TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

M A S T E R A R B E I T

A Lagrangian Decomposition Approach Combined with Metaheuristics for the Knapsack Constrained Maximum Spanning Tree Problem

ausgeführt am

Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter der Anleitung von

Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
und
Univ.-Ass. Dipl.-Ing. Dr.techn. Jakob Puchinger

durch

Sandro Pirkwieser, Bakk.techn.
Matrikelnummer 0116200
Simmeringer Hauptstraße 50/30, A-1110 Wien

Datum

Unterschrift

Abstract

This master's thesis deals with solving the *Knapsack Constrained Maximum Spanning Tree* (KCMST) problem, which is a so far less addressed $\mathcal{NP}$-hard combinatorial optimization problem belonging to the area of network design. Thereby sought is a spanning tree whose profit is maximal, but at the same time its total weight must not exceed a specified value.

For this purpose a Lagrangian decomposition approach, which is a special variant of Lagrangian relaxation, is applied to derive upper bounds. In the course of the application the problem is split up in two subproblems, which are likewise to be maximized but easier to solve on its own. The subgradient optimization method as well as the Volume Algorithm are deployed to solve the Lagrangian dual problem. To derive according lower bounds, i.e. feasible solutions, a simple Lagrangian heuristic is applied which is strengthened by a problem specific local search. Furthermore an Evolutionary Algorithm is presented which uses a suitable encoding for the solutions and appropriate operators, whereas the latter are able to utilize heuristics based on defined edge-profits. It is shown that simple edge-profits, derived straightforward from the data given by an instance, are of no benefit. Afterwards the Lagrangian and the Evolutionary algorithm are combined to form a hybrid algorithm, within both algorithms exchange as much information as possible. Thereby the two algorithms can be combined either in sequential or intertwined order, whereas this choice depends on the desired requirements.

Extensive tests were made on specially created instances of different types. The Lagrangian algorithm turns out to derive the optimal upper bound for all instances except for a few, and to yield, particularly in conjunction with local search, very good and for the most part optimal feasible solutions. Least optimal solutions are derived for maximal plane graphs, where a further improvement is expected by applying the hybrid algorithm. The Lagrangian algorithm is generally superior to previously applied solution methods, especially when considering its run-time.
The Hybrid Lagrangian Evolutionary Algorithm can nevertheless partly improve these results. Whereas the sequential variant actually finds more provably optimal solutions for large maximal plane graphs with up to 12000 nodes, the intertwined variant allows to obtain high-quality solutions usually earlier during a run, in particular on complete graphs.

Zusammenfassung

Diese Masterarbeit befasst sich mit der Lösung des rucksackbeschränkten maximalen Spannbaum-Problems (*Knapsack Constrained Maximum Spanning Tree* (KCMST) problem), einem bisher wenig behandelten $\mathcal{NP}$-schwierigen kombinatorischen Optimierungsproblem das dem Bereich des Netzwerkdesigns zuzuordnen ist. Dabei ist ein Spannbaum gesucht, dessen Profit maximal ist aber zugleich sein Gesamtgewicht einen gewissen Wert nicht überschreitet.

Es wird ein Lagrangescher Dekompositionsansatz präsentiert, welcher eine spezielle Variante der Lagrangeschen Relaxierung ist, um obere Schranken zu ermitteln. Im Zuge des Verfahrens wird das Problem in zwei ebenfalls zu maximierende Teilprobleme aufgespalten, die jedoch für sich gesehen leichter zu lösen sind. Zur Lösung des Lagrangeschen dualen Problems werden sowohl das Subgradientenverfahren als auch der Volume-Algorithmus herangezogen. Um auch entsprechende untere Schranken, also gültige Lösungen zu erhalten, wird eine einfache Lagrangesche Heuristik verwendet, die durch eine problemspezifische lokale Suche gestärkt wird. Des Weiteren wird ein Evolutionärer Algorithmus vorgestellt, der eine geeignete Kodierung der Lösungen und entsprechende Operatoren verwendet, wobei letztere in der Lage sind Heuristiken basierend auf definierten Kantenprofiten einzusetzen. Es wird gezeigt, dass einfache Kantenprofite, die direkt anhand der gegebenen Probleminstanz ermittelt werden, nicht von Vorteil sind. Danach werden der Lagrangesche und der Evolutionäre Algorithmus zu einem hybriden Algorithmus vereint, innerhalb dessen so viel Information wie möglich zwischen beiden ausgetauscht wird. Dabei können die zwei Algorithmen entweder in sequenzieller oder verschachtelter Reihenfolge kombiniert werden, wobei diese Wahl von den gewünschten Anforderungen abhängt.

Ausführliche Tests auf eigens generierten unterschiedlichen Instanzen ergeben, dass der Lagrangesche Algorithmus bis auf wenige Ausnahmen die optimale obere Schranke findet und vor allem zusammen mit der lokalen Suche auch sehr gute und meist sogar optimale gültige Lösungen liefert. Am wenigsten optimale Lösungen erhält man für maximal planare Graphen, wo durch die Anwendung des hybriden Algorithmus eine Verbesserung erwartet wird. Der Lagrangesche Algorithmus ist im Allgemeinen bisherigen Verfahren überlegen, besonders wenn man die Laufzeit in Betracht zieht.
Der Hybride Lagrangesche Evolutionäre Algorithmus kann dennoch diese Ergebnisse teilweise verbessern. Während die sequenzielle Variante tatsächlich mehr beweisbar optimale Lösungen für große maximal planare Graphen mit bis zu 12000 Knoten findet, ermöglicht die verschachtelte Variante, dass bei einem Lauf üblicherweise früher Lösungen mit höherer Güte erlangt werden, besonders für vollständige Graphen.

