Helsinki University of Technology Department of Communications and Networking
Teknillinen korkeakoulu Tietoliikenne- ja tietoverkkotekniikan laitos
Espoo 2010 Report 7/2010

RUSSIAN DOLL SEARCH ALGORITHMS FOR DISCRETE OPTIMIZATION PROBLEMS

Vesa Vaskelainen

Aalto-yliopisto
Teknillinen korkeakoulu
Aalto-universitetet
Tekniska högskolan
Aalto University
School of Science and Technology

Helsinki University of Technology Department of Communications and Networking
Teknillinen korkeakoulu Tietoliikenne- ja tietoverkkotekniikan laitos
Espoo 2010 Report 7/2010

RUSSIAN DOLL SEARCH ALGORITHMS FOR DISCRETE OPTIMIZATION PROBLEMS

Vesa Vaskelainen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Electronics, Telecommunications and Automation for public examination and debate in Auditorium S4 at Aalto University School of Science and Technology (Espoo, Finland) on the 19th of November, 2010, at 12 noon.

Aalto University
School of Science and Technology
Faculty of Electronics, Telecommunications and Automation
Department of Communications and Networking

Aalto-yliopisto
Teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikenne- ja tietoverkkotekniikan laitos

Distributor:
Aalto University
School of Science and Technology
Department of Communications and Networking
P.O. Box 13000
FI-00076 Aalto
Tel. +358-9-470 25300
Fax +358-9-470 22474

ISBN 978-952-60-3409-6
ISBN 978-952-60-3410-2 (pdf)
ISSN 1797-478X
ISSN 1797-4798 (pdf)

Multiprint Oy
Espoo 2010

Aalto University

ABSTRACT OF DOCTORAL DISSERTATION	AALTO UNIVERSITY SCHOOL OF SCIENCE AND TECHNOLOGY P.O. BOX 11000, FI-00076 AALTO http://www.aalto.fi

Author Vesa Vaskelainen

Name of the dissertation

Russian Doll Search Algorithms for Discrete Optimization Problems

Manuscript submitted 1.2.2010 | Manuscript revised 5.10.2010

Date of the defence 19.11.2010

☐ Monograph | ☑ Article dissertation (summary + original articles)

Faculty	Faculty of Electronics, Communications and Automation
Department	Department of Communications and Networking
Field of research	Information theory
Opponent(s)	Dr. Axel Kohnert
Supervisor	Prof. Patric Östergård
Instructor	Prof. Patric Östergård

Abstract

This dissertation discusses exhaustive search algorithms for discrete optimization problems. The search space of the problems is pruned by determining different lower or upper bounds for the solution of the problem. In some cases, redundant candidates can be pruned on the basis of structural symmetry. One effective approach to prune the search space is Russian doll search, which is based on the idea of dividing a problem into smaller subproblems that are subsequently solved in an ascending order. The solutions to the previous subproblems are then used to further prune, in order to solve the next subproblem. The final subproblem is, then, the original problem.

In this work, Russian doll search is applied to some optimization problems in digraphs and hypergraphs. The problems considered are the 'Steiner triple covering problem', the 'maximum transitive subtournament problem', and the 'best barbeque problem'. The Steiner triple covering problem is a hitting set problem that corresponds to a search for a vertex cover from a hypergraph. A search for a transitive subtournament from a digraph can be translated to a search for an independent set from a hypergraph. Moreover, the best barbeque problem can be presented as a problem on hypergraphs.

The performance of the implemented algorithms was experimentally evaluated. For example, the Russian doll search algorithm for the Steiner triple covering problem A_{135} has solved an instance approximately 100 times faster than the leading commercial software for integer programming. In the best barbeque problem context, the relevant parameters of test instances comprised the complete range of relevant parameters for real biological instances. Algorithms designed to find the maximum (order of) transitive subtournaments have succeeded in all but eight directed graphs, of up to five vertices, determining the lower bounds for Sperner capacities that meet the upper bounds, obtained by other methods. In addition, a new record, a tournament of order 14, in which tournament winners are disjoint by using the definitions developed by Banks and Slater, is discovered with the algorithms for finding the maximum (order of) transitive subtournaments.

Keywords	backtrack search, digraphs, Russian doll search, transitive tournaments		
ISBN (printed)	978-952-60-3409-6	ISSN (printed)	1797-478X
ISBN (pdf)	978-952-60-3410-2	ISSN (pdf)	978-952-60-3410-2
Language	English	Number of pages	vii+36
Publisher and print distribution	Aalto University School of Science and Technology Department of Communications and Networking		

☑ The dissertation can be read at http://lib.tkk.fi/Diss/2010/isbn9789526034102/

Aalto-yliopisto

VÄITÖSKIRJAN TIIVISTELMÄ	AALTO-YLIOPISTO TEKNILLINEN KORKEAKOULU PL 11000, 00076 AALTO http://www.aalto.fi

Tekijä Vesa Vaskelainen

Väitöskirjan nimi

Maatuskanukkealgoritmit diskreeteille optimointiongelmille

Käsikirjoituksen päivämäärä	1.2.2010	Korjatun käsikirjoituksen päivämäärä	5.10.2010
Väitöstilaisuuden ajankohta	19.11.2010		
☐ Monografia		☑ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)	

Tiedekunta	Elektroniikan, tietoliikenteen ja automaation tiedekunta
Laitos	Tietoliikenne- ja tietoverkkotekniikan laitos
Tutkimusala	Informaatioteoria
Vastaväittäjä(t)	Dr. Axel Kohnert
Työn valvoja	Prof. Patric Östergård
Työn ohjaaja	Prof. Patric Östergård

Tiivistelmä

Väitöskirja käsittelee diskreettien optimointiongelmien täydellisiä hakualgoritmejä. Ongelmien hakuavaruutta pyritään karsimaan erilaisin ongelman ratkaisulle määritettävin ala- tai ylärajoin. Joissain tapauksissa voidaan matemaattisen rakenteen symmetrioiden perusteella karsia redundantit ratkaisuehdokkaat. Yksi tehokkaaksi osoittaunut karsintamenetelmä on niin kutsuttu maatuskanukkealgoritmi, joka perustuu ideaan jakaa ongelma pienempiin osaongelmiin joita ratkotaan kasvavassa järjestyksessä. Tietoa edellisten osaongelmien ratkaisuista käytetään aina hyödyksi seuraavan ratkaisemisessa. Lopulta ratkaistaan alkuperäinen ongelma hyödyntäen tietoa kaikista aiemmista osa-ongelmista.

Tässä väitöskirjatyössä maatuskanukkealgoritmiä hyödynnetään eräiden suunnattujen graafien ja hypergraafien optimointiongelmien ratkaisemisessa. Käsiteltävät ongelmat ovat Steinerin kolmikkosysteemin peitto-ongelma, transitiivisen osaturnauksen etsiminen ja best barbeque -ongelma. Steinerin kolmikkosysteemin peitto-ongelma on osumisjoukko-ongelma, joka vastaa solmupeiton etsimistä hypergraafista. Transitiivisen osaturnauksen etsiminen suunnatusta graafista voidaan muuntaa riippumattoman joukon etsinnäksi hypergraafista. Best barbeque -ongelma on myös esitettävissä hypergraafiongelmana.

Implementoitujen algoritmien suorituskyky arvioitiin kokeellisesti. Esimerkiksi, Steinerin kolmikkosysteemin peitto-ongelman A_{135} ratkaisemiseen tehty maatuskanukkealgoritmi suoriutuu tehtävästä noin sadasosa ajassa siitä mitä saman ongelman ratkaisemiseen kuluu johtavaa kaupallista kokonaislukuohjelmointiohjelmistoa käyttäen. Best barbeque -ongelman ratkaisemiseen tehdyn maatuskanukkealgoritmin nopeus on riittävä aitojen biologisten instanssien ratkaisemiseen. Transitiivisten osaturnausten etsintään tehdyillä algoritmeillä onnistutaan määrittämään Spernerin kapasiteetin alarajat, jotka ovat yhtä suuria muulla tavoin määritettyjen ylärajojen kanssa, kaikille suunnatuille viisisolmuisille graafeille kahdeksaa tapausta lukuunottamatta. Lisäksi transitiivisten osaturnausten etsintään tehdyillä algoritmeillä löydetään uusi ennätys, kokoa 14 oleva turnaus, jolle turnausvoittajat Slaterin ja Banksin määritelmiä käyttäen ovat erilliset.

Asiasanat	maatuskanukkealgoritmi, peräytyvä haku, suunnatut graafit, transitiiviset turnaukset		
ISBN (painettu)	978-952-60-3409-6	ISSN (painettu)	1797-478X
ISBN (pdf)	978-952-60-3410-2	ISSN (pdf)	978-952-60-3410-2
Kieli	englanti	Sivumäärä	vii+36
Julkaisija ja painetun väitöskirjan jakelu	Aalto-yliopiston teknillinen korkeakoulu Tietoliikenne- ja tietoverkkotekniikan laitos		

☑ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/2010/isbn9789526034102/

Contents

Preface

This dissertation is a product of research carried out in the Department of Communications and Networking, in the Faculty of Electronics, Communications and Automation, Helsinki University of Technology TKK; under the project "Combinatorial Algorithms", grant number 107493, in the Academy of Finland research programme "Basic Research in Programming, Algorithms and their Support Functions".

I wish to express sincere gratitude to Professor Patric Östergård, who unfalteringly instructed and supervised this work. He has been an oracle-like resource, always available when I was stuck and needed help. Many thanks also go to the co-authors, Lasse Kiviluoto, Axel Mosig, and Vesa Riihimäki, whose contribution was essential to my dissertation. I would also like to thank Professor Harri Haanpää for his valuable comments. To my colleagues in the Information theory research group, Pekka Lampio, Markku Liinaharja, Vesa Linja-aho, Olli Pottonen, and Esa "Esa A" Seuranen, I would like to say thank you for inspiring an intellectual atmosphere that constantly fed my creativity.

