X. Delorme, X. Gandibleux, and J. Rodriguez. Résolution d'un problème d'évaluation de capacité d'infrastructure ferroviaire. In *Actes du colloque sur l'innovation technologique pour les transports terrestres (TILT)*, volume 2, pages 647–654. GRRT Lille, 2003.
21. K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Ant colony optimization in multiobjective portfolio selection. In J. P. de Sousa, editor, *MIC'2001 Proceedings of the 4th Metaheurstics International Conference, Porto, July16-20, 2001*, volume 1, pages 243–248, 2001.
22. K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Investitionsentscheidungen bei mehrfachen Zielsetzungen und künstliche Ameisen. In P. Chamoni, R. Leisten, A. Martin, J. Minnemann, and H. Stadtler, editors, *Operations Research Proceedings 2001, Selected Papers of OR 2001*, pages 355–362. Springer-Verlag, Berlin, Germany, 2002.
23. K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization: A metaheuristic approach to multiobjective portfolio selection. *Annals of Operations Research*, 131:79–99, 2004.
24. K. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, and C. Stummer. Pareto ant colony optimization with ILP preprocessing in multiobjective portfolio selection. *European Journal of Operational Research*, 171:830–841, 2006.
25. K. Doerner, R. F. Hartl, and M. Reimann. Are COMPETants more competent for problem solving? The case of a multiple objective transportation problem. In L. Lee Spector, A. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 802. Morgan Kaufmann, San Francisco, CA, 2001.
26. M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transcations in Operational Research*, 7:5–31, 2000.

27. M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22:425–460, 2000.
28. M. Ehrgott and X. Gandibleux. Multiobjective combinatorial optimization. In M. Ehrgott and X. Gandibleux, editors, *Multiple Criteria Optimization – State of the Art Annotated Bibliographic Surveys*, volume 52 of *Kluwer's International Series in Operations Research & Management Science*, pages 369–444. Kluwer Academic Publishers, Boston, MA, 2002.
29. M. Ehrgott and X. Gandibleux, editors. *Multiple Criteria Optimization – State of the Art Annotated Bibliographic Surveys*, volume 52 of *Kluwer's International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Boston, MA, 2002.
30. M. Ehrgott and X. Gandibleux. Approximative solution methods for multiobjective combinatorial optimization. *TOP*, 12(1):1–88, 2004.
31. M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34:2674–2694, 2007.
32. M. Ehrgott, K. Klamroth, and S. Schwehm. An MCDM approach to portfolio optimization. *European Journal of Operational Research*, 155(3):752–770, 2004.
33. M. Ehrgott and D. M. Ryan. Constructing robust crew schedules with bicriteria optimization. *Journal of Multi-Criteria Decision Analysis*, 11:139–150, 2002.
34. M. Ehrgott and D. Tenfelde-Podehl. Computation of ideal and nadir values and implications for their use in MCDM methods. *European Journal of Operational Research*, 151(1):119–131, 2003.
35. M. Ehrgott and M. Wiecek. Multiobjective programming. In J. Figueira, S. Greco, and M. Ehrgott, editors, *Multicriteria Decision Analysis: State of the Art Surveys*, pages 667–722. Springer Science + Business Media, New York, 2005.
36. P. Engrand. A multi-objective approach based on simulated annealing and its application to nuclear fuel management. In *Proceedings of the 5th ASME/SFEN/JSME International Conference on Nuclear Engineering. Icone 5, Nice, France 1997*, pages 416–423. American Society of Mechanical Engineers, New York, NY, 1997.
37. P. Engrand and X. Mouney. Une méthode originale d'optimisation multi-objectif. Technical Report 98NJ00005, EDF-DER Clamart, France, 1998.
38. T. Erlebach, H. Kellerer, and U. Pferschy. Approximating multiobjective knapsack problems. *Management Science*, 48(12):1603–1612, 2002.
39. E. Fernández and J. Puerto. Multiobjective solution of the uncapacitated plant location problem. *European Journal of Operational Research*, 145(3):509–529, 2003.
40. C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, 1993. University of Illinois at Urbana-Champaign*, pages 416–423. Morgan Kaufmann, San Francisco, CA, 1993.
41. C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.
42. M. P. Fourman. Compaction of Symbolic Layout using Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 141–153. Lawrence Erlbaum, 1985.

43. L. M. Gambardella, E. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, London, 1999.
44. X. Gandibleux, F. Beugnies, and S. Randriamasy. Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function. *4OR: Quarterly Journal of Operations Research*, 4(1):47–59, 2006.
45. X. Gandibleux and A. Fréville. Tabu search based procedure for solving the 0/1 multiobjective knapsack problem: The two objective case. *Journal of Heuristics*, 6(3):361–383, 2000.
46. X. Gandibleux, N. Mezdaoui, and A. Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In R. Caballero, F. Ruiz, and R. Steuer, editors, *Advances in Multiple Objective and Goal Programming*, volume 455 of *Lecture Notes in Economics and Mathematical Systems*, pages 291–300. Springer-Verlag, Berlin, Germany, 1997.
47. X. Gandibleux, H. Morita, and N. Katoh. A genetic algorithm for 0-1 multiobjective knapsack problem. In *International Conference on Nonlinear Analysis and Convex Analysis (NACA98) Proceedings*, 1998. July 28-31 1998, Niigata, Japan.
48. X. Gandibleux, H. Morita, and N. Katoh. The supported solutions used as a genetic information in a population heuristic. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 429–442. Springer-Verlag, Berlin, Germany, 2001.
49. X. Gandibleux, H. Morita, and N. Katoh. Impact of clusters, path-relinking and mutation operators on the heuristic using a genetic heritage for solving assignment problems with two objectives. In *Proceedings of The Fifth Metaheuristics International Conference MIC'03*, pages Paper ID MIC03–23. CD ROM, 2003.
50. X. Gandibleux, H. Morita, and N. Katoh. A population-based metaheuristic for solving assignment problems with two objectives. Technical Report n^{o}7/2003/ROI, LAMIH, Université de Valenciennes, 2003.
51. X. Gandibleux, H. Morita, and N. Katoh. Evolutionary operators based on elite solutions for biobjective combinatorial optimization. In C. Coello Coello and G. Lamont, editors, *Applications of Multi-Objective Evolutionary Algorithms*, chapter 23, pages 555–579. World Scientific, Singapore, 2004.
52. X. Gandibleux, D. Vancoppenolle, and D. Tuyttens. A first making use of GRASP for solving MOCO problems. Technical report, University of Valenciennes, France, 1998. Paper presented at MCDM 14, June 8-12 1998, Charlottesville, VA.
53. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.
54. F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.
55. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co., Reading, MA, 1989.
56. C. Gomes da Silva, J. Climaco, and J. Figueira. A scatter search method for bi-criteria $\{0, 1\}$-knapsack problems. *European Journal of Operational Research*, 169:373–391, 2006.

57. C. Gomes da Silva, J. Figueira, and J. Clímaco. Integrating partial optimization with scatter search for solving bi-criteria $\{0,1\}$-knapsack problems. *European Journal of Operational Research*, 177:1656–1677, 2007.
58. L. Gourvès. *Approximation polynomiale et optimisation combinatoire multicritère*. PhD thesis, Université dÉvry Val d'Essone, 2005.
59. M. Gravel, W. L. Price, and C. Gagné. Scheduling continuous casting of aluminium using a multiple objective ant colony optimization metaheuristic. *European Journal of Operational Research*, 143(1):218–229, 2002.
60. J. J. Grefenstette. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165. Oakland University, Rochester, MI, 1984.
61. P. Hajela and C. Y. Lin. Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4:99–107, 1992.
62. H. W. Hamacher and K.-H. Küfer. Inverse radiation therapy planing – A multiple objective optimization approach. *Discrete Applied Mathematics*, 118(1-2):145–161, 2002.
63. M. P. Hansen. *Metaheuristics for multiple objective combinatorial optimization*. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, Lyngby (Denmark), 1998. Report IMM-PHD-1998-45.
64. M. P. Hansen. Tabu search for multiobjective combinatorial optimization: TAMOCO. *Control and Cybernetics*, 29(3):799–818, 2000.
65. M. Hapke, A. Jaszkiewicz, and R. Slowinski. Interactive analysis of multiple-criteria project scheduling problems. *European Journal of Operational Research*, 107(2):315–324, 1998.
66. C. Haubelt, J. Gamenik, and J. Teich. Initial population construction for convergence improvement of MOEAs. In C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Sciences*, pages 191–205. Springer-Verlag, Berlin, Germany, 2005.
67. J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, FL, 29 June – 1 July 1994*, volume 1, pages 82–87. IEEE Service Center, Piscataway, NJ, 1994.
68. S. Iredi, D. Merkle, and M. Middendorf. Bi-criterion optimization with multi colony ant algorithms. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 359–372. Springer-Verlag, Berlin, Germany, 2001.
69. A. Jaszkiewicz. Multiple objective genetic local search algorithm. In M. Köksalan and S. Zionts, editors, *Multiple Criteria Decision Making in the New Millennium*, volume 507 of *Lecture Notes in Economics and Mathematical Systems*, pages 231–240. Springer-Verlag, Berlin, Germany, 2001.
70. A. Jaszkiewicz. Multiple objective metaheuristic algorithms for combinatorial optimization. Habilitation thesis, Poznan University of Technology, Poznan (Poland), 2001.
71. A. Jaszkiewicz. A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the Pareto memetic algorithm. *Annals of Operations Research*, 131:135–158, 2004.

72. D. Jones, S. K. Mirrazavi, and M. Tamiz. Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European Journal of Operational Research*, 137(1):1–9, 2002.
73. N. Jozefowiez. *Modélisation et résolution approchée de problèmes de tournées multi-objectif*. PhD thesis, Université de Lille 1, France, 2004.
74. N. Jozefowiez, F. Glover, and M. Laguna. A hybrid meta-heuristic for the traveling salesman problem with profits. Technical report, Leeds School of Business, University of Colorado at Boulder, 2006.
75. N. Jozefowiez, F. Semet, and E. G. Talbi. The bi-objective covering tour problem. *Computers and Operations Research*, 34:1929–1942, 2007.
76. J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948. IEEE Service Center, Piscataway, NJ, 1995.
77. J. D. Knowles and D. W. Corne. The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation. Washington, D.C.*, pages 98–105. IEEE Service Center, Piscataway, NJ, 1999.
78. F. Kursawe. Evolution strategies for vector optimization. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making, Taipei-Taiwan*, volume III, pages 187–193, 1992.
79. P. Lacomme, C. Prins, and M. Sevaux. A genetic algorithm for a bi-objective arc routing problem. *Computers and Operations Research*, 33:3473–3493, 2006.
80. M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169:932–942, 2006.
81. M. Laumanns, E. Zitzler, and L. Thiele. On the effect of archiving, elitism, and density based selection in evolutionary multi-objective optimization. In *Evolutionary Multi-Criteria Optimization. First International Conference, EMO 2001. Zürich, Switzerland, March 7–9, 2001. Proceedings*, volume 1993 of *Lecture Notes in Computer Science*, pages 181–196. Springer-Verlag, Berlin, Germany, 2001.
82. H. Lee and P. S. Pulat. Bicriteria network flow problems: Integer case. *European Journal of Operational Research*, 66:148–157, 1993.
83. M. López-Ibáñez, L. Paquete, and T. Stützle. Hybrid population-based algorithms for the biobjective quadratic assignment problem. Technical report, Computer Science Department, Darmstadt University of Technology, 2004.
84. P. Lučić and D. Teodorović. Simulated annealing for the multi-objective aircrew rostering problem. *Transportation Research A: Policy and Practice*, 33(1):19–45, 1999.
85. B. Manthey and L. S. Ram. Approximation algorithms for multi-criteria traveling salesman problems. In T. Erlebach and C. Kaklamanis, editors, *Approximation and Online Algorithms*, volume 4368 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, Berlin, Germany, 2007.
86. C. E. Mariano and E. Morales. MOAQ and ant-Q algorithm for multiple objective optimization problems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, Florida, USA, 13-17 July 1999*, volume 1, pages 894–901. Morgan Kaufmann, San Francisco, CA, 1999.

