FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Hybrid Algorithm for the Partition Coloring Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Gilbert Fritz
Matrikelnummer 0827276

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Dr. Bin Hu

Wien, 21.Oct.2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Gilbert Fritz
Schlosshofer Straße 49/18

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Danksagung

Ich danke meinen Betreuern, ao. Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl und Univ.Ass. Dipl.-Ing. Dr.techn. Bin Hu für ihre Unterstützung bei der Erstellung dieser Arbeit durch ihr konstruktives Feedback und ihre Ideen, welche es mir ermöglicht haben, immer neue Aspekte der Problemstellung zu erkennen.

Besonderer Dank gilt meinen Eltern, Franz und Justine Fritz, sowie meiner Partnerin Odnoo und meinen Freunden für Ihre Unterstützung.

Abstract

Todo

Kurzfassung

Todo

Contents

CHAPTER 1

Preliminaries

This chapter introduces theoretical fundamentals like definitions, terms and methods, that are necessary for analysing the Partition Coloring Problem. The presented notations will be used consistently in this thesis.

1.1 Optimization Problems and Complexity

Since this thesis deals with an optimization problem and the analysis of a solution to it needs to consider some complexity theory is an important field of computer science Some definition and explainations of optimization problems and complexity are given in this section. For a more detailed insight the reader is refered to [4, 9, 13].
In general an optimization problem is the problem of finding the best solution among all feasible solutions. Depending on weather the variables are continuous or discrete, the optimization problem is said to be a continuous optimization problem or a combinatorial optimization problem (COP). Since the PCP belongs to the latter category, this thesis will not cover further explainations on continuous optimization problems. For information on that topic, the reader is refered to [8, 12]. Most of the following the definitions have been introduced in [1, 3].

Definition 1 (Combinatorial Optimization Problem) *A Combinatorial Optimization Problem P is defined as $P = (S, f)$*

- *A set of variables $X = \{x_1, x_2, \dots x_n\}$*
- *variable domains $D_1, \dots, D_n$*
- *constraints among the variables*
- *an objective function $f : D_1 \times D_2 \times \dots D_n \to R$ that evaluates each element in S*

The set of all feasible assignments is $S = \{s = \{(x_1, v_1), (x_2, v_2), \dots (x_n, v_n)\}\}) \mid v_i \in D_i$ *satisfies all the constraints* $\}$

For each COP P there exists a corresponding decision problem D, i.e. a problem whose output is either *YES* or *NO*. The complexity of D determines the complexity of P.

Definition 2 (Decision Problem) *The decision problem D for a Combinatorial Optimization Problem P asks if, for a given solution $s \in S$, there exists a solution $s' \in S$, such that $f(s')$ is better than $f(s)$: for a minimization problem this means $f(s') < f(s)$ and for a maximization problem $f(s) > f(s')$.*

An important issue that comes up when considering combinatorial optimization problems is the classification problems by their difficulty . To categorize problems into easy and difficult ones, the class of problems that are solvable in polynomial time by a deterministic touring machine and problems that are solvable in polynomial time by a nondeterministic touring machine are considered. This thesis prefers to describe the characteristics of $\mathcal{P}$ and $\mathcal{NP}$ at a more intuitive level, rather than formalizing the classes via Turing Machines. An examples for a problem in $\mathcal{P}$ is single source shortest path.

Definition 3 (Complexity class $\mathcal{P}$) *A problem is in $\mathcal{P}$ iff it can be solved by an algorithm in polynomial time.*

The complexity class $\mathcal{NP}$ is associated with hard problems. $\mathcal{NP}$ stands for "nondeterministic polynomial time", where "nondeterministic" is a way to express that solutions are guessed. The class $\mathcal{NP}$ is restricted to Decision Problems.

Definition 4 (Complexity class $\mathcal{NP}$) *A decision problem is in $\mathcal{NP}$ iff any given solution of the problem can be verified in polynomial time.*

The above definition states, that the solutions for problems in $\mathcal{NP}$ do not require to be calculated in polynomial time, but the solutions need to be verified in polynomial time. Therefore $\mathcal{P} \subseteq \mathcal{NP}$ holds (slightly abusing notation by restricting $\mathcal{P}$ to decision problems) [3].