I offer my humblest thanks for the economic support that the Academy of Finland, Tekniikan edistämissäätiö, Walter Ahlströmin säätiö, and Alfred Kordelinin säätiö provided me.

Finally, I am grateful to my parents, parents-in-law, relatives, and relatives by marriage for the help offered in my everyday life. Above all, thanks to my loving wife Elina and to my pride and joys Vilppu, Veikka, and Venni.

Otaniemi 5.10.2010
Vesa Vaskelainen

List of publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I P. R. J. Östergård, V. Vaskelainen, and A. Mosig, Algorithms for the combinatorial best barbeque problem, MATCH Commun. Math. Comput. Chem. **58** (2007), 309–321.

II L. Kiviluoto, P. R. J. Östergård, and V. P. Vaskelainen, Algorithms for finding maximum transitive subtournaments, Helsinki University of Technology, Department of Communications and Networking, Report 5/2010, Espoo, 2010.

III V. P. Vaskelainen and V. Riihimäki, Bounds for the sample size in performance testing of combinatorial algorithms, InterStat, July 2009, #6 (electronic).

IV P. R. J. Östergård and V. P. Vaskelainen, Russian doll search for the Steiner triple covering problem, to appear in Optim. Lett., 8 pages.

V L. Kiviluoto, P. R. J. Östergård, and V. P. Vaskelainen, Sperner capacity of small digraphs, Adv. Math. Commun. **3** (2009), 125–133.

VI P. R. J. Östergård and V. P. Vaskelainen, A tournament of order 14 with disjoint Banks set and Slater set, Discrete Appl. Math. **158** (2010), 588–591.

The author of this thesis wrote the first versions of the articles [**I**,**III**,**IV**,**VI**], and took part in writing [**II**,**V**]. The author of this thesis designed the algorithms in [**I**,**IV**,**VI**], and part of the algorithms in [**II**]. These algorithms were implemented and experimentally evaluated solely by the author. In [**III**] the experimental methodologies were jointly developed with the co-author.

Chapter 1

Introduction

Discrete optimization problems arise from a wide variety of real-life situations. An everyday example of a challenging combinatorial optimization problem is the airline crew scheduling problem [64]. Optimal solutions for instances of the crew scheduling problem can produce significant economical savings for airline companies. Another good example is related to electronic circuits. By solving an instance of the minimum feedback vertex set problem [5], the number of components in hardware can be reduced to its minimum so that the testability of the electronic circuit is ensured [40].

Any instance of a finite combinatorial optimization problem can be solved in finite time and space. However, the required time can be impractically long or the required space can be unfeasibly large; for this reason, there will always be unsolved instances of finite combinatorial optimization problems. Complexity theory classifies combinatorial problems in terms of complexity classes [54]. Problems in the same complexity class share a common property; namely, that they can all be solved within the same bound of a required resource.

A desirable property for a practical algorithm is polynomial time complexity. Problems that can be solved in polynomial time belong to a complexity class called **P**. Nevertheless, for several discrete optimization problems there is still no known polynomial time algorithm. Moreover, it is widely believed that, for some of these problems, such algorithms do not even exist. Problems that require a "yes" or "no" answer, wherein the "yes" answer can be verified in polynomial time, belong to a complexity class called **NP**. If a problem in class **NP** is at least as hard as any other problem in class **NP**, then the problem is **NP**-complete [21]. **NP**-completeness is a valuable con-

cept in algorithm design. If a problem is known to be **NP**-complete, then the efforts to develop exhaustive algorithms can be directed towards designing exponential algorithms that are practical for small or special instances. For some applications near optimal solutions to **NP**-complete problems may be adequate and then the use of approximation algorithms is the better choice than the use of exhaustive algorithms.

Graphs such as undirected graphs, digraphs, and hypergraphs provide formalisms in which many discrete optimization problems can be presented. These formalisms form a general framework in which the application specific terminology is absent. Furthermore, connections between the optimization problems in graphs can be observed when the problems are presented within the same framework.

An optimal solution for an instance of a discrete optimization problem can be found by using exhaustive algorithms. Backtracking [24, 26, 34] is a basic method for solving a discrete optimization problem. Simple backtracking searches through an entire search space and is guaranteed to find an optimal solution. More advanced algorithms improve backtracking by using pruning to reduce the search space. Typically, pruning is based on lower or upper bounds that can be pre-calculated or evaluated during the search. Moreover, the symmetry of a discrete structure may allow pruning, which can then significantly diminish the required computational time.

Analysis of algorithms is an essential part of the algorithm research. Traditional algorithm analysis has produced asymptotic time bounds for CPU performance. In this dissertation, algorithms are evaluated by computational experiments that aim at comparability and reproducibility [10]. The performance of polynomial time algorithms can often be evaluated with mathematical methods in order to choose a suitable algorithm for a task, but similar comparison between exponential time algorithms is seldom adequate to make such an inference.

Here, algorithms for discrete optimization problems are studied. In Chapter 2, the concepts and terminology related to combinatorial optimization in graphs are introduced, and the exact definitions for the problems examined in [**I**,**II**,**IV**] are given. In Chapter 3 some exhaustive algorithms related to the algorithms used in [**I**,**II**,**IV**,**V**,**VI**] are presented. The analysis of computational experiments is discussed in [**III**] and also dealt with in Chapter 3. The dissertation is concluded in Chapter 4.

Chapter 2

Some problems in digraphs and hypergraphs

This chapter offers some definitions used in graph theory, as well as definitions of certain discrete optimization problems. The definitions used in graph theory are limited to those that appear in the descriptions of the discrete optimization problems in this dissertation.

2.1 Graphs

Many combinatorial optimization problems can be presented in terms of graphs. The following three subsections gather basic definitions of undirected graphs, digraphs, and hypergraphs.

2.1.1 Undirected graphs

The notation and definitions used in this section follow Diestel's textbook [17]. An introductory level text to graph theory is [65]. An *undirected graph* $G = (V, E)$ consists of a set of *vertices* V and a set of *edges* E, where

$$E \subseteq \{\{a, b\} : a \neq b \,\text{and}\, a, b \in V\}.$$

The notations $V(G)$ and $E(G)$ refer to the vertices of G and the edges of G, respectively. The *complement* graph $\bar{G}$ of $G = (V, E)$ is the graph,

the vertices of which are V and edges

$$\{\{a,b\} : a \neq b \,\text{and}\, a,b \in V\}\backslash E.$$

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$, if

$$V' \subseteq V \text{ and } E' \subseteq E,$$

where each edge $\{a,b\} \in E'$ has its *ends* $a, b \in V'$. A subgraph $G' = (V', E')$ of $G = (V, E)$ is an *induced subgraph* of G if

$$E' = \{\{x,y\} \in E : x, y \in V'\}.$$

A vertex $x \in V(G)$ is *incident* with an edge $e \in E(G)$, if $e = \{x,y\}$. Similarly, a vertex $y \in V(G)$ is incident with an edge $e \in E(G)$, if $e = \{x,y\}$. Vertices $x, y \in V(G)$ are *adjacent*, if $\{x,y\} \in E(G)$. An *independent set* is a set of vertices that are pairwise non-adjacent. A *vertex cover* of $G = (V, E)$ is a set $U \subseteq V$ such that every edge of E is incident with a vertex in U. An *edge cover* of $G = (V, E)$ is a set $S \subseteq E$ such that every vertex of V is incident with an edge in S. A *complete* graph is an undirected graph in which all vertices are pairwise adjacent. A *clique* is a set of vertices that are pairwise adjacent. The *clique number* $\omega(G)$ is the number of vertices in a maximum clique in G. A clique in an undirected graph $G = (V, E)$ is an independent set in the complement graph $\bar{G}$, in which the complement of the independent set with respect to V is a vertex cover of $\bar{G}$.

A *path* of length k in $G = (V, E)$ is a sequence of distinct vertices $P = x_0x_1 \ldots x_k$ so that there are edges

$$\{\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_k\}\} \in E(G).$$

The vertices $x_0, x_k \in V(G)$ are said to be *linked* by the path P. If there is a path $P = x_0x_1 \ldots x_k$ in $G = (V, E)$ with $k \geq 2$ and an edge $\{x_k, x_0\} \in E(G)$, then the sequence of vertices $x_0x_1 \ldots x_kx_0$ is a *cycle*. If there are no cycles in $G = (V, E)$, then G is *acyclic*. An undirected graph $G = (V, E)$ is *connected* if all the pairs of distinct vertices $x_i, x_j \in V(G)$ are linked by a path. A set $S \subset V$ is a *separating vertex set* of $G = (V, E)$ if the induced subgraph $G(V\backslash S)$ is disconnected. A *component* of $G = (V, E)$ is a maximal connected subgraph of G. A *tree* T is an undirected graph that is both acyclic and connected. Let the notation xTy denote the unique path in a tree T between vertices x and y. By defining

$$x \leq y \text{ if } x \in rTy$$

one obtains a partial ordering on $V(T)$. The least element of this partial order, r, is the *root* of the tree T. If the root of the tree is fixed, then the tree is a *rooted tree*. A *binary tree* is a tree T in which every vertex $v \in V(T)$ has zero, one, or two *neighbour* vertices v', $\{v, v'\} \in E(T)$, for which $v \leq v'$.

2.1.2 Digraphs

The notation and definitions used in this section follow those stated in Bang-Jensen's and Gutin's textbook [5]. A *directed graph* or *digraph* $D = (V, A)$ consists of a set of *vertices* V and a set of ordered pairs

$$A \subseteq V \times V$$

called *arcs*. The notations $V(D)$ and $A(D)$ refers to the vertices of D and the arcs of D, respectively. A *complement* digraph $\bar{D}$ of D is a digraph with vertices $V(D)$ and with arcs

$$(V(D) \times V(D)) \backslash A(D).$$

A digraph $F = (V', A')$ is a *subdigraph* of $D = (V, A)$, if

$$V' \subseteq V \text{ and } A' \subseteq A,$$

where each arc $xy \in A'$ has a *tail* $x \in V'$ and a *head* $y \in V'$. A subdigraph is *induced* if

$$A(F) = \{xy \in A(D) : x, y \in V(F)\}.$$

An *orientation* of an undirected graph G is a digraph obtained by replacing each edge $\{x, y\} \in E(G)$ by either an arc xy or an arc yx.