87. C. E. Mariano and E. Morales. A multiple objective ant-q algorithm for the design of water distribution irrigation networks. Technical Report HC-9904, Instituto Mexicano de Tecnología del Agua, 1999.
88. G. Mavrotas and D. Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107(3):530–541, 1998.
89. P. R. McMullen. An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. *Artificial Intelligence in Engineering*, 15:309–317, 2001.
90. P. R. McMullen and G. V. Frazier. Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations. *International Journal of Production Research*, 36(10):2717–2741, 1999.
91. K. Miettinen. *Nonlinear Multiobjective Optimization*, volume 12 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.
92. J. Moore and R. Chapman. Application of particle swarm to multiobjective optimization. Technical report, Department of Computer Science and Software Engineering, Auburn University, 1999.
93. H. Morita, X. Gandibleux, and N. Katoh. Experimental feedback on biobjective permutation scheduling problems solved with a population heuristic. *Foundations of Computing and Decision Sciences Journal*, 26(1):23–50, 2001.
94. T. Murata and H. Ishibuchi. MOGA: Multi-objective genetic algorithms. In *Proceedings of the 2nd IEEE International Conference on Evolutionary Computing, Perth, Australia*, pages 289–294. IEEE Service Center, Piscataway, NJ, 1995.
95. D. Nam and C. H. Park. Multiobjective simulated annealing: A comparative study to evolutionary algorithms. *International Journal of Fuzzy Systems*, 2(2):87–97, 2000.
96. C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access to web sources. In *Proceedings of the 41st Annual Symposium on the Foundation of Computer Science FOCS00*, pages 86–92. IEEE Computer Society, Los Alamitos, CA, 2000.
97. L. Paquete and T. Stützle. A two-phase local search for the biobjective traveling salesman problem. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization – Second International Conference, EMO 2003, Faro, Portugal, April 8-11, 2003, Proceedings*, volume 2632 of *Lecture Notes in Computer Science*, pages 479–493. Springer-Verlag, Berlin, Germany, 2003.
98. L. Paquete and T. Stützle. A study of stochastic local search for the biobjective QAP with correlated flow matrices. *European Journal of Operational Research*, 169:943–959, 2006.
99. L. F. Paquete. *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methods and Analysis.* PhD thesis, Department of Computer Science, Technical University of Darmstadt, 2005.
100. G. Parks and A. Suppapitnarm. Multiobjective optimization of PWR reload core designs using simulated annealing. In *Proceedings of the International Conference on Mathematics and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications. Madrid, Spain, September 1999*, volume 2, pages 1435–1444. Senda Editorial S. A., Madrid, Spain, 1999.

101. J. M. Pasia, X. Gandibleux, K. F. Doerner, and R. F. Hartl. Local search guided by path relinking and heuristic bounds. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Evolutionary Multi-Criterion Optimization*, volume 4403 of *Lecture Notes in Computer Science*, pages 501–515. Springer-Verlag, Berlin, Germany, 2007.
102. A. Przybylski, X. Gandibleux, and M. Ehrgott. Recursive algorithms for finding all nondominated extreme points in the outcome set of a multiobjective integer program. Technical report, LINA, Université de Nantes, 2007. Submitted for publication.
103. A. Przybylski, X. Gandibleux, and M. Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. Technical report, LINA – Laboratoire d'Informatique de Nantes Atlantique, 2007.
104. A. Przybylski, X. Gandibleux, and M. Ehrgott. Two phase algorithms for the biobjective assignment problem. *European Journal of Operational Research*, 185:509–533, 2008.
105. A. R. Rahimi-Vahed and S. M. Mirghorbani. A multi-objective particle swarm for a flow shop scheduling problem. *Journal of Combinatorial Optimization*, 13:79–102, 2007.
106. R. M. Ramos, S. Alonso, J. Sicilia, and C. González. The problem of the optimal biobjective spanning tree. *European Journal of Operational Research*, 111:617–628, 1998.
107. S. Randriamasy, X. Gandibleux, J. Figueira, and P. Thomin. Device and a method for determining routing paths in a communication network in the presence of selection attributes. Patent 11/25/04. #20040233850. Washington, DC, USA. www.freepatentsonline.com/20040233850.htm, 2004.
108. S. Sayın. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87:543–560, 2000.
109. J. D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, Nashville, TN (USA), 1984.
110. J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, Pittsburgh, PA, 1985.
111. P. Serafini. Simulated annealing for multiobjective optimization problems. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making, Taipei-Taiwan*, volume I, pages 87–96, 1992.
112. P. S. Shelokar, V. K. Jarayaman, and B. D. Kulkarni. Ant algorithm for single and multiobjective reliability optimization problems. *Quality and Reliability Engineering International*, 18(6):497–514, 2002.
113. P. S. Shelokar, V. K. Jayarama, and B. D. Kulkarni. Multiobjective optimization of reactor-regenerator system using ant algorithm. *Petroleum Science and Technology*, 21(7&8):1167–1184, 2003.
114. P. S. Shelokar, S. Adhikari, R. Vakil, V. K. Jayaraman, and B. D. Kulkarni. Multiobjective ant algorithm: Combination of strength Pareto fitness assignment and thermodynamic clustering. *Foundations of Computing and Decision Sciences*, 25(4):213–230, 2000.

115. K. Sörensen. Multi-objective optimization of mobile phone keymaps for typing messages using a word list. *European Journal of Operational Research*, 179:838–846, 2007.

116. F. Sourd, O. Spanjaard, and P. Perny. Multi-objective branch and bound. application to the bi-objective spanning tree problem. Technical report, Department of Decision, Intelligent Systems and Operations Research Université Pierre et Marie Curie, Paris, 2006.

117. N. Srinivas and K. Deb. Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

118. R. Steuer, J. Silverman, and A. Whisman. A combined Tchebycheff/aspiration criterion vector interactive multiobjective programming procedure. *Management Science*, 39(10):1255–1260, 1993.

119. M. Sun. Applying tabu search to multiple objective combinatorial optimization problems. In *Proceedings of the 1997 DSI Annual Meeting, San Diego, California*, volume 2, pages 945–947. Decision Sciences Institute, Atlanta, GA, 1997.

120. A. Suppapitnarm and G. Parks. Simulated annealing: An alternative approach to true multiobjective optimization. In A. S. Wu, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99). Orlando, Florida*. Morgan Kaufmann, San Francisco, CA, 1999.

121. A. Suppapitnarm, K. Seffen, G. Parks, and P. Clarkson. A simulated annealing algorithm for multiobjective optimization. *Engineering Optimization*, 33(1):59–85, 2000.

122. K. C. Tan, C. Y. Cheong, and C. K. Goh. Solving multiobjective vehicel routing problem with stochastic demand via evolutionary computation. *European Journal of Operational Research*, 177:813–839, 2007.

123. K. C. Tan, Y. H. Chew, and L. H. Lee. A hybrid multi-objective evolutionary algorithm for solving truck and trailer vehicle routing problems. *European Journal of Operational Research*, 172:855–885, 2006.

124. K. C. Tan, Y. H. Chew, and L. H. Lee. A hybrid multiobjective evolutionary algorithm for solving vehicle routing problem with time windows. *Computational Optimization and Applications*, 34:115–151, 2006.

125. J. Teghem, D. Tuyttens, and E. L. Ulungu. An interactive heuristic method for multi-objective combinatorial optimization. *Computers and Operations Research*, 27(7-8):621–634, 2000.

126. D. Tenfelde-Podehl. *Facilities Layout Problems: Polyhedral Structure, Multiple Objectives and Robustness*. PhD thesis, University of Kaiserslautern, Department of Mathematics, 2002.

127. M. Thompson. Application of multi objective evolutionary algorithms to analogue filter tuning. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 546–559. Springer-Verlag, Berlin, Germany, 2001.

128. V. T'kindt, N. Monmarché, F. Tercinet, and D. Laügt. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2):250–257, 2002.

129. D. Tuyttens, J. Teghem, P. Fortemps, and K. Van Nieuwenhuyse. Performance of the MOSA method for the bicriteria assignment problem. *Journal of Heuristics*, 6(3):295–310, 2000.

130. E. L. Ulungu. *Optimisation combinatoire multicritère: Détermination de l'ensemble des solutions efficaces et méthodes interactives*. PhD thesis, Faculté des Sciences, Université de Mons-Hainaut. Mons, Belgium, 1993.
131. E. L. Ulungu and J. Teghem. The two-phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1994.
132. E. L. Ulungu, J. Teghem, P Fortemps, and D. Tuyttens. MOSA method: A tool for solving multi-objective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8(4):221–236, 1999.
133. E. L. Ulungu, J. Teghem, and C. Ost. Efficiency of interactive multi-objective simulated annealing through a case study. *Journal of the Operational Research Society*, 49:1044–1050, 1998.
134. A. Viana and J. Pinho de Sousa. Using metaheuristics in multiobjective ressource constrained project scheduling. *European Journal of Operational Research*, 120(2):359–374, 2000.
135. M. Visée, J. Teghem, M. Pirlot, and E. L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-obective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998.
136. A. Warburton. Approximation of Pareto optima in multiple-objective shortest-path problems. *Operations Research*, 35(1):70 –79, 1987.
137. H. Yapicioglu, A. E. Smith, and G. Dozier. Solving the semi-desirable facility location problem using bi-objective particle swarm. *European Journal of Operational Research*, 177:733–749, 2007.
138. E. Zitzler and L. Thiele. An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. Technical Report 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, May 1998.
139. E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.
140. E. Ziztler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

Multilevel Refinement for Combinatorial Optimisation: Boosting Metaheuristic Performance

Chris Walshaw

Computing and Mathematical Sciences, University of Greenwich
Old Royal Naval College, Greenwich, London
C.Walshaw@gre.ac.uk

Summary. The multilevel paradigm as applied to combinatorial optimisation problems is a simple one, which at its most basic involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found, usually at the coarsest level, and then iteratively refined at each level, coarsest to finest, typically by using some kind of heuristic optimisation algorithm (either a problem-specific local search scheme or a metaheuristic). Solution extension (or projection) operators can transfer the solution from one level to another. As a general solution strategy, the multilevel paradigm has been in use for many years and has been applied to many problem areas (for example multigrid techniques can be viewed as a prime example of the paradigm). Overview papers such as [39] attest to its efficacy. However, with the exception of the graph partitioning problem, multilevel techniques have not been widely applied to combinatorial problems and in this chapter we discuss recent developments. In this chapter we survey the use of multilevel combinatorial techniques and consider their ability to boost the performance of (meta)heuristic optimisation algorithms.