Definition 5 ($\mathcal{NP}$-optimization Problem) *A COP is a $\mathcal{NP}$-optimization problem (NPO) if the corresponding decision problem is in $\mathcal{NP}$.*

As an example the decision variant of the Stardard Vertex Coloring Problem (VCP) is considered, which asks weather a graph G is colorable within k colors or not. An algorithm has to verify for each vertex v colored with color c_v, if all its neighbours are colored with a color different to c_v and further count the number of distinct colors to check weather the number of colors is lower or equal to k. This can be done in time $\mathcal{O}(|V^2|)$, where V is the set of vertices.

The VCP and many other optimization and decision problems are at least as difficult as any problem in $\mathcal{NP}$. These problems are said to be $\mathcal{NP}$-hard. Giving a polynomial-time reduction from an $\mathcal{NP}$-hard problem to a particular problem shows that this problem is $\mathcal{NP}$-hard, too.

Such a reduction links the considered problem to the known $\mathcal{NP}$-hard problem in such a way that iff the considered problem can be solved in polynomial time also the $\mathcal{NP}$-hard problem to which it has been reduced can. [3] To gain a more detailed insight into that topic, the reader is referenced to [15].

Definition 6 ($\mathcal{NP}$-hard problems) *A problem is called $\mathcal{NP}$-hard iff it is at least as difficult as any problem in $\mathcal{NP}$, i.e., each problem in $\mathcal{NP}$ can be reduced to it.*

A lot of optimization problems are $\mathcal{NP}$-hard but not in $\mathcal{NP}$. For example, the optimization variant of the VCP, which searches for the minimum chromatic number is clearly at least as hard at its decision variant described above. Since the output is a number rather than a decision, it is not in $\mathcal{NP}$. $\mathcal{NP}$-hard problems that are also in $\mathcal{NP}$ are called $\mathcal{NP}$-complete. Many decision variants of $\mathcal{NP}$-hard problems like the VCP are $\mathcal{NP}$-complete.

Definition 7 ($\mathcal{NP}$-complete problems) *A problem is $\mathcal{NP}$-complete iff it is $\mathcal{NP}$-hard and in $\mathcal{NP}$.*

Informally, $\mathcal{NP}$-complete problems are the hardest problems in the class $\mathcal{NP}$. If there is an algorithm that solves any $\mathcal{NP}$-complete problem in polynomial time, then every problem in $\mathcal{NP}$ can be solved in polynomial time. So far no polynomial time deterministic algorithm has been found to solve one of them.

Theorem 1 *If any $\mathcal{NP}$-complete problem can be solved by a polynomial-time deterministic algorithm, then $\mathcal{NP} = \mathcal{NP}$. If any problem in $\mathcal{NP}$ cannot be solved by a polynomial-time deterministic algorithm, then $\mathcal{NP}$-complete problems are not in $\mathcal{P}$.*

Most computer scientists assume that $\mathcal{P} \neq \mathcal{NP}$, although it has not been proven yet. The question $\mathcal{P} = \mathcal{NP}$ is one of the most promiment unresolved questions in the field of complexity theory, since a proof would imply a huge impact on any other discipline in discrete mathematics and computer science.
Nevertheless, for some $\mathcal{NP}$-complete problems of this class it is possible develop algorithms that have an average-case polynomial time complexity, despite having exponential time complexity in worst case. For other problems in this class, approximation algorithms can be found that return solutions in polynomial time with a guarantee of a specific solution quality. The development and analysis of approximation algorithms is an important field of research. The following definitions are taken from [16].

Definition 8 (Approximation algorithm) *An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.*

The α is called approximation ratio or performance guarantee of the α-approximation algorithm. For minimization problems $\alpha > 1$ and for maximization problems $\alpha < 1$ holds. For example, a $1/2$-approximation algorithm for a maximization problem always returns a solution in polynomial time, that is at least half as good as the optimal solution. For some problems there even exist polynomial time algorithms, whose approximation ratio can be given as parameter. They have so called polynomial-time approximation schemes.