A *tournament* is a digraph that is an orientation of a complete graph. A tournament is *transitive* if the occurrence of the arcs xy and yz implies the occurrence of the arc xz. A *subtournament* is a subdigraph that is a tournament. A *cycle* is a sequence $x_0x_1x_2 \ldots x_k$ of distinct vertices $x_i \in V(D)$ so that there are arcs

$$\{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k, x_kx_0\} \in A(D)$$

where $k \geq 2$. An *acyclic* digraph has no cycles. A *feedback vertex set* in a digraph D is a set of vertices S so that the graph induced by $V(D)\backslash S$ is acyclic. The set of vertices S of a transitive subtournament in a digraph D is an acyclic induced subdigraph in the complement digraph $\bar{D}$, and vice versa [**II**, Theorem 1]. Furthermore, the set $V(D)\backslash S$ is a feedback vertex set in $\bar{D}$.

2.1.3 Hypergraphs

Hypergraphs are generalizations of graphs. Some definitions on hypergraphs are given here and more can be found in [7]. Formally, a *hypergraph* is a collection of subsets

$$H = \{E_1, E_2, \ldots, E_m\},$$

where each *hyperedge* E_i is a subset of a finite set of vertices

$$X = \{x_1, x_2, \ldots, x_n\}.$$

The *complement* of a hypergraph H with vertex set X is a hypergraph $\bar{H} = \mathcal{P}(X) \backslash H$, where $\mathcal{P}(X)$ denotes the *power set* of X, the set of all the subsets of X. The *dual* of a hypergraph H is a hypergraph,

$$H^* = \{X_1, X_2, \ldots, X_n\}$$

the set of vertices of which $E = \{e_1, e_2, \ldots, e_m\}$ correspond to the hyperedges of H. The hyperedges of H^* are defined by

$$X_i = \{e_j : x_i \in E_j \in H\}.$$

Concepts defined in graphs can be generalized to hypergraphs. An independent set S in a hypergraph H is a subset of X, so that for every hyperedge E_i of H it holds that $E_i \nsubseteq S$. A vertex cover for a hypergraph H is a subset of X that intersects every hyperedge $E_i \in H$. An edge cover for a hypergraph H is a subset of hyperedges of H, the union of which equals X. A vertex cover of size t in H corresponds to an edge cover of size t in H^*.

2.2 Computational complexity

Computational complexity theory [54] enables us to determine which computational problems are hard to solve by computers. Computers solve computational problems with algorithms. Therefore, formal treatment of complexity theory requires a formalized notion of an algorithm, which in turn requires a fixed model of computation (for example a program for a deterministic one-tape Turing machine which recognizes languages) [21]. This section presents some complexity theoretic definitions informally in terms of problems and algorithms (instead of a formal approach in terms of languages and Turing machines).

A large part of complexity theory applies only to decision problems. These problems have two possible solutions, the answer is either "yes" or "no". A *decision problem* Π consists of a set of *instances* D_Π (finite objects such as integers, sets, sequences, graphs, digaphs, hypergraphs, etc.) and a subset $Y_\Pi \subseteq D_\Pi$ which denotes the instances for which the answer is "yes" [21]. A *complexity class* is a set of problems which share the property that they can all be solved within the same bound of a required resource, which is in most cases time or space. For any problem in class **P** there exists a polynomial time algorithm. For any instance of the problem in class **NP** for which the answer is "yes", the following holds. There exists a proof that the answer for the instance is "yes" which can be verified by a polynomial time algorithm. The question of equality (or nonequality) of the classes **P** and **NP** is one of the most important in computer science. In order to examine relationships between complexity classes **P** and **NP** the definition of a polynomial transformation [21] is given in Definition 1.

Definition 1 A *polynomial transformation* from a decision problem Π_1 to a decision problem Π_2 is a function $f : D_{\Pi_1} \to D_{\Pi_2}$ that is computable by a polynomial time algorithm and the following is true for all instances $x \in D_{\Pi_1}$: $x \in Y_{\Pi_1}$ iff $f(x) \in Y_{\Pi_2}$.

Decision problems are ordered in respect to their difficulty by polynomial transformation, since polynomial transformation is transitive. That is, if there are decision problems Π_1, Π_2, Π_3, and a polynomial transformation from Π_1 to Π_2 and a polynomial transformation from Π_2 to Π_3, then there is also a polynomial transformation from Π_1 to Π_3. The maximal elements of this order are particularly interesting. Note that Definition 1 gives just one possible way to define what it means that a problem is at least as hard as another one. In general, a polynomial transformation is an example of a *reduction* between problems, other examples can be found in [54]. By applying Definition 1 to class **NP** and by using a polynomial transformation we obtain a definition of **NP**-complete problems. Such problems are decision versions of the problems discussed in this dissertation.

Definition 2 Let $\mathcal{C}$ be a complexity class, and let P be a problem in $\mathcal{C}$. The problem P is $\mathcal{C}$-*complete* if any problem in $\mathcal{C}$ can be reduced to P.

Decision problems are somewhat limited to describing practical combinatorial problems including the optimization problems studied in this dissertation. These problems can be classified to be search problems. A *search*

problem Π consists of a set of *instances* D_Π and for each instance $x \in D_\Pi$ there is a set of solutions $S_\Pi(x)$ (note that a decision problem can be presented as a search problem by setting $S_\Pi(x) = \{\text{“yes”}\}$ for each $x \in Y_\Pi$ and $S_\Pi(x) = \emptyset$ for each $x \notin Y_\Pi$). An algorithm *solves* a search problem Π for a given input $x \in D_\Pi$ if it determines the lack of solution when $S_\Pi(x) = \emptyset$ and otherwise it returns a solution $s \in S_\Pi(x)$. A more general type of reduction, a polynomial time Turing reduction [21], is presented in Definition 3 for reductions between search problems.

Definition 3 A *Turing reduction* from a search problem Π_1 to a search problem Π_2 is an algorithm A that solves instances of Π_1 by using a hypothetical subroutine S for solving instances of Π_2 so that, if S were a polynomial time algorithm for solving instances of Π_2, then A would be a polynomial time algorithm for solving instances of Π_1.

Applying Definition 4 to class **NP** and by using a Turing reduction we obtain a definition of **NP**-hard problems. Most of the combinatorial optimization problems discussed in this dissertation are of this type.

Definition 4 Let $\mathcal{C}$ be a complexity class, and let P be a computational problem. The problem P is *$\mathcal{C}$-hard* if a $\mathcal{C}$-complete problem reducible to P exists.

According to Definition 4, if there is a Turing reduction from an **NP**-complete problem to a combinatorial optimization problem, then this combinatorial optimization problem is **NP**-hard [21]. In general, optimization versions of **NP**-complete decision problems are **NP**-hard [39]. The problems studied in this dissertation are either instances of optimization versions of **NP**-complete problems or instances of **NP**-hard problems.

2.3 The set covering problem

The decision version of the *set covering problem*, for a given set S and a collection $\mathcal{C}$ of subsets of S, asks whether there is a collection of subsets $\mathcal{X} \subseteq \mathcal{C}$, the union of which equals S and which has cardinality $|\mathcal{X}| \leq k$. This problem is **NP**-complete [21]. The optimization version of the set covering

problem is to find a collection of subsets $\mathcal{X} \subseteq \mathcal{C}$, for which $\bigcup \mathcal{X} = S$ and $|\mathcal{X}|$ is the minimum.

In the *weighted* set covering problem, a positive real valued weight is given for each subset in the collection $\mathcal{C}$. In addition, the condition for $|\mathcal{X}|$ in the decision version is replaced by the condition for the total weight of $\mathcal{X}$, $w(\mathcal{X}) \leq w_k$. The optimization version of the weighted set covering problem is to find a collection of subsets $\mathcal{X} \subseteq \mathcal{C}$, for which $\bigcup \mathcal{X} = S$ and $w(\mathcal{X})$ is the minimum. An important application of the set covering problem is *crew scheduling* in airline [64], railway [13], and mass-transit companies [14]. The optimization version of the weighted set covering problem can be formulated as an integer linear program.

Problem 1 ***The optimization version of the set covering problem***

$$\begin{array}{ll} \textit{minimize} & \boldsymbol{c}^\top \boldsymbol{x} \\ \textit{subject to} & \boldsymbol{M}\boldsymbol{x} \geq \boldsymbol{e}, \end{array}$$

where $\boldsymbol{c}$ *is a constant* $n \times 1$ *vector,* $\boldsymbol{x}$ *is an* $n \times 1$ $(0,1)$*-vector,* $\boldsymbol{M}$ *is an* $m \times n$ $(0,1)$*-matrix, and* $\boldsymbol{e}$ *is an* $m \times 1$ *vector of* 1*s.*

If Problem 1 is unweighted, then the constants c_i are equal to 1, and if it is weighted, then the constants c_i are greater than 0. The optimization version of the unweighted set covering problem is considered here.

An equivalent to the set covering problem is the hitting set problem. For a set S and a collection $\mathcal{F}$ of subsets of S, a subset $B \subseteq S$ is a *hitting set* if B intersects all members of $\mathcal{F}$. The decision version of the *hitting set problem*, given S and $\mathcal{F}$, asks whether there is a hitting set B which has cardinality $|B| \leq k$. The optimization version of the hitting set problem is to find a hitting set with minimum cardinality.

An instance of the hitting set problem can be translated to a hypergraph by taking the set S as the set of vertices and the collection $\mathcal{F}$ as the set of hyperedges. A hitting set in the original instance is a vertex cover in the constructed hypergraph. In the dual of the hypergraph, the hitting set is an edge cover, which, in turn, translates back to the instance of the set cover problem.