1 Introduction

The last 50 years have seen a huge amount of research effort devoted to the study of combinatorial optimisation problems. Spanning applied mathematics, through operations research, to management sciences, such problems are typically concerned with the allocation of resources in some way and are used in many diverse applications including scheduling, timetabling, logistics and the design of computer components.

There are now a bewildering array of (meta)heuristic optimisation algorithms (e.g. ant colony optimisation, genetic algorithms, simulated annealing, tabu search, variable neighbourhoods, etc., [4]) to address such problems and for any given application, the practitioner must often answer the question '*which is likely to be the best algorithm for my problem?*'

C. Walshaw: *Multilevel Refinement for Combinatorial Optimisation: Boosting Metaheuristic Performance*, Studies in Computational Intelligence (SCI) **114**, 261–289 (2008)
www.springerlink.com

In considering the multilevel paradigm however, a different question occurs. Since the multilevel framework is a *collaborative* one, which always acts in concert with some other technique, the question that arises is '*given that I am using algorithm X for addressing my problem, can its performance be boosted by using a multilevel version of algorithm X?*'

As we shall see, very often the answer appears to be emphatically *yes*, either in terms of the solution quality, or the computational runtime, or both.

Indeed, in certain instances, it does seem to be the case that, in terms of performance, the decision to use a multilevel scheme is far more important then the actual choice of the underlying search algorithm.

Even more encouragingly, anecdotal evidence suggests that, in developing a full-blown multilevel scheme, the multilevel framework (which at its most basic requires a coarsening algorithm and a projection/extension operator) is generally far easier to implement than a high-quality optimisation algorithm for the problem. In fact, it is often the case that a coarsening algorithm can be built from components found in solution construction heuristics, i.e. those which are usually used to find an initial feasible (although poor quality) solution, whilst the solution extension operator is usually a trivial reversal of coarsening.

1.1 Overview

The rest of the chapter is organised as follows.

First we motivate the ideas and, via some sample results, consider evidence for the strengths of the multilevel paradigm by describing by discussing the widespread use of multilevel graph partitioning schemes. Multilevel techniques have been employed in this field since 1993 and enable very high quality solutions to be found rapidly. Thus, in Sect. 2, we provide sample results which demonstrate that the multilevel approach when used in combination with a state-of-the-art (single-level) local search strategy can dramatically improve the asymptotic convergence in solution quality.

In Sect. 3, we then survey the use of multilevel techniques, both in the field of graph-partitioning and also their increasing use in other combinatorial optimisation problems. We also look at related ideas. In Sect. 4, we look at generic features and extract some guiding principles that might aid the application of the paradigm to other problems. Finally we summarise the chapter in Sect. 5.

2 Extended Example: the Graph Partitioning Problem

The k-way graph partitioning problem (GPP) can be stated as follows: given a graph $G(V, E)$, possibly with weighted vertices and/or edges, partition the vertices into k disjoint sets such that each set contains the same (or nearly the same) vertex weight and such that the **cut-weight**, the total weight of edges cut by the partition, is minimised. In combinatorial optimisation terms, the cut-weight is the objective function whilst balancing the vertex weight is a constraint (the balance constraint) and it is well known that this problem is NP-hard.

The GPP has a number of applications, most notably the partitioning of unstructured meshes for parallel scientific computing (often referred to as mesh partitioning).

2.1 Multilevel Graph Partitioning

The GPP was the first combinatorial optimisation problem to which the multilevel paradigm was applied and there is now a considerable volume of literature about multilevel partitioning algorithms which we survey in Sect. 3.1. Initially used as an effective way of speeding up partitioning schemes, it was soon recognised as, more importantly, giving them a more 'global' perspective [21], and has been successfully developed as a strategy for overcoming the localised nature of the Kernighan-Lin (KL) [26] and other optimisation algorithms.

Typically, multilevel implementations match and coalesce pairs of adjacent vertices to define a new graph and recursively apply this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned (possibly with a crude algorithm) and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. At each change of levels, the final partition of the coarser graph is used to give the initial partition for the next level down.

The use of multilevel combinatorial refinement for partitioning was first proposed by both Hendrickson & Leland [19] and Bui & Jones [10], inspired by Barnard & Simon [2], who used a multilevel numerical algorithm to speed up spectral partitioning.

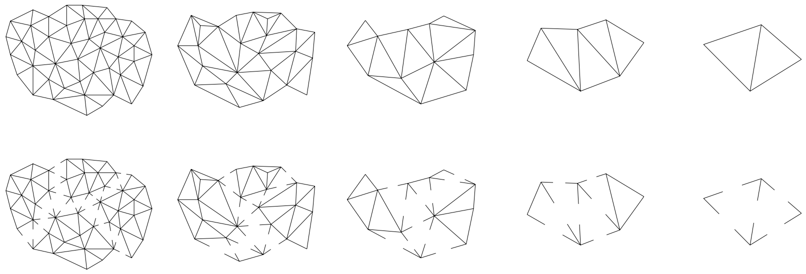


Fig. 1. An example of multilevel partitioning

Fig. 1 shows an example of a multilevel partitioning scheme in action. On the top row (left to right) the graph is coarsened down to 4 vertices which are (trivially) partitioned into 4 sets (bottom right). The solution is then successively extended and refined (right to left; each graph shows the final partition for that level). Although at each level the refinement is only local in nature, a high quality partition is still achieved.

Graph Coarsening

A common method to create a coarser graph $G_{l+1}(V_{l+1}, E_{l+1})$ from $G_l(V_l, E_l)$ is the edge contraction algorithm proposed by Hendrickson & Leland [19]. The idea is to find a maximal independent subset of graph edges, or a **matching** of vertices, and then collapse them. The set is independent if no two edges in the set are incident on the same vertex (so no two edges in the set are adjacent), and maximal if no more edges can be added to the set without breaking the independence criterion.

Having found such a set, each selected edge is collapsed and the vertices, $u_1, u_2 \in V_l$ say, at either end of it are merged to form a new vertex $v \in V_{l+1}$ with weight $||v|| = ||u_1|| + ||u_2||$. Edges which have not been collapsed are inherited by the child graph, G_{l+1}, and, where they become duplicated, are merged with their weight summed. This occurs if, for example, the edges (u_1, u_3) and (u_2, u_3) exist when edge (u_1, u_2) is collapsed. Because of the inheritance properties of this algorithm, it is easy to see that the total vertex weight remains the same, $||V_{l+1}|| = ||V_l||$, and the total edge weight is reduced by the sum of the collapsed edge weights.

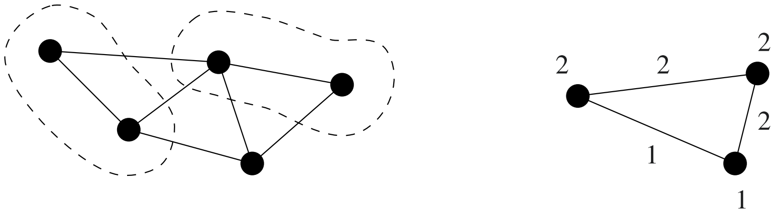


Fig. 2. An example of coarsening via matching and contraction

Fig. 2 shows an example where, on the left, two pairs of vertices are matched (indicated by a dotted line). On the right, the resulting coarsened graph is shown, with numbers illustrating the resulting vertex and edge weights (assuming that the original graph had unit weights).

A simple way to construct a maximal independent subset of edges is to create a randomly ordered list of the vertices and visit them in turn, matching each unmatched vertex with an unmatched neighbour (or with itself if no unmatched neighbours exist). Matched vertices are removed from the list. If there are several unmatched neighbours the choice of which to match with can be random, but it has been shown by Karypis & Kumar [21] that it can be beneficial to the optimisation to collapse the most heavily weighted edges.

As discussed below (Sect. 4.1), coarsening has the effect of **filtering** the solution space. To see this suppose that two vertices $u, v \in G_l$ are matched and coalesced into a single vertex $v' \in G_{l+1}$. When a refinement algorithm is subsequently used on G_{l+1} and whenever v' is assigned to one of the partition subsets, both u and v are also both being assigned to that subset. In this way the matching restricts a refinement algorithm working on G_{l+1} to consider only those configurations in the solution space in which u and v lie in the *same* subset, although the particular subset to which they are assigned is not specified at the time of coarsening. Since many vertex pairs are generally coalesced from all parts of G_l to form G_{l+1} this set of restrictions is equivalent to filtering the solution space and hence the surface of the objective function.

The Initial Partition

The hierarchy of graphs is constructed recursively until the number of vertices is smaller than some threshold and then an initial partition is found for the coarsest graph. At its simplest, the coarsening is terminated when the number of vertices in the coarsest graph is the same as the number of subsets required, k, and then vertex i is assigned to subset S_i. However, since the vertices of the coarsest graph are not generally homogeneous in weight, this does require some mechanism for ensuring that the final partition is balanced, i.e. each subset has (approximately) the same vertex weight. Various methods have been proposed for achieving this, commonly either by terminating the contraction so that the coarsest graph G_L still retains enough vertices, $|V_L|$, to achieve a balanced initial partition (i.e. so that typically $|V_L| \gg k$) [19, 21], or by incorporating load-balancing techniques alongside the refinement algorithm, e.g. [46].

Refinement

At each level, the partition from the previous level is extended to give an initial partition and then refined. Various refinement schemes have been successfully used including a variety of metaheuristics, which we survey below, Sect. 3.1. Most commonly, however, the refinement is based on the Kernighan-Lin (KL) bisection optimisation algorithm [26] which includes the facility (albeit limited) to enable it to escape from local minima. Recent implementations almost universally use the linear time complexity improvements (e.g. bucket sorting of vertices) introduced to partitioning by Fiduccia & Mattheyses [14]. For more details see one of the many implementations, e.g. [19, 23, 46].

In terms of the multilevel framework, the only requirements are that the scheme must be able to refine an existing partition (rather than starting from scratch) and must be able to cope with weighted graphs since, even if the original graph is not weighted, the coarsened graphs will all have weights attached to both vertices and edges because of the coarsening procedures.

Partition Extension

Having optimised the partition on a graph G_l, the partition must be extended onto its parent G_{l-1}. The extension algorithm is trivial; if a vertex $v \in V_l$ is in subset S_i then the matched pair of vertices that it represents, $v_1, v_2 \in V_{l-1}$, are also assigned to S_i.

2.2 Multilevel Results

To illustrate the potential gains that the multilevel paradigm can offer, we give some example results. These are not meant to be exhaustive in any way but merely give an indication of typical performance behaviour.

In [44], detailed tests are carried out to assess the impact of multilevel refinement on the GPP. Here we summarise those results.