Definition 9 (Polynomial-time approximation scheme) *A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\epsilon\}$, where there is an algorithm for each $\epsilon > 0$, such that A_ϵ is a $(1+\epsilon)$-approximation algorithm (for minimization problems) or a $(1-\epsilon)$-approximation algorithm (for maximization problems).*

1.2 Graph Theory Definitions

Definition 10 (Graph) *A graph is a tuple $G = (V, E)$, where V denotes the set of nodes and $E \subseteq V \times V$ denotes the set of edges. An edge from node i to j is denoted by $\{i, j\}$. We call a graph simple, if it does not contain multiple edges, i.e. more than one edge between the same nodes, or loops, i.e. edges $\{i, i\}$.*

Definition 11 (Directed graph) *A directed graph or digraph is a tuple $D = (V, A)$, where V denotes the set of nodes and $A \subseteq V \times V$ denotes the set of arcs or directed edges. An arc from node i to j is denoted by (i, j). We call a directed graph simple, if it does not contain multiple arcs, i.e. more than one arc between the same nodes, or loops, i.e. arcs (i, i).*

Unless declared explicitly, this thesis considers only simple, undirected graphs $G = (V, E)$.

Definition 12 (Directed graph) *Given a graph $G = (V, E)$, $G' = (V', E')$ is called a subgraph if $V' \subseteq V$, $E' \subseteq E$ and $E' \subseteq V' \times V'$. If $V' = V$ we call G a spanning subgraph or factor.*

Definition 13 (Deletion of a node) *Given a graph $G = (V, E)$, $G - v = (V \backslash v, E \backslash \{e \mid v \in e\})$.*

Definition 14 (Adjacency and incidence) *Two nodes x and y are called adjacent, if they share an edge e, i.e. $\exists e = \{x, y\} \in E$. Two edges e and f are called adjacent, if they share a node x, i.e. $e \cap f = x$. A node v is called incident to an edge e, if $v \in e$.*

Definition 15 (Node degree) *The degree of a node v in an undirected graph G, denoted by $d(v)$, is the number of edges, that are incident to the node v, i.e. in E there exists an edge $\{v, x\}$. The number of outgoing arcs (v, x) from a node v in a directed graph D is called out-degree and is denoted by $d^+(v)$, the number of ingoing arcs (x, v) to a node v is called in-degree and is denoted by $d^-(v)$.*

Lemma 1 (Handshaking Lemma)

$$\sum_{v \in V} d(v) = 2\,|E|$$

Proof. As every edge $\{i, j\}$ is incident to exactly two nodes, namely i and j, it is counted one time at $d(i)$ and one time at $d(j)$. So the sum over all node degrees is exactly two times the number of edges. □

Corollary 1 (Directed graph) *The number of nodes with odd node degree is even.*

Proof. This immediately follows from the handshaking lemma. □

Lemma 2

$$\sum_{v \in V} d^+(v) = \sum v \in V d^-(v) = 2\,|A|$$

Proof. As every arc (i, j) has exactly one "in-node" and one "out-node", it follows, that the sum of all out-degrees equals the sum of all in-degrees and hence the number of arcs. □

Definition 16 (Maximum and minimum node degree) $\triangle(G) = max\{d(v) \mid v \in V\}$ *denotes the maximum node degree in a graph.* $\delta(G) = min\{d(v) \mid v \in V\}$ *denotes the minimum node degree in a graph.*

Definition 17 (Neighborhood of a node) *The neighborhood of a node* $v \in V$ *is denoted by* $N(v) = \{x \mid \{v, x\} \in E\}$*. In the directed case the neighborhood consists of all nodes that are reachable from v, i.e.* $N(v) = \{x \mid (v, x) \in A\}$*.*

Definition 18 (Walk) *A sequence* $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ *with* $n \geq 0$ *is called a walk, if for all* v_i *with* $i \neq 0$ *exists an* $e_i = \{v_{i-1}, v_i\} \in E$*.*

Definition 19 (Directed walk) *A sequence* $v_0, a_1, v_1, a_2, \ldots, a_n, v_n$ *with* $n \geq 0$ *is called a directed walk, if for all* v_i *with* $i \neq 0$ *exists an* $a_i = (v_{i-1}, v_i) \in A$*.*

Definition 20 (Trail) *A sequence* $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ *with* $n \geq 0$ *is called a trail, if for all* v_i *with* $i \neq 0$ *exists an* $e_i = \{v_{i-1}, v_i\} \in E$ *and all* e_i *are distinct.*

Definition 21 (Directed Trail) *A sequence* $v_0, a_1, v_1, a_2, \ldots, a_n, v_n$ *with* $n \geq 0$ *is called a directed trail, if for all* v_i *with* $i \neq 0$ *exists an* $a_i = (v_{i-1}, v_i) \in A$ *and all* a_i *are distinct.*