An instance of the set covering problem in which the $(0,1)$-matrix M is an incidence matrix of a certain *Steiner triple system*, which is defined in [**IV**]

on page 2. is called a *Steiner triple covering problem.* A Steiner triple covering problem is considered in many papers on algorithms, including [20, 44, 50, 53]. Usually, the Steiner triple covering problem is presented as a hitting set problem. An instance of the Steiner triple covering problem is solved in [**IV**] in the framework of the hitting set problem.

2.4 The maximum transitive subtournament problem

The decision version of the *transitive subtournament problem* for a given digraph asks whether there is a subdigraph of order k that is a transitive tournament. The corresponding feedback vertex set problem is **NP**-complete [5, 60]. The optimization version of the transitive subtournament problem for a given digraph is to find a subdigraph that is a transitive tournament with the maximum number of vertices. The maximum transitive subtournament problem is formulated as follows.

Problem 2 ***The maximum transitive subtournament problem***
Given a digraph $D = (V, A)$, find a maximum transitive subdigraph of D.

As was stated in Section 2.1.2, searching for a maximum transitive subtournament in digraph D is equivalent to searching for a maximum induced acyclic subdigraph in $\bar{D}$. An instance of the maximum induced acyclic subdigraph problem can be expressed in terms of a hypergraph. The set of vertices of the hypergraph is $V(\bar{D})$ and the vertices of each cycle in digraph $\bar{D}$ form one hyperedge. A maximum independent set in this hypergraph corresponds to a maximum induced acyclic subdigraph in $\bar{D}$. The complement with respect to $V(\bar{D})$ of a maximum independent set is a minimum vertex covering. As we recall from Section 2.3, a hitting set that is a solution to an instance of the hitting set problem is a vertex cover in the hypergraph corresponding to that instance. Therefore, the maximum transitive subtournament problem can be reduced to the minimum hitting set problem.

An approach commonly used in the pertinent literature to solve instances of the maximum transitive subtournament problem in a digraph D is that of solving the minimum feedback vertex set problem in the complement digraph $\bar{D}$ [8, 43, 51, 59]. If there are a small number of cycles in D, then

the size of the maximum transitive subtournament in D and the size of the maximum induced acyclic subdigraph in $\bar{D}$ are expected to be large, with the size of the minimum feedback vertex set in $\bar{D}$ being small. Due to the small size of the solution, the problem is often computationally faster to solve by finding the minimum feedback vertex set in $\bar{D}$. However, for some applications, it is preferable to search for transitive subtournaments or induced acyclic subdigraphs. This is the case at least in the first two applications described in following section.

2.4.1 Applications for transitive subtournaments

Applications for transitive subtournaments include determining the Sperner capacity, tournament solutions, and testing electronic circuits. These three applications are described in this section. Testing electronic circuits is expressed here as an application of a minimum feedback vertex set.

Sperner capacity is a generalization of Shannon capacity to digraphs. In order to define the concepts of Shannon and Sperner capacity, some auxiliary definitions are needed. Consider an undirected graph G that has one vertex for each input symbol of a discrete communication channel and an edge between two vertices if and only if the two symbols cannot lead to the same output symbol (due to channel errors). The graph G is known as the *characteristic graph* of the channel. The clique number $\omega(G)$ of the characteristic graph $G = (V, E)$ gives the largest set of symbols that can be used for transmitting one symbol over the channel error-free, and the amount of information transmitted is $\log_2 \omega(G)$ bits per transmission.

For undirected graphs G_1 and G_2, the co-normal product $G_1 \cdot G_2$ has the set of vertices

$$V(G_1 \cdot G_2) = V(G_1) \times V(G_2)$$

and the set of arcs

$$E(G_1 \cdot G_2) = \{\{\{x_1, y_1\}, \{x_2, y_2\}\} : \{x_1, x_2\} \in E(G_1) \text{ or } \{y_1, y_2\} \in E(G_2)\}.$$

For an undirected graph $G = (V, E)$ the (co-normal) power G^n has the vertex set $V^n = \{(u_1, u_2, \ldots, u_n) : u_i \in V(G)\}$, and $(u_1, u_2, \ldots, u_n)$ and $(v_1, v_2, \ldots, v_n)$ are connected by an edge if and only if there is an i such that $\{u_i, v_i\} \in E(G)$. If information is transmitted in blocks of size n, then the information rate is

$$\frac{\log_2 \omega(G^n)}{n} = \log_2 \omega(G^n)^{1/n}$$

bits per transmission, whereas the Shannon's *zero-error capacity* [58] of the channel is

$$\sup_{n\geq 1} \log_2 \omega(G^n)^{1/n}$$

bits per transmission. When this problem is studied purely in the context of graphs, one ignores the logarithm and defines the zero-error capacity of G as

$$\Theta(G) := \sup_{n\geq 1} \omega(G^n)^{1/n}.$$

For digraphs D_1 and D_2, the *co-normal product* $D_1 \cdot D_2$ has the set of vertices

$$V(D_1 \cdot D_2) = V(D_1) \times V(D_2)$$

and the set of arcs

$$A(D_1 \cdot D_2) = \{(x_1y_1, x_2y_2) : x_1x_2 \in A(D_1) \text{ or } y_1y_2 \in A(D_2)\}.$$

For a digraph $D = (V, A)$, the (co-normal) power graph D^n has the vertex set $V^n = \{(u_1, u_2, \ldots, u_n) : u_i \in V(D)\}$, and there is an arc from $(u_1, u_2, \ldots, u_n)$ to $(v_1, v_2, \ldots, v_n)$ if and only if there is an i such that $u_iv_i \in A(D)$. The (non-logarithmic) *Sperner capacity* [22] of D is then

$$\sigma(D) := \sup_{n\geq 1} \omega_t(D^n)^{1/n},$$

where $\omega_t(D)$ is the size of the maximum transitive subtournament.

The theoretical aspects of Sperner capacity are studied in [37, 56]. Sperner capacity can be used to determine the zero-error capacity of a compound channel [49] and has been applied in extremal set theory [23]. In this dissertation Sperner capacity was determined for all but eight digraphs up to five vertices in [**V**] by using algorithms from [**II**]. Next, an application related to an election is considered.

Let V be the set of candidates and for each pair x, y of distinct candidates let m_{xy} be the number of voters who prefer x to y. If $m_{xy} > m_{yx}$, then x is preferred to y by the majority of voters. The result of this method can be presented as a digraph $D = (V, A)$, where V is the set of candidates, and there is an arc $xy \in A$ whenever $m_{xy} > m_{yx}$. If the resulting digraph $D = (X, A)$ is a tournament, it is called a *majority tournament* [31].

Methods for choosing the winner of a tournament are called *tournament solutions* [41]. Simple majority is one way of choosing the winner of elections and has been in use for centuries [18]. Sophisticated ways of choosing

the winners of a tournament can be found in [15, 41]. Some tournament solutions are: a *Copeland winner* of a tournament is a vertex x with a maximum out-degree, a *Banks winner* of a tournament is the first vertex of a maximal (with respect to inclusion) transitive subtournament, and a *Slater winner* of a tournament is the first vertex of a *Slater order*, which is a transitive tournament obtained by reversing the minimum number of arcs in the tournament. The *Copeland set*, the *Banks set*, and the *Slater set* are sets of all Copeland, Banks, and Slater winners of a tournament, respectively. Determining the Slater set and the Banks set are both **NP**-hard problems [3, 9, 31, 66]. Slater winners and Banks winners are examined in [**VI**].

Lastly, an application for transitive subtournaments, which is expressed as an application of a minimum feedback vertex set, is discussed. The testing of electronic circuits [5, 42] is an important application of the minimum feedback vertex set. An electronic circuit can be modeled by a digraph $D = (V, A)$ so that the components of the circuit — primary inputs, primary outputs, and outputs of the gates and flip flops — correspond to the set of vertices V, and the set of arcs A consists of the arcs vw, for which v is an input of a component with an output w. A *storage element* graph $D' = (V', A')$ [40] is a subdigraph of D, which is defined as follows. Terminals and outputs of flip flops correspond to the vertices $V' \subseteq V$ and there is an arc from a vertex $v \in V'$ to $w \in V'$ if there is a path from v to w in the digraph D. By solving the feedback vertex set problem in the digraph D', the minimum number of flip flops that must be directly accessible in order to ensure the testability of the electronic circuit is obtained [40].

2.5 The best barbeque problem

The best barbeque problem is a combinatorial optimization problem that asks for the largest intersection of n sets that are taken one from each of n given collections of subsets of a universal set $\{1, 2, \ldots, m\}$. The collections of subsets fully specify an instance of the combinatorial best barbeque problem. This combinatorial optimization problem arises in the context of discovering so-called cis-regulatory modules in regulatory DNA sequences, and the decision version of the problem is **NP**-complete [48]. The name of the problem originates from an analogy with a barbeque party with n guests. For each guest there is a plate with m different barbeque ingredients arranged randomly in multiple layers. Then each guest prepares one skewer for themselves by stabbing just once into their plate. One possible outcome

is that the set of ingredients common to all is maximized.

Problem 3 ***Combinatorial best barbeque problem.*** *For positive integers m and n, let $\mathcal{C}_i$, $1 \leq i \leq n$ be collections of subsets of $\{1, 2, \ldots, m\}$ and denote the subsets in a collection $\mathcal{C}_i$ by $C_{i,j}$, where $1 \leq j \leq |\mathcal{C}_i|$. Determine the maximum value of*

$$\left| \bigcap_{i=1}^{n} C_{i,\nu(i)} \right|,$$

where $\nu : \{1, \ldots, n\} \to \mathbb{Z}^+$ with $\nu(i) \leq |\mathcal{C}_i|$ and optimization is done over all such ν.

A combinatorial best barbeque problem with a *support parameter* $1 \leq \sigma \leq n$ is to find the maximum value of

$$\left| \bigcap_{i=1}^{\sigma} C_{t(i),\nu(t(i))} \right|,$$

where t is a bijection $t : \{1, \ldots, n\} \to \{1, \ldots, n\}$, and optimization is done over all t and ν.