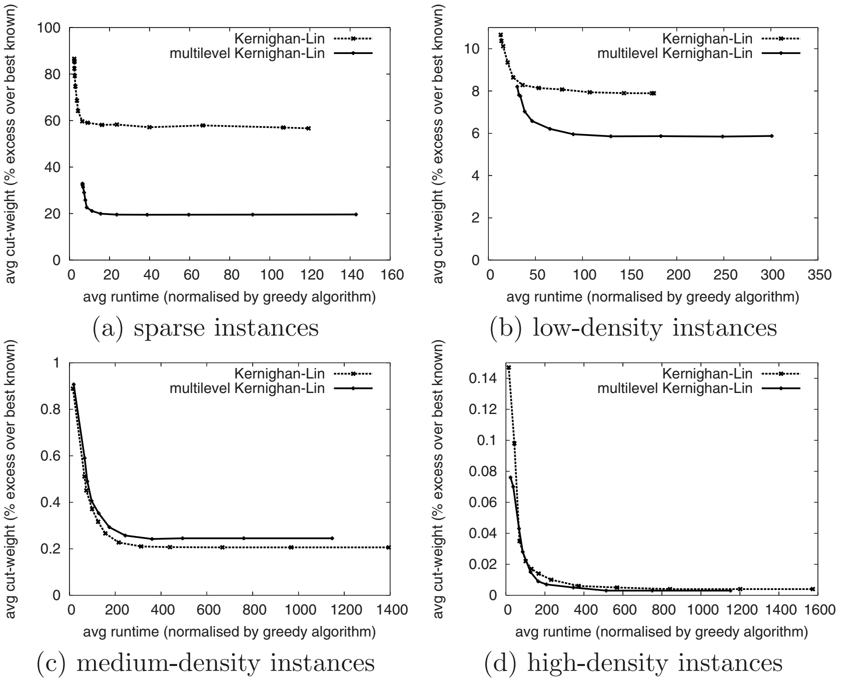
The experimental data consists of two test suites, one of which is a smallish collection of 16 sparse, mostly mesh-based graphs, drawn from a number of real-life applications, and often used for benchmarking. The other test suite consists of 90 instances, originally compiled to test graph-colouring algorithms and including a number of randomly generated examples. Although perhaps not representative of partitioning applications, they reveal some interesting results. This colouring test suite is further subdivided into 3 density classes; low (under 33 %) with 58 out of 90 instances, medium (between 33 % and 67 %) with 23 instances and high (over 67 %) with just 9 instances.

In this context, the density, or **edge density**, of a graph, $G(V, E)$, is defined as the percentage of all possible edges and given by $2|E|/[|V| \cdot (|V| - 1)]$, so that a **complete** graph (where every vertex is adjacent to every other), with $|V| \cdot (|V| - 1)/2$ edges, has a density of 100 %.

The tests compare the JOSTLE implementation of the Kernighan-Lin (KL) algorithm [46] against its multilevel counterpart (MLKL). As discussed in [44], and similar to most local search schemes, the KL algorithm contains a parameter, λ, known as the **intensity**, which allows the user to specify how long the search should continue before giving up (specifically how much effort the search should make to escape local minima). When $\lambda = 0$ the refinement is purely greedy in nature, whereas if $\lambda \to \infty$ the search would continue indefinitely.

To assess the algorithms, we measure the run-time and solution quality for a chosen group of problem instances and for a variety of intensities. We then normalise these values with reference solution quality and run-time values and finally plot averaged normalised solution quality against averaged normalised run-time for each intensity value.

Fig. 3(a) shows the results for the sparse suite and the dramatic improvement in quality imparted by the multilevel framework is immediately clear. Even for purely greedy refinement (i.e. the extreme left-hand point on either curve) the MLKL solution quality is far better than KL and it is results like



(a) sparse instances

(b) low-density instances

(c) medium-density instances

(d) high-density instances

Fig. 3. Plots of convergence behaviour for the partitioning test suites

these that have helped to promote multilevel partitioning algorithms to the status they enjoy today.

Figs. 3(b)–(d) meanwhile show the partitioning results for the colouring test suite. Fig. 3(b) more or less confirms the conclusions for the sparse results and although the curves are closer together, MLKL is the clear winner. For the medium and high-density examples however, it is a surprise (especially considering the widely accepted success of multilevel partitioning) to find that these conclusions are no longer valid. For the high-density instances, Fig. 3(d), MLKL is still the leading algorithm, although only very marginally. However for the medium-density results, Fig. 3(c), MLKL fails to achieve the same performance as KL and the multilevel framework appears to actually hinder the optimisation. We discuss this further in the following section.

2.3 Iterated Multilevel Results

Although the medium density results are disappointing, in fact a simple resolution does exist which works by reusing the best partitions that have been found. Indeed, given any partition of the original problem we can carry out solution-based **recoarsening** by insisting that, at each level, every vertex v matches with a neighbouring vertex in the same set. When no further coarsening is possible this will result in a partition of the coarsest graph with the same cost as the initial partition of the original. Provided the refinement algorithms guarantee not to find a worse partition than the initial one, the multilevel refinement can then guarantee to find a new partition that is no worse than the initial one.

This sort of technique is frequently used in graph-partitioning for dynamic load-balancing, e.g. [36, 45], although if the initial partition is unbalanced, the quality guarantee can be lost in satisfying the balance constraint. However it can also be used to find very high quality partitions, albeit at some expense, and the multilevel procedure can be iterated via repeated coarsening and uncoarsening. At each iteration the current best solution is used to construct a new hierarchy of graphs, via recoarsening, and guarantees not to find a worse solution than the initial one. However, if the matching includes a random factor, each iteration is very likely to give a different hierarchy of graphs to previous iterations and hence allows the refinement algorithm to visit different solutions in the search space.

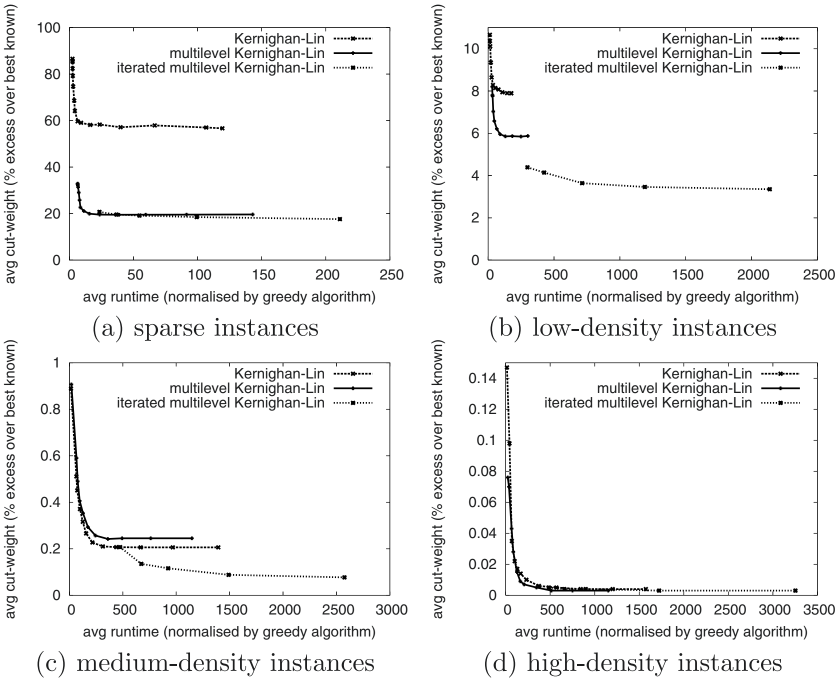
We refer to this process as an **iterated multilevel algorithm** (see Sect. 3.1 for further discussion). It requires the user to specify an additional intensity parameter, namely the number of failed outer iterations (i.e. the number of times the algorithm coarsens and uncoarsens the graph without finding a better solution).

Fig. 4 illustrates the results for the iterated multilevel algorithm (IMLKL) alongside the MLKL and KL results for the two test suites. These plots contain exactly the same information about MLKL and KL as Fig. 3, only here it is more compressed because of the long IMLKL run-times.

In fact the results for the sparse and high-density instances are not so interesting; for the sparse suite, Fig. 4(a), IMLKL more or less continues the MLKL curve in Fig. 3(a) with a few percentage points improvement and very shallow decay, whilst for the high-density instances, Fig. 4(d), IMLKL does not appear to offer much improvement at all. However for the low and medium-density subclasses, in Figs. 4(b) and 4(c) respectively, the asymptotic performance offered by IMLKL is impressive and worthy of further and more thorough investigation. In both cases IMLKL dramatically improves on MLKL and, for the medium-density instances, even appears to overcome the shortcomings of MLKL and significantly improves on the KL results.

Summary

It is clear that the multilevel framework can enormously benefit the performance of a state-of-the-art local search scheme. Its task is made more difficult if the graphs are dense, but nevertheless, via the simple technique of recoarsening, an iterated multilevel scheme seems to be able to overcome this.



(a) sparse instances

(b) low-density instances

(c) medium-density instances

(d) high-density instances

Fig. 4. Plots of convergence behaviour including iterated multilevel partitioning results

3 Survey of Multilevel Implementations

In this section we survey existing literature about multilevel implementations in a variety of contexts.

The survey is broken up into three categories: graph-partitioning, other combinatorial problems and related ideas. As mentioned above, multilevel graph-partitioning has been widely investigated and so Sect. 3.1 affords a number of general observations about multilevel strategies. Indeed it is because of its success in this area that the multilevel paradigm has started to be utilised elsewhere and so Sect. 3.2 takes a look at the increasing numbers of multilevel algorithms for other combinatorial problems. Finally Sect. 3.3 considers similar concepts such as smoothing.

3.1 Survey of Multilevel Graph-Partitioning

As has been mentioned in Sect. 2, graph-partitioning has been by far the most common application area for multilevel refinement and it is now a *de facto* standard technique to use. Furthermore, it has been tested with a wide variety of metaheuristic and local refinement schemes and gives a good indication of how robust the multilevel framework can be. In addition, a number of enhancements and extensions have appeared, all of which demonstrate the flexibility of the paradigm.

Table 1. Multilevel graph-partitioning algorithms and variants

Feature of interest	**References**
Multilevel metaheuristics	
Ant Colony Optimisation	Langham & Grant [28]; Korošec *et al.* [27]
Cooperative Search	Toulouse *et al.* [40]
Genetic Algorithms	Kaveh & Rahimi [25]; Soper *et al.* [38]
Tabu Search	Battiti *et al.* [3]; Vanderstraeten *et al.* [41]
Simulated Annealing	Romem *et al.* [32]; Vanderstraeten *et al.* [41]
Multilevel enhancements	
Coarsening Homogeniety	Abou-Rjeili & Karypis [1]; Gupta [17]
Constraint Relaxation	Walshaw & Cross [46]
Inexact Coarsening	Bui & Jones [10]
Recoarsening	Gupta [17]; Toulouse *et al.* [40]; Walshaw [44]
Related problems	
Aspect Ratio Optimisation	Vanderstraeten *et al.* [41]; Walshaw *et al.* [48]
Hypergraph Partitioning	Karypis & Kumar [24]
Multi-constraint/objective	Karypis & Kumar [22]; Schloegel *et al.* [37]
Network Mapping	Walshaw & Cross [47]

Table 1 shows a summary of some of the implementations. This list is far from comprehensive; indeed a survey of multilevel graph-partitioning algorithms could fill a paper in itself. However, here the aim is to highlight a variety of interesting cases and we focus on three different classes: **multilevel metaheuristics**, **multilevel enhancements** and **related problems** that have been successfully addressed by multilevel algorithms.