Definition 22 (Path) *A sequence* $v_0, e_1, v_1, e_2, \ldots, e_n, v_n$ *with* $n \geq 0$ *is called a path, if for all* v_i *with* $i \neq 0$ *exists an* $e_i = \{v_{i-1}, v_i\} \in E$ *and all* v_i *are distinct.*

Definition 23 (Directed path) *A sequence* $v_0, a_1, v_1, a_2, \ldots, a_n, v_n$ *with* $n \geq 0$ *is called a directed path, if for all* v_i *with* $i \neq 0$ *exists an* $a_i = (v_{i-1}, v_i) \in A$ *and all* v_i *are distinct.*

Definition 24 (Length of a path) *Given a path* $P = v_0, e_1, v_1, e_2, \ldots, e_n, v_n = P(v_0, v_n)$*, the length is the number of edges and denoted by* $l(P) = n$*, analogously for the directed case.*

Definition 25 (Cycle) *A cycle is a path, where $v_0 = v_n$.*

Definition 26 (Acyclic graph) *A graph is called acyclic if it does not contain a cycle.*

Theorem 2 *Let $W = W(v_0, v_n)$ be a walk, then there is a subsequence $P = P(v_0, v_n) \subseteq W(v_0, v_n)$ such that P is a path.*

Proof. We know that for a path holds $v_i = v_j \forall i < j$. Suppose for W holds that $v_i = v_j$ for arbitrary $i < j$. Then $W = v_0, e_1, v1, \ldots, v_i, e_{j+1}, v_{j+1}, \ldots, v_n$ is a walk with $i - j$ less edges and W is a subsequence of W. Applying this until $\forall i, j : v_i = v_j$ yields a path from v_0 to v_n. This of course also holds for the directed case. $\square$

Definition 27 (Network) *A network $N = (G, c)$ consists of a graph $G = (V, E)$ and a cost function $c : E(G) \longrightarrow \mathbb{R} \geq 0$, which assigns each edge e a nonnegative value c_e . Networks are also called weighted graphs.*

Definition 28 (Costs of a graph) *The cost c_G of a graph G is the sum of its edge costs, i.e. $c_G = \sum_{e \in E} c_e$.*

Definition 29 (Coloring) *A coloring is a mapping $v \to c_v \mid c_v \in \mathbb{R}, \forall v \in V$. The coloring is feasible, iff $\{x, y\} \in E \mid c_x \neq c_y, \forall x \in V, \forall y \in V$*

Definition 30 (chromatic number) *Let $G = (V, E)$ be a graph. We state that c is a (proper) k-coloring of G if all the vertices in V are colored using k colors such that no two adjacent vertices have the same color. The chromatic number is defined as the minimum k for which there exists a (proper) k-coloring of G.*

1.3 Metaheuristics

As defined before, a COP consists of a set of feasible solutions. In almost every case this is of huge size compared to the size of the instance. Solving a COP exactly means finding the optimal solution out of that set. Since for $\mathcal{NP}$-complete problems no algorithm that performs in polynomial time could be found yet, scientists try to find algorithms that apporixmate optimal solutions. In general there exist two classes of approximation methods: construction and improvement heuristics. As its name states, the former construct a solution from scratch by adding components until the solution is complete. Usually these algorithms perform fastest but often return a solution quality that is inferior to the ones returned by improvement heuristics. Used as initial solution for an improvement heuristic, the construction heuristic may return an infeasible solution. Afterwards an improvement heuristic iteratrively tries to replace the solution by a better/feasible one that is derived from the current solution.

Improvement heuristics can be be divided into two groups, population based and local search based heuristics. [1] The former includes – but is not restricted to – Ant Colony Optimization

(ACO), Evolutionary Computation (EC) including Genetic Algorithms (GA) and the second group consists of Iterated Local Search (ILS), Simulated Annealing (SA) and Tabu Search (TS). TS is described in more detail in section 1.5 and as this thesis does not deal with population based algorithms, it excludes further descriptions of this group. The reader is refered to [5].
All these methods form a relatively new group of heuristics and are sumed up by the term *metaheuristics*, which was first introduced in [7]. In 1996 Osman and Laporte provided a formal definition of metaheuristics [11]:

Definition 31 (Metaheuristic) *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.*