A problem related to the combinatorial best barbeque problem is *frequent itemset mining* [2, 48]. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items. A subset of the set of items $X \subseteq \mathcal{I}$ is called an *itemset*. A pair $T = (id, I)$ is a *transaction* over $\mathcal{I}$ where id is an identifier of the transaction and I an itemset. The set of transactions is a *transaction database* $\mathcal{D}$ over $\mathcal{I}$. A transaction $T = (id, I)$ *supports* an itemset X if the itemset $X \subseteq I$. An itemset X is *frequent* if the number of transactions $supp(X, \mathcal{D})$ in the database $\mathcal{D}$ that support the itemset X is greater or equal to the *minimal support threshold* σ_{abs}. Given a set of items $\mathcal{I}$, a transaction database $\mathcal{D}$ over $\mathcal{I}$, and the minimal support threshold σ_{abs}, the problem of frequent itemset mining [25] is to find the collection of frequent itemsets $\mathcal{F}(\mathcal{D}, \sigma_{\text{abs}}) = \{X \subseteq \mathcal{I} : supp(X, \mathcal{D}) \geq \sigma_{\text{abs}}\}$. Problem 4 corresponds to the best barbeque problem with the support parameter σ and $|\mathcal{C}_i| = 1$ for all $1 \leq i \leq n$.

Problem 4 ***Frequent itemset with maximum cardinality***. *Let $\mathcal{I}$ be a set of items, $\mathcal{D}$ be a transaction database over $\mathcal{I}$, and σ_{abs} be the minimal support threshold. Determine the maximum value of $|X|$ so that a frequent itemset $X \subseteq \mathcal{I}$ and $supp(X, \mathcal{D}) \geq \sigma_{abs}$.*

In frequent itemset mining, itemsets are subsets of the set of items. In the combinatorial best barbeque problem with limited support, we are given collections of subsets of the set of items. Therefore, the combinatorial best barbeque problem, with limited support, is the natural generalization of the frequent itemset mining. More information on frequent itemset mining can be found in [29].

An instance of the combinatorial best barbeque problem translates straightforwardly to a combinatorial problem in a hypergraph. Let the universal set $\{1, 2, \ldots, m\}$ correspond to a set of vertices X and a subset $\mathcal{C}_{i,j}$ to a hyperedge, the colour of which is i. The combinatorial problem in the hypergraph is the search for a subset of vertices $U \subset X$ with maximum cardinality so that U is a subset of at least one hyperedge of each colour.

Chapter 3

Exhaustive search algorithms

In this chapter, algorithms are considered in the framework of constraint satisfaction problems and valued constraint satisfaction problems [57, 63]. The optimization problems discussed in Chapter 2 can be presented as valued constraint satisfaction problems. Consideration of the optimization problems can be limited to minimization problems, since maximization problems can always be straightforwardly transformed to minimization problems. More information on constraint satisfaction problems can be found in [16].

To begin with, this chapter introduces the concept of the backtrack search algorithm. Secondly, more advanced variants of the basic backtrack algorithm are discussed. Finally, the principles of computational experiments are surveyed. A backtrack algorithm can be used to solve instances of the constraint satisfaction problem. A modification of backtracking, namely the depth-first branch-and-bound algorithm, can be used to solve instances of the valued constraint satisfaction problem [62]. Dynamic programming and Russian doll search are more advanced methods based on backtrack search. They can be also used to solve instances of the valued constraint satisfaction problem.

Problem 5 ***Constraint satisfaction problem*** *A constraint satisfaction problem is defined by a set of variables* $X = \{x_1, x_2, \ldots, x_n\}$*, with respective finite domains* $D = \{D_1, D_2, \ldots, D_n\}$ *that are sets of all of the possible values for each variable, and by the set of constraints* $C = \{(X_1, R_1), (X_2, R_2), \ldots, (X_m, R_m)\}$ *where each* $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,r}\} \subseteq X$ *and* $R_i \subseteq D_{i,1} \times D_{i,2} \times \cdots \times D_{i,r}$ *is a relation on the variables* X_i*. Relations*

$R_1, R_2, \ldots, R_m$ denote simultaneously legal values for the variables in the subsets $X_1, X_2, \ldots, X_m$, respectively. An instance of the constraint satisfaction problem is denoted by a triple (X, D, C). The objective is to find a solution or determine that there is no solution.

Problem 6 ***The valued constraint satisfaction problem***. *A valued constraint satisfaction problem is defined by an instance of the constraint satisfaction problem (X, D, C); by a valuation structure $S = (E, \succ, \otimes)$, where E is a set, $\succ$ is a total order on E, and $\otimes$ is an operation on E; and by an application φ from C to E, which assigns a valuation to each constraint. Let A be a complete assignment for an instance of the valued constraint satisfaction problem and $C_{viol}(A)$ the set of the constraints that are violated by the assignment A. The valuation $\varphi(A)$ is defined by*

$$\varphi(A) = \bigotimes_{c \in C_{\text{viol}}(A)} \varphi(c).$$

The objective is to find a complete assignment that minimizes $\varphi(A)$.

Note that, in both Problems 5 and 6, the search space is finite. Therefore, with sufficient running time, a solution is guaranteed to be found for an instance of the constraint satisfaction problem, or, alternatively a lack of a solution will be determined. Similarly, an optimal solution will be found to an instance of the valued constraint satisfaction problem. Example 1 connects Problem 6 to the hitting set problem introduced in Section 2.3. Constraints in Example 1 are given as numeric constraints, that is, they are written by using arithmetic expressions.

Example 1 ***An instance of the hitting set problem formulated as an instance of the valued constraint satisfaction problem***. *Recall from Section 2.3 how an instance of the hitting set problem can be translated to a problem on a hypergraph. Let a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ represents vertices of a hypergraph H (Section 2.1.3). Each x_i has the domain $D_i = \{0, 1\}$, $i \in \{1, 2, \ldots, n\}$. The set of constraints C consists of one constraint for each vertex of H, $C_i : x_i = 0$, $i \in \{1, 2, \ldots, n\}$, and one constraint for each hyperedge of H,*

$$C_{n+j} : \sum_{x_v \in E_j} x_v \geq 1,\ j \in \{1, 2, \ldots, m\}.$$

A suitable valuation structure is $S = (\mathbb{N} \cup \infty, >, +)$ *and an application* φ *from* C *to* $\mathbb{N} \cup \infty$ *is*

$$\varphi(c) = \begin{cases} 1 & , c \in \{C_1, C_2, \ldots, C_n\} \\ \infty & , c \in \{C_{n+1}, C_{n+2}, \ldots, C_{n+m}\}. \end{cases}$$

3.1 Backtracking

Backtracking [24] builds up feasible solutions for instances of the constraint satisfaction problem, one step at a time, by exhaustively covering all the possibilities in a systematic fashion. Algorithm 1 presents the formulation of backtracking for a constraint satisfaction problem as a search for the product space $D^n = D_1 \times D_2 \times \cdots \times D_n$ and is invoked with $search(() , 0)$. A feasible solution satisfies all constraints C_i, where $i \in \{1, 2, \ldots, m\}$. Let $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,t}\}$ where $t = |X_i|$. Thus the relation R_i is a subset of $D_{i,1} \times D_{i,2} \times \cdots \times D_{i,t}$.

Algorithm 1 General backtrack algorithm for a constraint satisfaction problem

Find all $(x_1, x_2, \ldots, x_n) \in (D_1, D_2, \ldots, D_n)$ so that $(x_{i,1}, x_{i,2}, \ldots, x_{i,t}) \in R_i$ for all $i \in \{1, 2, \ldots, m\}$.

procedure $search((x_1, x_2, \ldots, x_l), l)$;
1: **if** $l = n$ **then**
2: Save the current solution $(x_1, x_2, \ldots, x_n)$
3: **else**
4: **for each** $x_{l+1} \in D_{l+1}$ **do**
5: **if** $(x_{i,1}, x_{i,2}, \ldots, x_{i,t}) \in R_i$ for all $i \in \{1, 2, \ldots, m\}$ **then**
6: $search((x_1, x_2, \ldots, x_{l+1}), l + 1)$
7: **end if**
8: **end for**
9: **end if**
end procedure

Note that $(x_{i,1}, x_{i,2}, \ldots, x_{i,t})$ is considered in this work to be in R_i even if not all variables in X_i have been assigned, but the remaining unassigned variables can be assigned so that $(x_{i,1}, x_{i,2}, \ldots, x_{i,t}) \in R_i$.

To find a complete assignment of minimum valuation for an instance of the valued constraint satisfaction problem, a modification of backtracking is used. The depth-first branch-and-bound algorithm [12, 63] presented as

Algorithm 2 is an extension of the backtrack algorithm. Algorithm 2 finds a solution for a valued constraint satisfaction problem.

Algorithm 2 Depth-first branch-and-bound algorithm for a valued constraint satisfaction problem

Find an assignment A of the problem variables so that $\varphi(A)$ is minimized.

```
procedure dfbb()
 1: ub ← +∞
 2: sdfbb(1)
end procedure
procedure sdfbb(j)
 3: for each k ∈ D_j do
 4:    A[j] ← k
 5:    lb ← FC(A)
 6:    if lb < ub then
 7:       if j = n then
 8:          ub ← lb
 9:          Save the current assignment A
10:       else
11:          sdfbb(j + 1)
12:       end if
13:    end if
14: end for
end procedure
```

In Algorithm 2, A is a global array that stores the current assignment, ub is a global variable that stores the valuation of the best assignment found so far, and lb is a global variable that stores the lower bound for a current partial assignment. The lower bound lb bounds from below the valuation of the best complete assignment that can be extended from the current partial assignment. If $j = n$, then lb equals the valuation of the current complete assignment. For an assignment in A, partial or complete, a *forward checking* method (FC) [16, 19, 30] computes lb in line 5. Therefore, when $lb \geq ub$ in line 6, the current partial assignment cannot be extended to a better complete assignment than the current best (or the complete assignment is worse than the current best) and Algorithm 2 backtracks. The depth-first branch-and-bound search can be improved by the choice of good variable ordering or value ordering, or by having a good forward checking function that computes large lower bounds early. Example 2 describes the use of Algorithm 2 for solving an instance of the problem presented in Example 1 in page 18.