Multilevel Metaheuristic Refinement Schemes

A first point to note is the variety of optimisation algorithms with which the multilevel framework has been used. Apart from the commonly used problem-specific algorithm of Kernighan & Lin (KL) [26], mentioned above, a wide variety of well-known metaheuristics have been applied. These are normally used to refine the solution at each level and, provided they can operate on weighted graphs, essentially require no modifications. Specifically, ant colony optimisation [28, 27], cooperative search [40], evolutionary/genetic algorithms [25], simulated annealing [32, 41] and various flavours of tabu search [3, 41], have been successfully applied, generally with great success.

Although space precludes a more detailed study, positive comments by authors are common. For example, Vanderstraeten *et al.*, referring to multilevel simulated annealing and multilevel tabu search, state that the '*contraction procedure not only speeds up the ... partitioning method, but also results in better mesh decompositions*' [41]. Meanwhile, Korošec *et al.*, in their implementation of multilevel ant colony optimisation, state that '*with larger graphs, which are often encountered in mesh partitioning, we had to use a multilevel method to produce results that were competitive with the results given by other algorithms*' [27].

Multilevel Enhancements

Perhaps of more interest, especially in terms of applying the multilevel framework to completely different problem areas, is the development of a number of multilevel enhancements which can be used to improve performance still further.

For example, a number of authors have noted that the coarsest graphs can become very inhomogeneous in terms of vertex weight [1, 17]. This is particularly noticeable for so-called **scale-free graphs**[1], which often arise when modelling links between web pages. In the worst cases, the coarsening may start to form star graphs, with one very large 'super-vertex' attached to many small vertices. Most coarsening schemes cannot cope with this type of graph and will only manage to collapse one edge at each level (i.e. a multilevel scheme for a star graph with 1,000 vertices will have 1,000 levels) resulting in very poor runtime performance. As a result, coarsening algorithms have been introduced to ensure that the vertex weights stay relatively homogeneous, usually by merging clusters of two (or more) vertices which are not adjacent, e.g. [1, 17]. We refer to this as **coarsening homogeneity** and although it might seem very specific to partitioning, a similar idea has recently been used in a multilevel algorithm for the Vehicle Routing Problem [30] (see below, Sect. 3.2).

Another multilevel enhancement is developed in [46], where it is demonstrated that by relaxing the balancing constraint (i.e. that the subsets are all of the same size) at the coarsest graph levels, and tightening it up level by level, higher quality results can be found. This idea of **constraint relaxation** has also been used for the Vehicle Routing Problem [30] (see also Sect. 3.2).

In one of the first multilevel graph partitioning implementations [10], Bui & Jones collapsed edges and merged vertices without taking account of vertex or edge weights (see Sect. 2.1). As a result, the coarsened graph did not represent the original problem with complete accuracy (e.g. since an edge in a coarsened graph might equally represent one or one hundred edges of the

[1] Typically in scale-free graphs the degree distribution follows a power-law relationship so that some vertices are highly connected, although most are of low degree.

original graph and so it is no longer obvious which cut edges to prefer). We refer to this as **inexact** coarsening and although it is not necessarily recommended, especially in the case of graph partitioning where it is fairly easy to create **exact** coarse representations of the problem, it is interesting to see that this kind of coarsening could still '*dramatically improve the performance of the Kernighan-Lin and greedy algorithms*' [10]. This is especially encouraging for other combinatorial problems, where it may not even be possible to create an exact coarsened representation.

Finally, as we have seen in Sect. 2.3, a powerful, and very straightforward technique is to use solution-based **recoarsening**. Indeed this feature can be used in two different ways: either, as in [44], as an outer loop which treats the multilevel partitioner almost as a black-box and iterates its use; or, as in [17], where the framework cycles up and down through the coarser levels to find a very high quality initial solution for very little computational expense (since the coarse graphs are very small). In Sect. 3.2, we classify these as either **external** or **internal** recoarsening, respectively.

Related Partitioning Problems

The multilevel paradigm has also been successfully extended to a range of partitioning problem variants.

In particular, the development of successful multilevel hypergraph[2] partitioning schemes has sparked a great deal of interest in the VLSI/circuit partitioning community, e.g. [11]. Indeed, Karypis and Kumar, pre-eminent developers of such schemes state that '*the power of [their algorithm] is primarily derived from the robustness of the multilevel paradigm that allows the use of a simple k-way partitioning refinement heuristic instead of the [commonly used] $O(k^2)$ complexity k-way FM refinement*' [24].

Similarly, in [22, 37], multilevel graph partitioning techniques have been successfully modified to deal with other, more generalised graphs such as those which have multiple vertex weights (multi-constraint problems) or multiple edge weights (multi-objective problems).

Finally and more specifically, multilevel graph partitioning techniques have been shown to adapt well to modified objective functions. For example, in [41, 48], the aspect ratio (shape) of the subdomains is optimised as an alternative to the cut-weight. Meanwhile, [47] considers the problem mapping graphs onto parallel networks with heterogeneous communications links – a successful mapping is then one in which adjacent subsets generally lie on 'adjacent' processors.

An important point about these last two problems is that, in both cases, the coarsening scheme requires almost no modifications and the new problem class is optimised solely by changes to the objective function (and hence

[2] A hypergraph is a generalisation of a graph in which an edge describes a relationship between an arbitrary number of vertices.

the refinement scheme). This makes an interesting point: using the multilevel framework, the *global* layout of the final partition can be radically changed just by modifying the *local* cost function. This corroborates the suggestion that the multilevel framework adds a global perspective to (local) partitioning schemes.

3.2 Survey of Multilevel Combinatorial Optimisation

Apart from its wide uptake in the graph-partitioning area, multilevel refinement schemes are increasingly appearing for other combinatorial problems. Typically, the results are early indicators and the ideas not yet fully developed, but in most applications authors report successes, either in speeding up a metaheuristic or local search algorithm, or in improving the quality of its results. Furthermore, it is often seen as an appropriate method for addressing larger-scale problem instances which, hitherto, have not been tractable.

To survey these implementations, we first describe each one briefly, highlighting the multilevel techniques that it employs, and then, at the end of this section, attempt to summarise and classify the approaches.

Biomedical Feature Selection. In [29], Oduntan investigates multilevel schemes for feature selection and classification in biomedical data. Two possible coarsening strategies are suggested, one of which clusters decision variables, in a manner akin to multilevel partitioning, and represents them as a single variable and another, 'feature pre-setting', which recursively excludes features (decision variables) from each level. This second approach is unlike most multilevel implementations in that it is essentially evaluating only **partial** solution spaces at each level (as opposed to **restricted** spaces), but, because of the computational cost of maintaining an exact cost evaluation for the clustering version, random feature pre-setting is used for most of the experimentation. Nonetheless, the multilevel technique is '*outstandingly better*' and '*generates higher and more stable average classification accuracies*' than the other evaluated feature selection techniques.

Capacitated Multicommodity Network Design (CMND). In [12], Crainic *et al.* implement a multilevel cooperative search algorithm for the CMND problem. The coarsening is achieved via fixed edges which are computed from an initial solution calculated by a tabu search (solution-based coarsening) and a form of iterated multilevel refinement is performed by propagating elite solutions up through the multilevel hierarchy (as well as down). The authors discuss the issue of how rapidly to coarsen the problem and the compromise between a high number of levels and large coarsening factors, and cite this as a future research challenge. Nonetheless, the initial results are very encouraging and experiments on a set of benchmark problems showed that the approach yields '*solutions comparable to those obtained by the current best metaheuristics for the problem*' and that the multilevel framework '*appears to perform better when the number of commodities is increased (which, normally, increases the difficulty of the problem)*'.

Covering Design. Another problem that has recently been investigated with multilevel cooperative techniques is that of covering design in which the aim is to minise the number subsets of a given size which 'cover' the problem instance (i.e. so that every member of the problem instance is contained in at least a given number of the subsets). This problem is extremely difficult to solve to optimality and in [13], Dai *et al.* use multilevel cooperative tabu search to improve upper bounds (in order, for example, to improve the efficiency of enumeration techniques such as branch-and-cut). They use solution-based re-coarsening and a 'direct interpolation' operator, which projects elite solutions found at the coarsest level directly down to the the original problem. The results are impressive – the methods succeed in computing new upper bounds for 34 out of 158 well-known benchmark problems – and the authors confirm that, for this problem, '*multilevel search and cooperation drastically improve the performance of tabu search*'. Furthermore, since similar cost functions and neighbourhood structures exist in other problems such as packing design, t-design and feature selection in bio-informatics and data mining, the authors suggest that '*the main framework of the re-coarsening and direct interpolation operations can be applied in a multilevel cooperative search algorithm for such problems*'.

DNA Sequencing by Hybridization. Blum & Yábar use a multilevel framework to enhance an ant colony algorithm for the computational part of this problem [5, 6]. It is closely related to the selective travelling salesman problem, a variant in which only a subset of the vertices are visited, and the authors use a similar coarsening approach to that mentioned below, in which edges are fixed to create tour segments or paths. In experiments, the algorithms (both single- and multilevel) solve all benchmark problems to optimality, but the multilevel version substantially reduces the computation time of the ant colony optimisation, and is between 3 to 28 times faster. Furthermore the authors believe that for larger problem instances, the multilevel framework '*may also improve the quality of the obtained solutions*'.

Graph Colouring Problem. In [44], Walshaw discusses the development of multilevel graph colouring schemes. The coarsening uses similar vertex matching to the graph partitioning, with one important difference: since the objective here is to colour each vertex differently to any of its neighbours, the matching is carried out between *non-adjacent* vertices (so that all vertices in a coarsened cluster can be given the same colour). The experimentation illustrates that, for sparse and low-density graphs at least, the multilevel paradigm can either speed up or even give some improvements in the asymptotic convergence of well known algorithms such as tabu search. However, the impact of the multilevel framework is less impressive than for other problems and the paper concludes with a number of suggestions that might help to improve the results.

Graph Ordering. One problem area that has received a fair amount of interest from multilevel practitioners is that of graph ordering. This is almost certainly because the applications from which it arises (such as the efficient solution of linear systems) are closely related to those in which graph partitioning occurs (e.g. parallel scientific computing) and the two problems have some overlap.

For example, Boman & Hendrickson describe the use of a multilevel algorithm for reducing the envelope of sparse matrices [7], a technique which aims to place all the non-zeroes as close as possible to the diagonal of a matrix and which can help to speed up the solution of sparse linear systems. They report good results, better than some of the commonly used methods, although they conclude that their scheme would probably be better if combined with a state-of-the-art local search algorithm. This conclusion is confirmed by Hu & Scott who have also developed a multilevel method for the same problem and which uses such a scheme, the hybrid Sloan algorithm, on the coarsest graph only [20]. They report results which are of similar quality to the standalone hybrid Sloan algorithm (i.e. as good as the best known results) but which, on average, can be computed in half the time.

Both of these approaches use a coarsening algorithm based on vertex matching, and similar to those used for multilevel graph partitioning schemes. More recently, however, Safro *et al.* discuss the use of a multilevel algorithm for the linear arrangement problem [35], which has the aim of ordering the vertices of a graph so that the sum of edge lengths in the corresponding linear arrangement is minimised, a problem with several diverse applications. In this work, the coarsening is described as 'weighted aggregation' (as opposed to 'strict aggregation') in which each vertex can be divided into fractions which are then merged. In other words, a coarsened vertex will in general consist of fractions of several vertices from the original graph, rather than a discrete set of two (or more). In experiments, the authors found that their algorithm generally outperformed standard methods on most test cases, although it had some problems on uniform random graphs. This echoes some of the findings for multilevel graph partitioning which seems to prefer certain types of graph.