The famous no free lunch theorem [17] states, that over all possible problems, there is no heuristic that performs better than any other heuristic including random search. According to the theorem, if a strategy performs better in one subarea, it performs worse in another. By acquiring and including problemspecific knowledge, it is possible to develop strategies for classes of problems that perform better than others [2]. In this context, Gebhard defined in [14]:

Definition 32 *Metaheuristic algorithms make no assumptions on the problem and (in theory) can be applied on any optimization problem. They define an abstract order of instructions which lead to improved or feasible solutions. In almost any case these instructions must be implemented using problem specific knowledge.*

1.4 Basic Local Search

Basic local search (LS) is an improvement heuristic that iteratively tries to replace the solution by a better one that is located in an appropriately defined neighborhood structure of the current solution. [1]

Definition 33 (Neighborhood structure) *A neighborhood structure is a function $\mathcal{N} : S \to 2^S$ that assigns to every $s \in S$ a set of neighbors $\mathcal{N}(s) \subseteq S$. $\mathcal{N}(s)$ is called the neighborhood of s.*

Iterately improving the solution by choosing the first neighbor from $\mathcal{N}$(s) that is better is called *First Fit* (FF), choosing the best neighbor is called *Best Fit* (BF). In the basic version of LS, both variants stop if no better solution can be found, which is called a local minimum. As this thesis is about the PCP which is a minimization problem, optimum and minimum are used equivalently.

Definition 34 (Local minimum) *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 $\forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) < f(s)$.*

Stopping at a local minimum results in quite unsatisfactory solutions, therefore methods have been developed to escape from such a local minimum in order to find the global minimum. A simple technique is to start the LS from different points repeatedly, which is not very efficient, since the search information from preciding searches is not used. Metaheuristic algorithms use more complex termination conditions including maximum number of iterations or CPU time.

Definition 35 (Global minimum) *A global minimum (optimum) of a minimizing combinatorial optimization problem is a solution, such that $f(\hat{s}) \leq f(s)$, $\forall x \in X$. Therefore a global optimum is a local optimum for all neighborhood structures $\mathcal{N}$.*

1.5 Tabu Search

1.6 Linear Programming

Linear Programming (LP) or mathematical programming is a large field of operations re- search and tries to optimize an objective function under a set of constraints. The objective function and all constraints are linear inequations and equations. Most optimization prob- lems which arise from nowadays industries needs can be written as a linear program. A short introduction to the field of linear programming and the corresponding solving methods is given in Section 2.5. The following sections provide a detailed insight into the implemented metaheuristic algorithms.

Bibliography

[1] C. Blum and A. Roli. *Metaheuristics in combinatorial optimization: Overview and conceptual comparision*. ACM Computing Surveys, 2003.

[2] Raymond Chiong. Nature-inspired algorithms for optimisation. *Studies in Computational Intelligence*, 193, 2009.

[3] Bioinspired Computation F. Neumann, C. Witt. Bioinspired computation in combinatorial optimization. *Combinatorial Optimization, Natural Computing Series*, 2010.

[4] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. *Series of Books in the Mathematical Sciences*, 1, 1979.

[5] M. Gendreau. *Handbook of metaheuristics*. Springer, 2010.

[6] S.Pirkwieser G.Fritz, G.Raidl. Heuristic methods for the hop constrained survivable network design problem. 2011.

[7] F. Glover. Future paths for integer programming and links to artificial intelligence. *Cambridge University Press*, 13(533–549), 1986.

[8] S. Wright J. Nocedal. Numerical optimization. 2000.

[9] E. Lawler. Combinatorial optimization: Networks and matroids. 2002.

[10] B.Hu M.Leitner, G.Raidl. Solving two generalized network design problems with exact and heuristic methods. 2006.

[11] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(513-623), 1996.

[12] H.Romeijn P. Pardalos. Handbook of global optimization. 2, 2002.

[13] C. M. Papadimitriou. Computational complexity. 1, 1994.

[14] S.Pirkwieser P.Gebhard, G.Raidl. The vehicle routing problem with compartments based on solutions of lp-relaxations. 2012.

[15] Ingo Wegener. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer, 2005. Reflects recent developments in its emphasis on randomized and approximation algorithms and communication models.

[16] David P. Williamson and David B. Shmoys. The design of approximation algorithms. *Cambridge University Press*, 2010.

[17] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(67-82), 1997.