Example 2 ***Proceeding of Algorithm 2 for an instance of the valued constraint satisfaction problem***. *Let $n = 4$ in Example 1 and set of constraints C consists of $C_i : x_i = 0$, $i \in X$, $C_5 : x_1 + x_2 + x_4 \geq 1$, $C_6 : x_2 + x_3 \geq 1$, and $C_7 : x_3 + x_4 \geq 1$. Let variable ordering be (x_1, x_2, x_3, x_4), value ordering $(0, 1)$, and a forward checking function*

$$\mathrm{FC}(A) = \sum_{c \in C_{\mathrm{viol}}(A)} \varphi(c) + \sum_{q \in \{j+1, j+2, \ldots, n\}} \min_{k \in D_q} \Big(\sum_{c \in C_{\mathrm{viol}}(A_q)} \varphi(c) \Big)$$

where

$C_{\mathrm{viol}}(A) = \{c \in C : c$ *is assigned and violated by* $A[i], i \in \{1, 2, \ldots, j\}\}$,
and $A_q = A[i], i \in \{1, 2, \ldots, j, q\}, q \in \{j+1, j+2, \ldots, n\}$,

($\mathrm{FC}(A)$ *is equal to* LB_2 *in [63]). Algorithm 2 proceeds as follows.*

$A[1] = 0$, $lb = 0$
  $A[2] = 0$, $lb = \varphi(C_3) + \varphi(C_4) = 2$
    $A[3] = 0$, $lb = \varphi(C_4) + \varphi(C_6) = \infty$
    $A[3] = 1$, $lb = \varphi(C_3) + \varphi(C_4) = 2$
      $A[4] = 0$, $lb = \varphi(C_3) + \varphi(C_5) = \infty$
      $A[4] = 1$, $lb = \varphi(C_3) + \varphi(C_4) = 2$,
      $ub = 2$, Save $A = (0, 0, 1, 1)$
  $A[2] = 1$, $lb = \varphi(C_2) = 1$
    $A[3] = 0$, $lb = \varphi(C_2) + \varphi(C_4) = 2$
    $A[3] = 1$, $lb = \varphi(C_2) + \varphi(C_3) = 2$
$A[1] = 1$, $lb = \varphi(C_1) = 1$
  $A[2] = 0$, $lb = \varphi(C_1) + \varphi(C_3) = 2$
  $A[2] = 1$, $lb = \varphi(C_1) + \varphi(C_2) = 2$

Algorithm 2 terminates with a complete assignment $A = (0, 0, 1, 1)$ which minimizes $\varphi(A)$. In a hypergraph H the subset $\{x_3, x_4\} \subseteq X$ is a minimum vertex cover. In an original instance of the hitting set problem the subset $\{x_3, x_4\} \subseteq X$ is a hitting set with minimum cardinality.

Many variants of the basic branch-and-bound search are referred to in the relevant literature; including recursive best first [35], iterative deepening [36], iterative objective relaxing [36], iterative approximating [55], Russian doll search [63], iterative deepening together with Russian doll search [12], and specialized Russian doll search [46]. The aforementioned search methods, apart from the specialized Russian doll search, are experimentally compared in [12]. The specialized Russian doll search is compared to Russian doll search in [46, 47]. The following two sections consider the specialized Russian doll search and dynamic programming [38] for a valued constraint satisfaction problem.

3.1.1 Dynamic programming

Dynamic programming [6] is a method for solving problems that can be divided into subproblems. For instances of the valued constraint satisfaction problems, dynamic programming can be applied through a tree decomposition of the constraint graph.

The *tree decomposition* of a graph G is a pair $(T, \mathcal{V})$, where T is a tree, $\mathcal{V} = (V_t)_{t \in V(T)}$ is a family of vertex sets $V_t \subseteq V(G)$ indexed by vertices of T, and which satisfies the following three conditions:

(i) $V(G) = \bigcup_{t \in V(T)} V_t$,

(ii) for every edge $e = \{x, y\} \in E(G)$ a vertex $t \in V(T)$ exists such that $x, y \in V_t$,

(iii) $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$ whenever t_2 belongs to the unique path from t_1 to t_3 in T.

The *width* of $(T, \mathcal{V})$ is defined by $\max\{|V_t| - 1 : t \in V(T)\}$. The smallest width of any tree decomposition of G is called *tree-width* of G. Note that a tree has tree-width 1. Theorem 1 states an important feature of a tree-decomposition [17, Lemma 12.3.1], and Theorem 2 [17, Lemma 12.3.2] passes tree-decompositions to subgraphs.

Theorem 1 *Let $\{t_1, t_2\} \in E(T)$ be any edge of a tree T and let T_1, T_2 be the components of a tree*

$$T' = (V(T), E(T) \backslash \{t_1, t_2\}),$$

with $t_1 \in T_1$ and $t_2 \in T_2$. Then $V_{t_1} \cap V_{t_2}$ separates $U_1 = \bigcup_{t \in V(T_1)} V_t$ from $U_2 = \bigcup_{t \in V(T_2)} V_t$ in G.

Theorem 2 *For every subgraph H of G, the pair $(T, (V_t \cap V(H))_{t \in V(T)})$ is a tree decomposition of H.*

A *constraint hypergraph* [16] is a hypergraph in which the vertices correspond to the set of variables $X = \{x_1, x_2, \ldots, x_n\}$, and the hyperedges to the subsets $X_i \subseteq X$ in the set of constraints

$$C = \{(X_1, R_1), (X_2, R_2), \ldots, (X_m, R_m)\}.$$

A *constraint graph* is a special case of the constraint hypergraph in which each X_i has a cardinality of at most two. The concepts *hypertree decomposition* and *hypertree-width* are defined in [28]. In the following it is assumed that every subset X_i involves two variables at the most.

The idea of the dynamic programming algorithm based on the tree decomposition approach is explained as follows. Let $G = (V, E)$ be a constraint graph the tree decomposition of which is $(T, \mathcal{V})$. The optimal assignment in disconnected components then depends only on the assignment in a separating vertex set S. According to Theorem 1, any edge t_1t_2 of T specifies a separating vertex set $S = V_{t_1} \cap V_{t_2}$ and disconnected components $U_1 = \bigcup_{t \in T_1} V_t$ and $U_2 = \bigcup_{t \in T_2} V_t$. According to Theorem 2,

$$(T, (V_t \cap U_1)_{t \in T}) \text{ and } (T, (V_t \cap U_2)_{t \in T})$$

are tree-decompositions for U_1 and U_2, respectively.

Algorithm 3 Tree decomposition approach

Find an optimal assignment for the vertices V of a tree decomposition of the constraint graph.

```
procedure dp()
1: for i = n downto 1 do
2:    if i is a leaf node then
3:       Compute all assignments for i
4:    else
5:       Compute all assignments for T_i from T_j and T_k
6:       Store the best assignment for T_i
7:    end if
8: end for
end procedure
```

For the sake of simplicity, the dynamic programming algorithm [38] presented in Algorithm 3 concentrates on the way in which a solution to the original problem is obtained from solutions to subproblems. In addition, it is assumed that a tree decomposition of the constraint graph is both rooted and binary. The notation used in Algorithm 3 is as follows. The set of vertices of the tree is $I = \{1, 2, \ldots, n\}$, enumeration runs from the leaves towards the roots, and a subtree rooted at vertex i is denoted by T_i. In line 5, a vertex i is assumed to be the predecessor of j and k. After termination of execution, the solution to the original problem is obtained from the stored best assignment for T_1.

Solving an instance of the valued constraint satisfaction problem with the arbitrary structure of the constraint graph may require exponential space [63]. If the treewidth of the constraint graph is bounded, then the instance of the valued constraint satisfaction problem may be solved in polynomial time. Many **NP**-complete or **NP**-hard problems are indeed solvable in polynomial time for graphs of bounded treewidth, for example maximum independent set [4].

The performance of the dynamic programming algorithm will depend on the width of the tree decomposition. For this reason, it is important to find a tree decomposition of small width. Nevertheless, the problem of finding a tree decomposition with optimal width is **NP**-hard [38]. Complexity analysis for backtracking with tree-decomposition is presented in [32, 61], and for the dynamic programming algorithm in [38].

3.1.2 Russian doll search

The Russian doll search algorithm [12, 63] is presented as Algorithm 4. The Russian doll search algorithm solves an instance of the valued constraint satisfaction problem through n successive nested subproblems. The first subproblem includes just the nth variable, the second subproblem the last two variables, and so on until the the nth subproblem equals the original problem. Each subproblem is solved by the depth-first branch-and-bound search. Recordings of the valuations of the optimal assignments for the solved subproblems help in the search for solutions to future subproblems. Dynamic programming and Russian doll search methods have common features, but they differ in the way in which they use the results from previous subproblems; the latter only uses them to improve the lower bound on the global valuation.

In Algorithm 4, A is a global array that stores the current assignment to a subproblem consisting of variables from i to n, ub is a global variable that stores the valuation of the best assignment found so far, and lb is a global variable that stores the lower bound for a current partial assignment. The lower bound lb bounds from below the valuation of the best complete assignment that can be extended from the current partial assignment for a subproblem consisting of variables from i to n. A global array $c[i]$ stores $\varphi(A)$ for the optimal valuation A of each subproblem. The lower bound lb in line 9 is evaluated by the forward checking method (FC) and a value of the $c[i]$ array.