Finally, it is worth noting that unlike most problems discussed in this section, the schemes described in all three papers produce inexact representations of the problem in coarsened spaces. This is due to the nature of the graph ordering problem and may be inevitable, but see the classification section below (page 277) for further discussion.

Travelling Salesman Problem (TSP). The TSP is a prototypical (arguably *the* prototypical) combinatorial problem and consequently is one of the first on which new algorithms are tested.

In the earliest multilevel approach to the TSP, Saab, using a technique described as 'dynamic contraction', applied algorithms to recursively construct chains of cities and then implemented a simple local search algorithm [34]. The results demonstrate that '*the improvement due to contraction*' (in our context, the improvement due to the multilevel framework) '*is significant in all cases and ranges from 33.1 % to 108.4 %*' although possibly, as the author acknowledges, this is because the local search is a very basic heuristic in this case. This paper is also interesting because it proposes dynamic contraction as a generic strategy for combinatorial problems and, apart from the TSP, also demonstrates the approach on the graph bisection problem.

Subsequently, Walshaw independently applied a similar coarsening approach using fixed edges to build a multilevel version of the well-known chained Lin-Kernighan algorithm (together with a more powerful variant, Lin-Kernighan-Helsgaun) [42, 44]. The multilevel results showed significant improvement on the single-level versions and are '*shown to enhance considerably the quality of tours for both the Lin-Kernighan and Chained Lin-Kernighan algorithms, in combination the TSP champion heuristics for nearly 30 years*'.

Finally, in [8], Bouhmala combined a multilevel approach with a genetic algorithm adapted for the TSP. Interestingly, and in contrast to the two previous approaches, the coarsening is based on merging pairs of cities and averaging their coordinates. Although this is a simpler approach to fixing edges, it means that any coarsened version does not represent the original problem instance exactly and, because of the modified coordinates, the cost of a solution in one of the coarsened spaces is not the same as the cost if that solution is projected back to the original. Nonetheless, the results indicate that '*the new multilevel construction algorithm*' (i.e. just using the coarsening without any refinement) '*produces better results than the Clark-Wright algorithm, and that the multilevel genetic algorithm was found the clear winner when compared to the traditional genetic algorithm*'.

Vehicle Routing Problem (VRP). A problem closely related to the TSP (although significantly complicated by additional side constraints) is the VRP and in [30], Rodney *et al.* demonstrate that it too is susceptible to multilevel techniques. The coarsening scheme is similar to that used previously for the TSP [34, 42] and recursively fixes edges into potential routes (whilst additionally respecting vehicle capacity constraints).

The resulting framework is tested with a variety of problem-specific local search heuristics and it is found that, for a test suite of well-known benchmark problems, the best multilevel solutions are on average within 7.1 % of optimality as compared with 18.4 % for the single-level version of the algorithm. Although this is not quite so good as results from a state-of-the-art genetic algorithm code, it has since been shown that an iterated version (see below and Sect. 2.3) of the multilevel scheme finds solutions within 2.6 % of optimal, comparable with the genetic algorithm and considerably faster (especially as the problem size increases).

As part of this work on the VRP, two multilevel enhancement techniques are developed and tested.

The first employs **constraint relaxation** (referred to in [30] as 'route overloading'), as discussed in Sect. 3.1. In the context of the VRP, this means allowing routes in the coarser level to exceed the vehicle capacity constraint. This is then tightened up gradually as the multilevel scheme approaches the finer levels and, ultimately, the original problem.

The second enchancement (referred to as 'segment balancing') tried to balance the fixed edge segments in terms of demand and cost in order to provide more **homogeneous coarsening** (see also Sect. 3.1).

Combined together, these two enhancements provide the best results and are useful ideas in the generic application of multilevel techniques.

Other Problems. The above list is not comprehensive and, for example, multilevel techniques have been applied to other problems such as satisfiability [33] and the still life problem [15]. Unfortunately, neither example showed significant benefits, either because the work is only in its preliminary stages and still requires further development, or possibly because the problem is simply not suitable for multilevel coarsening. In either case, it seems appropriate to exclude it from generalisations about the multilevel framework pending further investigation (since, as is common with heuristics, we merely seek evidence that multilevel techniques can be used successfully, rather than proof that they will always be successful). However, it is worth noting, as is discussed in [44], that although the multilevel paradigm seems to assist in most problems to which it has been applied, the benefits are not necessarily universal.

Summary and Classification

Because of the flexibility of the paradigm, multilevel schemes can appear in a variety of guises. Nonetheless, it is still possible to draw out some common features from the schemes listed above. Table 2 summarises the above approaches and aims to help classify them, both by the optimisation algorithm they use for refinement, and in terms of the multilevel techniques they employ: the coarsening strategy (**coarsen**); whether they produce **exact** coarse representations of the problem; and what type, if any, of recoarsening they use (**recoarsen**).

One of the most obvious classifications then is the **refinement** algorithm used to optimise the solution at each level. Often these are special-purpose or **problem-specific**, but in many cases a 'standard' metaheuristic is used (although it should be understood that in the context of combinatorial optimisation there is no such thing as a standard metaheuristic and all will require some modification or, at the very least, parameter tuning).

A second way of classifying the schemes is by their coarsening strategy (the column headed **coarsen**). Although this is always problem-specific, it will generally depend on whether the problem requires a solution based on an ordering or a classification of the vertices. In the case of ordering problems (such as the travelling salesman and vehicle routing problems), it is usual to recursively fix edges between vertices, a **path**-based coarsening. For classification problems (such as graph partitioning and graph colouring), it is usual to merge vertices, a **set**-based approach.

Although it might seem that these are very similar operations, in practice path-based coarsening generates a set of fixed path-segments in the coarsened

Table 2. Multilevel algorithms for combintorial problems

Problem, [reference]	refinement	coarsen	exact	recoarsen
Biomedical feature selection [29]	Tabu Search	set$^+$	yes	
Capacitated multicommodity network design [12]	Cooperative Search	path	yes	internal
Covering design [13]	Cooperative Search	set	yes	internal
DNA sequencing [5, 6]	Ant Colonies	path	yes	
Graph colouring [44]	Tabu Search	set	yes	
Graph ordering (envelope) [7]	problem-specific	set	no	
Graph ordering (wavefront) [20]	problem-specific	set	no	
Graph ordering (linear) [35]	Simulated Annealing	set*	no	external
Travelling salesman [34]	problem-specific	path	yes	
Travelling salesman [42]	problem-specific	path	yes	
Travelling salesman [8]	Genetic Algorithms	set	no	
Vehicle routing [30]	problem-specific	path	yes	external

spaces and each path-segment contains an ordering of its internal vertices. Meanwhile, set-based coarsening usually generates a weighted graph; edges that are internal to each merged set of vertices are collapsed.

Note that two variant approaches were also employed: in biomedical feature selection [29], as an alternative to set-based clustering the implementation also coarsens by recursively excluding decision variables and hence evaluating only partial solutions, denoted as 'set$^+$' in the table; meanwhile the multilevel approach to the linear graph ordering problem [35] uses an aggregated set-based approach, denoted as 'set*' in the table – see above for details.

A third way of comparing schemes is by considering whether coarsened versions of the problem instance give an **exact** or an **inexact** approximation of the original solution space. In this context, by exact we mean that evaluation of the objective function for a solution of a coarsened space is exactly the same as if that solution were extended back into the original space and the objective function evaluated there.

Note that even for problems where exact representation is possible, it is usually also possible to generate inexact representations. As we have seen in Sect. 3.1, at its simplest this can just be by ignoring vertex and or edge weights when coarsening graphs for partitioning purpose. Alternatively, if nearby vertices are merged to create a coarsened version of a travelling salesman problem (as mentioned above), then the inter-vertex distances will be incorrect and hence the coarsed problem is inexact.

Usually, employing an exact coarsening is more accurate but will involve some modifications to the optimisation scheme (for example, to take account of vertex weights or fixed edges). Conversely, an inexact coarsening is easier to implement and the optimisation scheme can often be used without modification.

Interestingly, all of the approaches to graph ordering use set-based coarsening and merge vertices, rather than employing the path-based fixed edges that might be expected from an ordering problem, and as a result they all end up with an inexact representation of the problem. It is intriguing to ask whether an exact representation might be more effective in these cases.

A final method we have used to classify the work is to look at whether the approach employs **recoarsening** (the column labelled **recoarsen**), possibly as part of an iterated multilevel scheme. As we have seen in Sect. 2.3 this can be extremely easy to implement and highly effective. However we also distinguish here between **external** and **internal** recoarsening.

In this context, internal recoarsening can propagate elite solutions *up* through the multilevel hierarchy (as well as down) and the scheme may only ever carry out one refinement phase on the original uncoarsened problem (i.e. once it believes that a very high quality solution has been found at the coarser levels).

Conversely, external recoarsening takes place as part of repeated multilevel cycles (see Fig. 6), in an iterated multilevel scheme (in this case refinement will take place on the original uncoarsened problem at the end of every cycle). Of course, the distinction is not completely clear and it is possible for a scheme using internal recoarsening to return repeatedly to the original problem and thus resemble an iterated scheme.

3.3 Other Related Work

Because multilevel algorithms are well-known in many areas of mathematics other than combinatorial optimisation, there is a large body of literature which could be said to be related to the methods presented here. In particular, **multigrid** methods are often used to solve partial differential equations on a hierarchy of grids or meshes, whilst **multi-scale** or **multi-resolution** methods typically address continuous problems by viewing them at a number of different levels. However, because of the nature of the problems the operators which transfer solutions from one scale to another are necessarily somewhat different from the discrete techniques discussed here. For interested readers a good start is Brandt's review paper [9] and for an analysis of the fundamental similarities of all these ideas see Teng [39].

Another related idea is that of **aggregation** which can be used either to approximate an intractable problem with a smaller one or, sometimes, to provide decision-makers with models at different levels of detail. In that sense, it tends to deal more with the modelling aspects of optimisation problems (as opposed to finding a solution for a given model). However, it is certainly true that, when combined with **disaggregation** there is a degree of crossover with multilevel ideas. For further information see the survey paper of Rogers *et al.* [31].

An idea particularly related in scope and design to the principles behind multilevel refinement is the search space smoothing scheme of Gu and Huang

[16]. This uses recursive smoothing (analogous to recursive coarsening) to produce versions of the original problem which are simpler to solve. Thus in the example application Gu and Huang apply their technique to the TSP by forcing the inter-city edges to become increasingly uniform in length at each smoothing phase (if all edges between all cities are the same length then every tour is optimal). The obvious drawback is that each smoothing phase distorts the problem further (so that a good solution to a smoothed problem may not be a good solution to the original). In addition, the smoothed spaces are the same size as the original problem, even if the solution is potentially easier to refine, and hence may be equally as expensive to optimise. By contrast, exact multilevel coarsening filters the solution space (although with the obvious drawback that the best solutions may be removed from the coarsened spaces) and so the coarsened spaces are smaller and hence can be refined more rapidly (even inexact coarsening reduces the size of the space although strictly it does not filter it). It is also unclear whether search space smoothing is as general as coarsening and hence whether it could be applied to problems other than the TSP.

Finally, multilevel combinatorial optimisation is also closely related to developments in various semi-discrete optimisation problems. For example, it has been applied, with great success, to force-directed (FD) graph drawing. This is not a combinatorial problem, since the optimisation typically minimises a continuous energy function, but it does share some of the characteristics. Until recently FD methods were generally limited to small, sparse graphs, typically with no more than 1,000 vertices. The introduction of the multilevel framework, however, extended the size of graphs to which they could successfully be applied by several orders of magnitude, again through the 'global' improvement given by the multilevel scheme [18, 43].

4 Generic Analysis of the Multilevel Framework

In this section we draw together common elements of the examples in the previous sections. We give an explanation for the strengths of the multilevel paradigm and derive some generic guidelines for future attempts at other combinatorial problems.

4.1 Multilevel Dynamics

As we have seen, the multilevel paradigm is a simple one, which at its most basic involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found and then iteratively refined, usually with a local search algorithm, at each level in reverse order.

Considered from the point of view of the hierarchy, a series of increasingly coarser versions of the original problem are being constructed. It is hoped that each problem P_l retains the important features of its parent P_{l-1} but the (usually) randomised and irregular nature of the coarsening precludes any rigorous analysis of this process.

On the other hand, viewing the multilevel process from the point of view of the objective function and, in particular the hierarchy of solution spaces, is considerably more enlightening. Typically the coarsening is carried out by matching groups (usually pairs) of solution variables together and representing each group with a single variable in the coarsened space.

Previously authors have made a case for multilevel schemes (and in particular partitioning) on the basis that the coarsening successively *approximates* the problem with smaller, and hence easier to solve, solution spaces.

In fact it is somewhat better than this; as we have seen in the discussion about coarsening for the graph-partitioning problem (GPP), Sect. 2.1, provided that the coarsening is exact, it actually *filters* the solution space by placing restrictions on which solutions the refinement algorithm can visit.

A similar argument can be made for path-based coarsening algorithms, such as those employed for the TSP [44], and in a more general sense, we can think about combining decision variables so that changing one decision variable in a coarsened space is equivalent to changing several in the original solution space. In all cases, provided that the coarsening is exact, then coarsening has the effect of filtering the solution space.

Furthermore, an investigation of how the filtering actually performs for the GPP & TSP is carried out in [49] and it is shown that typically the coarsening filters out the higher cost solutions at a much faster rate than the low cost ones, especially for sparse and low-density problems.

We can then hypothesise that, if the coarsening manages to filter the solution space so as to gradually remove the irrelevant high cost solutions, the multilevel representation of the problem combined with even a fairly simple refinement algorithm should work well as an optimisation strategy. And when combined with sophisticated metaheuristics, it can be very powerful indeed.

On a more pragmatic level this same process also allows the refinement to take larger steps around the solution space (e.g. for graph partitioning, rather than swapping single vertices, the local search algorithm can swap whole sets of vertices as represented by a single coarsened vertex). This may be why the strategy still works even if the coarsening is inexact.

One further general observation about multilevel dynamics, that can be drawn from experimental evidence, is that the multilevel framework often seems to have the effect of stabilising the performance of the local search schemes. In particular, for the graph partitioning and travelling salesman problems (and the GCP at low intensities) the multilevel versions appear to have much lower variation in solution quality (in terms of the standard deviation of randomised results) [44]. However, it is not clear why this should be the case.

4.2 A Generic Multilevel Framework

To summarise the paradigm, multilevel optimisation combines a coarsening strategy together with a refinement algorithm (employed at each level in reverse order) to create an optimisation metaheuristic. Fig. 5 contains a schematic of this process in pseudo-code (here P_l refers to the coarsened problem after l coarsening steps, C_l is a solution of this problem and C_l^0 denotes the initial solution).

```
multilevel refinement( input problem instance $P_0$ )
begin
    $l \leftarrow 0$        // level counter
    while (coarsening)
        $P_{l+1} \leftarrow$ coarsen( $P_l$ )
        $l \leftarrow l+1$
    end
    $C_l =$ initialise( $P_l$ )
    while ($l > 0$)
        $l \leftarrow l-1$
        $C_l^0 \leftarrow$ extend( $C_{l+1}$, $P_l$ )
        $C_l \leftarrow$ refine( $C_l^0$, $P_l$ )
    end
    return $C_0$   // solution
end
```

Fig. 5. A schematic of the multilevel refinement algorithm

The question then arises, how easy is it to implement a multilevel strategy for a given combinatorial problem?

First of all let us assume that we know of a refinement algorithm for the problem, which refines in the sense that it attempts to improve on existing solutions. If no such refinement algorithm exists (e.g. if the only known heuristics for the problem are based on construction) it is not clear that the multilevel paradigm can be applied.

Typically the refinement algorithm will be either a problem-specific algorithm, or a metaheuristic, and it must be able to cope with any additional restrictions placed on it by the coarsening (e.g. for the set-based coarsening, the graphs are always weighted whether or not the original is; for path-based, the refinement must not change fixed edges in the coarsened levels).

To implement a multilevel algorithm, given a problem and a refinement strategy for it, we then require three additional basic components: a coarsening algorithm, an initialisation algorithm and an extension algorithm (which takes the solution on one problem and extends it to the parent problem). It is difficult to talk in general terms about these requirements, but the existing examples suggest that the extension algorithm can be a simple and obvious reversal of the coarsening step which preserves the same cost.

The initialisation is also generally a simple canonical mapping where by canonical we mean that a solution is 'obvious' (e.g. GPP – assign k vertices one each to k subsets; GCP – colour a complete graph; TSP – construct a tour to visit 2 cities) and that the refinement algorithm cannot possibly improve on the initial solution at the coarsest level (because there are no degrees of freedom).

Coarsening

This just leaves the coarsening which is then perhaps the key component of a multilevel implementation. For the existing examples three principles seem to hold:

- The number of levels need not be determined *a priori* but coarsening should cease when any further coarsening would render the initialisation degenerate.
- A solution in any of the coarsened spaces should induce a legitimate solution on the original space. Thus, at any stage after initialisation the current solution could simply be extended through all the problem levels to achieve a solution of the original problem.
- Ideally, any solution in a coarsened space should have the same cost with respect to the objective function as its extension to the original space (i.e. the coarsening is exact): this requirement ensures that the coarsening is truly filtering the solution space. However, the paradigm does seem to work well even if this is not the case (and indeed, exact coarsening techniques are not always possible).

This still does not tell us *how* to coarsen a given problem and the examples in Sect. 3 suggest that it is very much problem-specific. However, it is often possible that construction heuristics (traditionally used to construct an initial feasible solution prior to single-level refinement) can be modified into coarsening heuristics (e.g. the TSP & VRP rely on this technique). Furthermore, it has been shown (for partitioning at least), that it is usually more profitable for the coarsening to respect the objective function in some sense (e.g. [21, 48]) and most construction heuristics (apart from completely randomised instances) have this attribute.

Multilevel Enhancements

As we saw in Sect. 3, there are a number of generic ideas that can be used to improve the performance of a multilevel algorithm (these are discussed in more detail under multilevel enhancements in Sect. 3.1 and also in the summary of Sect. 3.2). In particular, techniques such as **constraint relaxation** and

coarsening homogeneity seem to be beneficial and are worth investigation for other problems.

Perhaps the most powerful, however, is solution-based **recoarsening**, particularly if used as part of an iterated multilevel algorithm. This is usually very easy to implement (relying merely on the coarsening being capable of respecting an existing solution) and can often considerably improve the solution quality of a multilevel scheme (e.g. see Sect. 2.3).

```
iterated multilevel refinement( input problem instance $P$ )
begin
    $C \leftarrow$ multilevel refinement( $P$ )   // initialise best solution
    $i \leftarrow 0$                              // unsuccessful iteration count
    while ( $i < \gamma$ )                        // for intensity parameter $\gamma$
        $C' \leftarrow$ multilevel refinement( $P$, $C$ )
        if ( $f(C') < f(C)$ )                     // $C'$ has lower cost
            $C \leftarrow C'$                     // update best solution
            $i \leftarrow 0$                      // reset unsuccessful count
        else
            $i \leftarrow i+1$
        endif
    end
end
```

Fig. 6. A schematic of the iterated multilevel refinement algorithm

Fig. 6 shows a schematic of a possible iterated multilevel algorithm (although there are other ways to specify the stopping criterion). Thus, after calculating an initial solution the algorithm repeatedly recoarsens and refines until no lower cost solution (where $f()$ denotes the objective/cost function) has been found after γ iterations. The only modification required to the multilevel algorithm of Fig. 5 is that it must take an existing solution as an additional input and coarsen that.

Summary

It seems likely that the most difficult aspect in implementing an effective multilevel scheme for a given problem and refinement algorithm is the (problem-specific) task of devising the coarsening strategy. However, examples indicate that this is often relatively straightforward.

4.3 Typical Runtime

One of the concerns that might be raised about multilevel algorithms is that instead of having just one problem to optimise, the scheme creates a whole hierarchy of approximately $O(\log_2 N)$ problems (assuming that the coarsening approximately halves the problem size at each level). In fact, it is not too difficult to show that there is approximately a factor-of-two difference in the runtime between a local search algorithm (or metaheuristic) at intensity λ, LS_λ, and the multilevel version, MLLS_λ. In other words, if $T(\text{A})$ denotes the runtime of algorithm A, then $T(\text{MLLS}_\lambda) \approx 2T(\text{LS}_\lambda)$.

Suppose first of all that the LS algorithm is $O(N)$ in execution time and that the multilevel coarsening manages to halve the problem size at every step. This is an upper bound and in practice the coarsening rate is actually slightly lower (e.g. between 5/8 to 6/8 is typical for GPP and TSP examples [44], rather than the theoretic maximum of 1/2). Let $T_L = T(\text{LS}_\lambda)$ be the time for LS_λ to run on a given instance of size N and T_C the time to coarsen and contract it. The assumption on the coarsening rate gives us a series of problems of size $N, N/2, N/4, \ldots, O(1)$ $(\approx N/N)$ whilst the assumption on LS_λ having linear runtime gives the total runtime for MLLS_λ as $T_C + T_L/N + \ldots + T_L/4 + T_L/2 + T_L$. If λ is small then T_C can take large proportion of the runtime and so multilevel algorithms using purely greedy refinement policies (i.e. typically $\lambda = 0$) tend to take more than twice the runtime of the equivalent local search although this depends on how long the coarsening takes compared with the solution construction (typically both $O(N)$). However, if λ is large enough then typically $T_C \ll T_L$ and so we can neglect it giving a total runtime of $T_L/N + \ldots + T_L/2 + T_L \approx 2T_L$, i.e. MLLS_λ takes twice as long as LS_λ to run.

Furthermore, if the local search scheme is also linear in λ, the intensity, it follows from $T(\text{MLLS}_\lambda) \approx 2T(\text{LS}_\lambda)$ that $T(\text{MLLS}_\lambda) \approx T(\text{LS}_{2\lambda})$. This effect is particularly visible for the TSP [42], but even for other problems where, for example λ expresses the number of failed iterations of some loop of the scheme, although this factor of two rule-of-thumb breaks down somewhat, it still gives a guide figure for the cost of a multilevel scheme.