Algorithm 4 Russian doll search algorithm for a valued constraint satisfaction problem

Find an assignment A of the problem variables so that $\varphi(A)$ is minimized.

```
procedure russiandoll
  c[n + 1] ← 0
  for i = n downto 1 do
    ub ← +∞
    rds(i, i)
    c[i] ← lb
  end for
end procedure
procedure rds(i, j)
  for each k ∈ D_j do
    A[j] ← k
    lb ← FC(A) + c[j + 1]
    if lb < ub then
      if j = n then
        ub ← lb
        if i = 1 then
          Save the current assignment A
        end if
      else
        rds(i, j + 1)
      end if
    end if
  end for
end procedure
```

The Russian doll search algorithm was originally developed for use in handling the daily management problems of an earth observation satellite [1, 63]. Other applications of the Russian doll search algorithm are, for example, the maximum clique problem [52], the radio link frequency assignment problem [11, 47], the combinatorial best barbeque problem [**I**], the maximum transitive subtournament problem [**II**], and the Steiner triple covering problem [**IV**].

3.2 Computational experiments

Mathematical methods can be used to predict consumption of the computational resources of an algorithm. With polynomial time algorithms predictions are often accurate enough to permit inferences about the suitable algorithm for a task before implementation. Examples of this are presented in [39]. With algorithms for **NP**-complete or more difficult problems, predictions for practical instance usually require experimental analysis. For the specific instance, more accurate prediction of the usage of time (or space) can be obtained by computational experiments that provide knowledge of algorithms in real-world situations [45].

The Russian doll search and depth-first branch-and-bound algorithms (Sections 3.1 and 3.1.2) are theoretically compared in [63] via two extreme instances. Let us consider valued constraint satisfaction problems P_{loose} and P_{tight}, both having n variables, each variable having the domain size of d, and both having a complete constraint graph (Section 3.1.1). The set of constraints of P_{loose} allows all d values for each variable and the set of constraints of P_{tight} denies all d values for each variable. All the valuations of P_{loose} and P_{tight} equal 0 and $n(n-1)/2$ (the number of constraints), respectively. The depth-first branch-and-bound algorithm for P_{loose} accepts the first complete assignment of the variables as an optimal solution, but the Russian doll search algorithm for P_{loose} has to examine one assignment for each subproblem before an optimal solution is found. In contrast, the depth-first branch-and-bound algorithm for P_{tight} goes through all the possible d^n complete assignments before termination, whereas the Russian doll search algorithm for P_{tight} determines the exact lower bound for a valuation based on the first assigned variable implying termination of the search in each subproblem after all values have been assigned for the first variable.

Therefore, the Russian doll search algorithm may require more computational time than the depth-first branch-and-bound algorithm in some cases; however, in other cases, the Russian doll search algorithm may use exponentially less computational time than the depth-first branch-and-bound algorithm. Computational experiments can be used to obtain more information about the cases in which the Russian doll search algorithm outperforms the depth-first branch-and-bound algorithm. Experimental evaluation of Russian doll search algorithms can be found in [46, 47, 63] and [**I**,**II**].

The experimental approach to analyzing algorithms is a relatively young branch of algorithm research. Therefore, experimental analysis of algo-

rithms is still in the developmental stage. The fundamental principles of experiments, however, are the same for experimental algorithmics as for any other field of science. A possible explanation for the slow adoption of existing standard practices especially in analyzing the results of experiments on backtrack search is the erratic behavior of the mean and the variance of run times of the backtrack search. The distribution of run times of backtrack search can be heavy-tailed in general [27]. The moments of these distributions are not all finite which may cause assumptions in the context of standard statistical methods to be erroneous.

The right tail distribution function, $P(X > x) = 1 - F(x)$, where $F(x)$ is the cumulative distribution function of the random variable X, should have an approximately linearly decaying tail in a log-log plot in case of the heavy-tailed distribution [27]. Figure 3.1 shows the empirically obtained right tail distribution functions for the run times of algorithms in [**II**] (the unit of the x-axis is one second) and presents the right tail distribution functions for standard normal distribution $\mathcal{N}(3, 1)$ and $\mathcal{N}(10^5, 5 \cdot 10^4)$ as reference material. Based on Figure 3.1 the heavy-tailedness of run time distributions

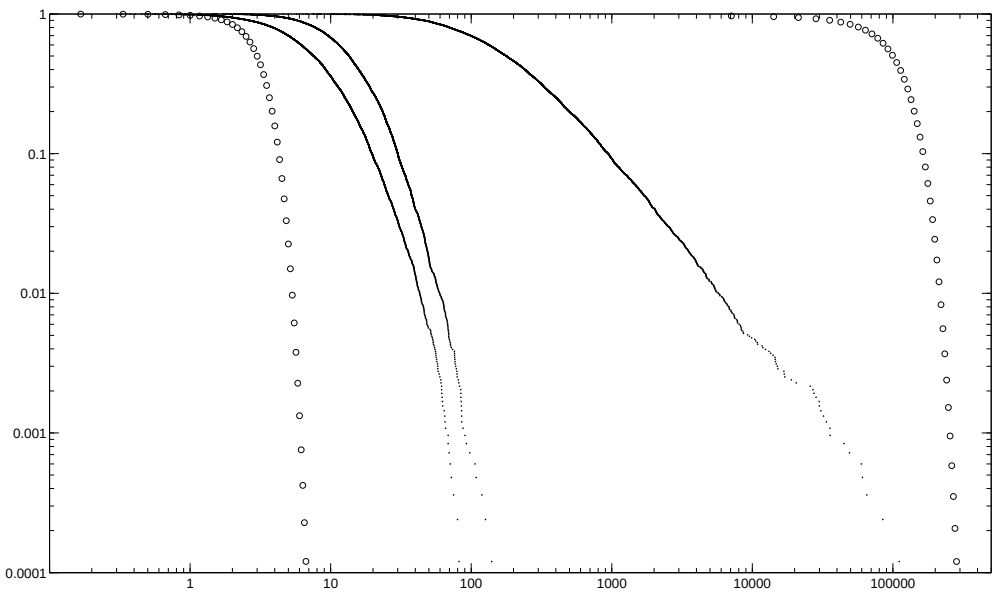

Fig. 3.1 Right tail distribution functions for $\mathcal{N}(3, 1)$, run times of algorithms A3, A2, A1 in [**II**], and $\mathcal{N}(10^5, 5 \cdot 10^4)$

of algorithms in [**II**] strongly depends on backtrack search strategy itself. Hence, the run time distribution of backtrack search with a sophisticated

pruning strategy is not necessary heavy-tailed. Standard statistical analysis tools for experimental analysis of combinatorial algorithms are studied in [**III**].

General guidelines for experimental analysis of algorithms are found in [33]. The use of statistics, including the methodology presented in [**III**], in a coherent way with [33] requires careful consideration. Statistics on run times are useful for supporting general conclusions on performance differences but too accurate results may waste computational resources on unimportant questions. However, information about the distribution of run times is essential when one estimates the accuracy of means of the run times. For this purpose it is in most cases fully adequate to present standard deviations of run times as has been done in [**II**,**IV**]. The methodology in [**III**] exploits information about the distribution in order to get precise means. Analysis of experiments in paper [**II**] benefited from the study done in [**III**], but the level of description of distributions is kept to a minimum.

Chapter 4

Conclusions

Several discrete optimization problems of practical importance are optimization versions of **NP**-complete problems for which no polynomial time algorithms are known. In this dissertation, efforts are directed towards the development of algorithms for small or special instances of optimization versions of **NP**-complete problems. As a result, real biological instances of the combinatorial best barbeque problem [**I**] and the Steiner triple covering problem of order 135 are solvable in reasonable computational time with a personal computer. Furthermore, Sperner capacity for all but eight digraphs of up to five vertices [**V**] is determined and a tournament of order 14 with disjoint Banks set and Slater set [**VI**] is found.

From among the many available exhaustive search methods, Russian doll search was selected for use in this work. The results listed above are concrete examples of the outcomes and they imply that Russian doll search is a recommendable method for solving practical discrete optimization problems. In a broad sense a general finding of this research is that the use of relatively simple algorithmic ideas can lead to practical algorithms. Easyness in implementation of the Russian doll search algorithm improve the chance to get successful results.

A further result of this research is the information obtained about the practical performance of the Russian doll search algorithm. The performance of the Russian doll search algorithms is experimentally analyzed and compared to other backtrack algorithms in [**I**] and [**II**]. The computational experiments outlined in [**II**] inspired the study of the experimental methods discussed in [**III**]. In addition, in [**I**] and [**II**], different variable orderings are studied. Nonetheless, more research would be needed in order to get a

better understanding of the influence of variable orderings on the Russian doll search algorithm.

Similarly, the results can be extended in each paper of this dissertation. Real biological instances of the combinatorial best barbeque problem could be more complex or the Russian doll search algorithm could be modified to handle a weighted or limited support version of the best barbeque problem. There are 80 non-isomorphic Steiner triple systems of order 15 that could be used to generate Steiner triple covering problems in a similar way as was done in [**IV**]. Eight instances of the Sperner capacity problem for digraphs were left open in [**V**]. In these eight instances, one digraph with five vertices is a subdigraph to all the other seven digraphs. Determination of Sperner capacity for this subdigraph might imply the solution to all the other seven instances.

Further research on Sperner capacity could be carried out by considering digraphs with six or more vertices. Additionally, open problems exist for tournament solutions, including that of the minimum size of a tournament in which the Banks set and the Slater set are disjoint [**VI**], and the minimum size of a tournament in which the Copeland set, the Slater set, and the Banks set are disjoint.

References

[1] J. C. Agn'ese, N. Bataille, D. Blumstein, E. Bensana, G. Verfaillie, Exact and approximate methods for the daily management of an earth observation satellite, in: Proc. of 5th ESA Workshop on Artificial Intelligence and Knowledge Based Systems for Space, ESA Publication Division, Noordwijk, The Netherlands, 1995.

[2] R. Agrawal, T. Imieliński, A. Swami, Mining association rules between sets of items in large databases, in: Proc. of 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93), ACM, New York, NY, 1993, pp. 207–216.

[3] N. Alon, Ranking tournaments, SIAM J. Discret. Math. 20 (2006) 137–142.

[4] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey, BIT 25 (1985) 2–23.

[5] J. Bang-Jensen, G. Gutin, Digraphs; Theory, Algorithms and Applications, Springer-Verlag, London, 2002.