Finally note that if the multilevel procedure is combined with an $O(N^2)$ or even $O(N^3)$ refinement algorithm then this analysis comes out even better for the multilevel overhead, i.e. $T(\text{MLLS}_\lambda) < 2T(\text{LS}_\lambda)$, as the final refinement step would requires an even larger proportion of the total.

Of course, this analysis assumes that the intensity of the search scheme is in some way dependent on the asymptotic convergence rate of the best solution. However, for local search schemes which are governed by a CPU time-limit, even tighter controls can be placed on the run-time of the multilevel version. Indeed, since the size of the problem at each level is known before any refinement takes place, the CPU time remaining after coarsening could be shared appropriately between each level and, for example, the multilevel version could be guaranteed to take the same time as the LS scheme.

5 Summary

We have seen evidence that the multilevel paradigm can aid metaheuristics and local search algorithms to find better or faster solutions for a growing number of combinatorial problems. We have discussed the generic features of these implementations and extracted some guidelines for its use. We have also identified some straightforward multilevel enhancements including recoarsening/iterated multilevel algorithms, homogeneous coarsening and constraint relaxation, all of which can boost its performance still further.

Overall this augments existing evidence that, although the multilevel framework cannot be considered as a panacea for combinatorial optimisation problems, it can provide a valuable (and sometimes a remarkably valuable) addition to the combinatorial optimisation toolkit. In addition, and unlike most metaheuristics, it is collaborative in nature and works alongside a separate refinement algorithm – successful examples include ant colony optimisation, cooperative search, genetic algorithms, simulated annealing and tabu search.

With regard to future work, clearly it is of great interest to further validate (or contradict) the conclusions by extending the range of problem classes. It is also valuable to identify and generalise those multilevel techniques which boost a particular application's performance still further (e.g. such as iterated multilevel schemes) so that they can become more widely employed.

References

1. A. Abou-Rjeili and G. Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proc. 20th Intl Parallel & Distributed Processing Symp., 2006*, page 10 pp. IEEE, 2006.
2. S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.
3. R. Battiti, A. Bertossi, and A. Cappelletti. Multilevel Reactive Tabu Search for Graph Partitioning. Preprint UTM 554, Dip. Mat., Univ. Trento, Italy, 1999.
4. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
5. C. Blum and M. Yábar. Multi-level ant colony optimization for DNA sequencing by hybridization. In F. Almeida *et al.*, editors, *Proc. 3rd Intl Workshop on Hybrid Metaheuristics*, volume 4030 of *LNCS*, pages 94–109. Springer-Verlag, Berlin, Germany, 2006.
6. C. Blum, M. Yábar-Vallès, and M. J. Blesa. An ant colony optimization algorithm for DNA sequencing by hybridization. *Computers & Operations Research*, 2007. (in press).
7. E. G. Boman and B. Hendrickson. A Multilevel Algorithm for Reducing the Envelope of Sparse Matrices. Tech. Rep. 96-14, SCCM, Stanford Univ., CA, 1996.
8. N. Bouhmala. Combining local search and genetic algorithms with the multilevel paradigm for the traveling salesman problem. In C. Blum *et al.*, editors, *Proc. 1st Intl Workshop on Hybrid Metaheuristics*, 2004. ISBN 3-00-015331-4.

9. A. Brandt. Multiscale Scientific Computation: Review 2000. In T. J. Barth, T. Chan, and R. Haimes, editors, *Multiscale and Multiresolution Methods*, pages 3–95. Springer-Verlag, Berlin, Germany, 2001.
10. T. N. Bui and C. Jones. A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. In R. F. Sincovec *et al.*, editors, *Parallel Processing for Scientific Computing*, pages 445–452. SIAM, Philadelphia, 1993.
11. J. Cong and J. Shinnerl, editors. *Multilevel Optimization in VLSICAD.* Kluwer, Boston, 2003.
12. T. G. Crainic, Y. Li, and M. Toulouse. A First Multilevel Cooperative Algorithm for Capacitated Multicommodity Network Design. *Computers & Operations Research*, 33(9):2602–2622, 2006.
13. C. Dai, P. C. Li, and M. Toulouse. A Multilevel Cooperative Tabu Search Algorithm for the Covering Design Problem. Dept Computer Science, Univ. Manitoba, 2006.
14. C. M. Fiduccia and R. M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *Proc. 19th IEEE Design Automation Conf.*, pages 175–181. IEEE, Piscataway, NJ, 1982.
15. J. E. Gallardo, C. Cotta, and A. J. Fernández. A Multi-level Memetic/Exact Hybrid Algorithm for the Still Life Problem. In T. P. Runarsson *et al.*, editors, *Parallel Problem Solving from Nature – PPSN IX*, volume 4193 of *LNCS*, pages 212–221. Springer-Verlag, Berlin, 2006.
16. J. Gu and X. Huang. Efficient Local Search With Search Space Smoothing: A Case Study of the Traveling Salesman Problem (TSP). *IEEE Transactions on Systems, Man & Cybernetics*, 24(5):728–735, 1994.
17. A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix reordering. *IBM Journal of Research & Development*, 41(1/2):171–183, 1996.
18. D. Harel and Y. Koren. A Fast Multi-Scale Algorithm for Drawing Large Graphs. *Journal of Graph Algorithms & Applications*, 6(3):179–202, 2002.
19. B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In S. Karin, editor, *Proc. Supercomputing '95, San Diego (CD-ROM).* ACM Press, New York, 1995.
20. Y. F. Hu and J. A. Scott. Multilevel Algorithms for Wavefront Reduction. RAL-TR-2000-031, Comput. Sci. & Engrg. Dept., Rutherford Appleton Lab., Didcot, UK, 2000.
21. G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
22. G. Karypis and V. Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In D. Duke, editor, *Proc. Supercomputing '98, Orlando.* ACM SIGARCH & IEEE Comput. Soc., 1998. (CD-ROM).
23. G. Karypis and V. Kumar. Multilevel *k*-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel & Distributed Computing*, 48(1):96–129, 1998.
24. G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. *VLSI Design*, 11(3):285–300, 2000.
25. A. Kaveh and H. A. Rahimi-Bondarabady. A Hybrid Graph-Genetic Method for Domain Decomposition. In B. H. V. Topping, editor, *Computational Engineering using Metaphors from Nature*, pages 127–134. Civil-Comp Press, Edinburgh, 2000. (Proc. Engrg. Comput. Technology, Leuven, Belgium, 2000).
26. B. W. Kernighan and S. Lin. An Efficient Heuristic for Partitioning Graphs. *Bell Systems Technical Journal*, 49:291–308, 1970.

27. P. Korošec, J. Šilc, and B. Robič. Solving the mesh-partitioning problem with an ant-colony algorithm. *Parallel Computing*, 30:785–801, 2004.
28. A. E. Langham and P. W. Grant. A Multilevel k-way Partitioning Algorithm for Finite Element Meshes using Competing Ant Colonies. In W. Banzhaf *et al.*, editors, *Proc. Genetic & Evolutionary Comput. Conf. (GECCO-1999)*, pages 1602–1608. Morgan Kaufmann, San Francisco, 1999.
29. I. O. Oduntan. A Multilevel Search Algorithm for Feature Selection in Biomedical Data. Master's thesis, Dept. Computer Science, Univ. Manitoba, 2005.
30. D. Rodney, A. Soper, and C. Walshaw. The Application of Multilevel Refinement to the Vehicle Routing Problem. In D. Fogel *et al.*, editors, *Proc. CISChed 2007, IEEE Symposium on Computational Intelligence in Scheduling*, pages 212–219. IEEE, Piscataway, NJ, 2007.
31. D. F. Rogers, R. D. Plante, R. T. Wong, and J. R. Evans. Aggregation and Disaggregation Techniques and Methodology in Optimization. *Operations Research*, 39(4):553–582, 1991.
32. Y. Romem, L. Rudolph, and J. Stein. Adapting Multilevel Simulated Annealing for Mapping Dynamic Irregular Problems. In S. Ranka, editor, *Proc. Intl Parallel Processing Symp.*, pages 65–72, 1995.
33. Camilo Rostoker and Chris Dabrowski. Multilevel Stochastic Local Search for SAT. Dept Computer Science, Univ. British Columbia, 2005.
34. Y. Saab. Combinatorial Optimization by Dynamic Contraction. *Journal of Heuristics*, 3(3):207–224, 1997.
35. I. Safro, D. Ron, and A. Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.
36. K. Schloegel, G. Karypis, and V. Kumar. Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. *Journal of Parallel & Distributed Computing*, 47(2):109–124, 1997.
37. K. Schloegel, G. Karypis, and V. Kumar. A New Algorithm for Multi-objective Graph Partitioning. In P. Amestoy *et al.*, editors, *Proc. Euro-Par '99 Parallel Processing*, volume 1685 of *LNCS*, pages 322–331. Springer-Verlag, Heidelberg, Germany, 1999.
38. A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. *Journal of Global Optimization*, 29(2):225–241, 2004.
39. S.-H. Teng. Coarsening, Sampling, and Smoothing: Elements of the Multilevel Method. In M. T. Heath *et al.*, editors, *Algorithms for Parallel Processing*, volume 105 of *IMA Volumes in Mathematics and its Applications*, pages 247–276. Springer-Verlag, New York, 1999.
40. M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level Cooperative Search: A New Paradigm for Combinatorial Optimization and an Application to Graph Partitioning. In P. Amestoy *et al.*, editors, *Proc. Euro-Par '99 Parallel Processing*, volume 1685 of *LNCS*, pages 533–542. Springer-Verlag, Berlin, 1999.
41. D. Vanderstraeten, C. Farhat, P. S. Chen, R. Keunings, and O. Zone. A Retrofit Based Methodology for the Fast Generation and Optimization of Large-Scale Mesh Partitions: Beyond the Minimum Interface Size Criterion. *Computer Methods in Applied Mechanics & Engineering*, 133:25–45, 1996.
42. C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. *Operations Research*, 50(5):862–877, 2002.
43. C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. *Journal of Graph Algorithms & Applications*, 7(3):253–285, 2003.

44. C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Operations Research*, 131:325–372, 2004.
45. C. Walshaw. Variable partition inertia: graph repartitioning and load-balancing for adaptive meshes. In S. Chandra M. Parashar and X. Li, editors, *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications*. Wiley, New York. (Invited chapter – to appear in 2008).
46. C. Walshaw and M. Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.
47. C. Walshaw and M. Cross. Multilevel Mesh Partitioning for Heterogeneous Communication Networks. *Future Generation Computer Systems*, 17(5):601–623, 2001.
48. C. Walshaw, M. Cross, R. Diekmann, and F. Schlimbach. Multilevel Mesh Partitioning for Optimising Domain Shape. *International Journal of High Performance Computing Applications*, 13(4):334–353, 1999.
49. C. Walshaw and M. G. Everett. Multilevel Landscapes in Combinatorial Optimisation. Tech. Rep. 02/IM/93, Comp. Math. Sci., Univ. Greenwich, London SE10 9LS, UK, April 2002.