[6] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[7] C. Berge, Hypergraphs, Elsevier, Amsterdam, 1989.

[8] R. Berghammer, A. Fronk, Exact computation of minimum feedback vertex sets with relational algebra, Fund. Inform. 70 (2005) 301–316.

[9] F. Brandt, F. Fischer, P. Harrenstein, The computational complexity of choice sets, in: Proc. of 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK '07), ACM, New York, NY, 2007, pp. 82–91.

[10] F. Brglez, J. A. Osborne, Performance testing of combinatorial solvers with isomorph class instances, in: Proc. of 2007 Workshop on Experimental Computer Science (ExpCS '07), ACM, New York, NY, 2007, article 13.

[11] B. Cabon, S. Givry, L. de Lobjois, T. Schiex, J. P. Warners, Radio link frequency assignment, Constraints 4 (1999) 79–89.

[12] B. Cabon, S. D. Givry, G. Verfaillie, Anytime lower bounds for constraint violation minimization problems, in: M. Maher, J.-F. Puget (eds.), Proc. 4th International Conference on Principles and Practice of Constraint Programming (CP98), LNCS 1520, Springer, Berlin, Germany, 1998, pp. 117–131.

[13] A. Caprara, M. Fischetti, P. Toth, D. Vigo, P. L. Guida, Algorithms for railway crew management, Math. Program. 79 (1997) 125–141.

[14] A. Ceder, Urban transit scheduling: Framework, review and examples, Journal of Urban Planning and Development 128 (2002) 225–244.

[15] I. Charon, O. Hudry, A survey on the linear ordering problem for weighted or unweighted tournaments, 4OR 5 (2007) 5–60.

[16] R. Dechter, Constraint Processing, Morgan Kaufmann, San Francisco, CA, 2003.

[17] R. Diestel, Graph Theory, 3rd ed., Springer-Verlag, New York, NY, 2005.

[18] P. C. Fishburn, Condorcet social choice functions, SIAM J. Appl. Math. 33 (1977) 469–489.

[19] E. C. Freuder, R. J. Wallace, Partial constraint satisfaction, Artif. Intell. 58 (1992) 21–70.

[20] D. Fulkerson, G. Nemhauser, L. Trotter, Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of steiner triple systems, Math. Programming Stud. 2 (1974) 72–81.

[21] M. R. Garey, D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman, New York, NY, 1990.

[22] L. Gargano, J. Körner, U. Vaccaro, Qualitative independence and Sperner problems for directed graphs, J. Comb. Theory Ser. A 61 (1992) 173–192.

[23] L. Gargano, J. Körner, U. Vaccaro, Capacities: From information theory to extremal set theory, J. Comb. Theory Ser. A 68 (1994) 296–316.

[24] P. B. Gibbons, P. R. J. Östergård, Computational Methods in Design Theory, in: C. J. Colbourn, J.H. Dinitz (Eds.), Handbook of Combinatorial Designs, Second Edition (Discrete Mathematics and Its Applications), 2nd ed., Chapman & Hall/CRC, Boca Raton, FL, 2006, pp. 755–782.

[25] B. Goethals, Efficient frequent pattern mining, Ph.D. thesis, Transnationale Universitet Limburg (2002).

[26] S. W. Golomb, L. D. Baumert, Backtrack programming, J. ACM 12 (1965) 516–524.

[27] C. P. Gomes, B. Selman, N. Crato, H. Kautz, Heavy-tailed phenomena in satisfiability and constraint satisfaction problems, J. Automat. Reason. 24 (2000) 67–100.

[28] G. Gottlob, M. Grohe, N. Musliu, M. Samer, F. Scarcello, Hypertree decompositions: Structure, algorithms, and applications, in: D. Kratsch (ed.), Proc. of 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), LNCS 3787, Springer, Berlin, 2005, pp. 1–15.

[29] J. Han, M. Kamber, Data Mining: Concepts and Techniques, 2nd ed., Morgan Kaufman, 2005.

[30] R. Haralick, G. Elliott, Increasing tree search efficiency for constraint satisfaction problems, Artif. Intell. 14 (1980) 263–313.

[31] O. Hudry, On the difficulty of computing the winners of a tournament, in: Proc. of the Workshop on Voting Theory and Preference Modelling, DIMACS, Annales du LAMSADE 6, 2006, pp. 181–191.

[32] P. Jégou, C. Terrioux, Hybrid backtracking bounded by tree-decomposition of constraint networks, Artif. Intell. 146 (2003) 43–75.

[33] D. S. Johnson, A theoretician's guide to the experimental analysis of algorithms, in: M. Goldwasser, D. S. Johnson, C. C. McGeoch (eds.), Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, American Mathematical Society, Providence, RI, 2002, pp. 215–250.

[34] P. Kaski, P. R. J. Östergård, Classification Algorithms for Codes and Designs, Springer, Berlin, 2006.

[35] E. R. Korf, Linear-space best-first search, Artif. Intell. 62 (1993) 41–78.

[36] R. E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artif. Intell. 27 (1985) 97–109.

[37] J. Körner, C. Pilotto, G. Simonyi, Local chromatic number and sperner capacity, J. Combin. Theory Ser. B 95 (2005) 101–117.

[38] A. Koster, Frequency assignment – Models and algorithms, Ph.D. thesis, University of Maastricht (1999).

[39] D. L. Kreher, D. R. Stinson, Combinatorial Algorithms, Generation, Enumeration, and Search, CRC Press, Boca Raton, FL, 1999.

[40] A. Kunzmann, H.-J. Wunderlich, An analytical approach to the partial scan problem, J. Electron. Test. 1 (1990) 163–174.

[41] J.-F. Laslier, Tournament Solutions and Majority Voting, Springer, Berlin, 1997.

[42] C. E. Leiserson, J. B. Saxe, Retiming synchronous circuitry, Algorithmica 6 (1991) 5–35.

[43] E. L. Lloyd, M. L. Soffa, C.-C. Wang, On locating minimum feedback vertex sets, J. Comput. Syst. Sci. 37 (1988) 292–311.

[44] C. Mannino, A. Sassano, Solving hard set covering problems, Oper. Res. Lett. 18 (1995) 1–5.

[45] C. C. McGeoch, Experimental algorithmics, Commun. ACM 50 (11) (2007) 27–31.

[46] P. Meseguer, M. Sánchez, Specializing russian doll search, in: Proc. of 7th International Conference on Principles and Practice of Constraint Programming (CP '01), LNCS 2239, Springer-Verlag, Berlin, Germany, 2001, pp. 464–478.

[47] P. Meseguer, M. Sánchez, G. Verfaillie, Opportunistic specialization in russian doll search, in: P. V. Hentenryck (ed.), Proc. of 8th International Conference on Principles and Practice of Constraint Programming (CP 2002), LNCS 2470, Springer, Berlin, Germany, 2002, pp. 264–279.

[48] A. Mosig, T. Bıyıkoğlu, S. J. Prohaska, P. F. Stadler, Discovering cis-regulatory modules by optimizing barbecues, Discrete Appl. Math. In Press.

[49] J. Nayak, K. Rose, Graph capacities and zero-error transmission over compound channels, IEEE Trans. Inform. Theory 51 (2005) 4374–4378.

[50] M. A. Odijk, H. van Maaren, Improved solutions to the Steiner triple covering problem, Inform. Process. Lett. 65 (1998) 67–69.

[51] T. Orenstein, Z. Kohavi, I. Pomeranz, An optimal algorithm for cycle breaking in directed graphs, J. Electron. Test. 7 (1995) 71–81.

[52] P. R. J. Östergård, A fast algorithm for the maximum clique problem, Discrete Appl. Math 120 (2002) 197–207.

[53] J. Ostrowski, J. Linderoth, F. Rossi, S. Smriglio, Constraint orbital branching, in: A. Lodi, A. Panconesi, G. Rinaldi (eds.), Proc. of 13th Conference on Integer Programming and Combinatorial Optimization (IPCO 2008), LNCS 5035, Springer, Berlin, Germany, 2008, pp. 225–239.

[54] C. H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1995.

[55] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Boston, MA, 1984.

[56] A. Sali, G. Simonyi, Orientations of self-complementary graphs and the relation of sperner and shannon capacities, European J. Combin. 20 (1999) 93–99.

[57] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: C. Mellish (ed.), Proc. of the International Joint Conference on Artificial Intelligence (IJCAI'95), Morgan Kaufmann, San Francisco, CA, 1995, pp. 631–637.

[58] C. Shannon, The zero error capacity of a noisy channel, Information Theory, IRE Trans. Inform. Theory 2 (1956) 8–19.

[59] G. W. Smith, R. B. Walford, The identification of a minimal feedback vertex set of a directed graph, IEEE Trans. Circuits and Systems 22 (1975) 9–15.

[60] E. Speckenmeyer, On feedback problems in digraphs, in: Proc. of 15th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '89), LNCS 411, Springer, Berlin, Germany, 1989, pp. 218–231.

[61] C. Terrioux, P. Jégou, Bounded backtracking for the valued constraint satisfaction problems, in: Proc. of 9th International Conference on Principles and Practice of Constraint Programming (CP-03), 2003, pp. 709–723.

[62] G. Verfaillie, S. Givry, Algorithmic problems and solutions in the valued constraint satisfaction problem framework, in: Proc. of 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT-97), 1997, pp. 947–952.

[63] G. Verfaillie, M. Lematre, T. Schiex, Russian doll search for solving constraint optimization problems, in: Proc. of 13th National Conference on Artificial Intelligence (AIII-96), AIII Press, Menlo Park, CA, 1996, pp. 181–187.

[64] D. Wedelin, An algorithm for large scale 0-1 integer programming with application to airline crew scheduling, Ann. Oper. Res. 57 (1995) 283–301.

[65] D. B. West, Introduction to Graph Theory, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2001.

[66] G. J. Woeginger, Banks winners in tournaments are difficult to recognize, Social Choice and Welfare 20 (2003) 523–528.