Danksagungen

Ich möchte meinen beiden Betreuern Prof. Günther Raidl und Dr. Jakob Puchinger dafür danken, dass sie mich, wie auch schon in einem Praktikum davor, für dieses Thema begeistert haben, mir immer die nötige Unterstützung gegeben haben und mich vor allem in die faszinierende Welt der kombinatorischen Optimierung eingeführt haben.

Ein ganz besonderer Dank gebührt meinen Eltern und Großeltern. Sie haben mich immer in meinem Studium unterstützt und ohne sie wäre nicht ein so sorgloses Weiterkommen möglich gewesen.

Nicht zuletzt will ich mich ganz herzlich bei Marlene für ihr Verständnis, ihre Unterstützung und ihre Liebe bedanken.

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

1.1 Problem Description

The problem this thesis deals with is the *Knapsack Constrained Maximum Spanning Tree* (KCMST) problem. It was first formulated in Yamamato and Kubo [43]. In [42] Yamada et al. gave a proof for its $\mathcal{NP}$-hardness (see Chapter 3), as well as algorithms and computational experiments. The following is given (see [42]): an undirected connected graph $G = (V, E)$, where V is a finite set of vertices and $E \subseteq V \times V$ is the set of edges. Each edge is associated with a weight $w : E \to \mathbb{Z}_+$ and a profit $p : E \to \mathbb{Z}_+$. In addition to that a knapsack with integer capacity $c > 0$ is given. The objective is to find a spanning tree $T \subseteq E$ on G, i.e. a cycle-free subgraph on G connecting all nodes V, whose weight $w(T) = \sum_{e \in T} w(e)$ does not exceed c and whose profit $p(T) = \sum_{e \in T} p(e)$ is maximal. A spanning tree is said to be *feasible* if it satisfies $w(T) \leq c$.
Formally the KCMST problem can be written as:

$$\begin{aligned} \text{maximize} \quad & p(T) && (1.1)\\ \text{subject to} \quad & w(T) \leq c, && (1.2)\\ & T \text{ is a spanning tree.} && (1.3) \end{aligned}$$

As one can see, this is a combination of the *maximum spanning tree problem* ((1.1) and (1.3)) and the *knapsack problem* ((1.1) and (1.2)), hence the resulting name. An exemplary instance and its solution are presented in Figure 1.1.
By using negative profits the problem would be equivalent to the knapsack constrained minimum spanning tree problem.

1.2 Applied Solution Methods

In this work the KCMST problem will be tackled by applying a Lagrangian decomposition with a simple Lagrangian heuristic as well as a problem specific local search to strengthen the heuristic. To solve the Lagrangian dual problem the Subgradient Optimization method and the Volume Algorithm are used. The subproblems resulting from decomposition are solved by adequate efficient algorithms. We will further investigate if a Hybrid Lagrangian Evolutionary Algorithm, obtained by appropriately combining an Evolutionary Algorithm tailored to this problem with the Lagrangian method, is of benefit.

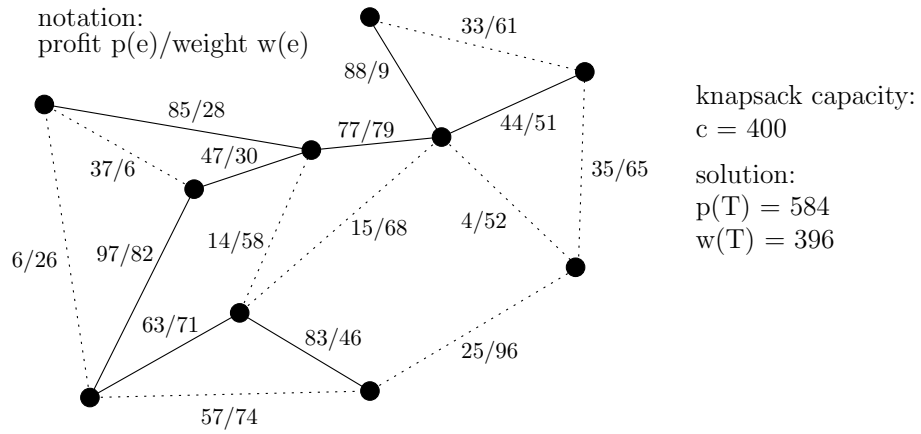


Figure 1.1: Exemplary small KCMST instance and its solution.

1.3 Motivation

This problem arises in the area of communication networks, though it can be of relevance for all kinds of networks. Assume that the goal is to connect all nodes of the network, whereas the before mentioned weight of an edge is related to the cost of construction (e.g. money or time) and the profit is related to whatever benefit the connection between two specific nodes is. The additional knapsack constraint can be a time interval or the budget for the project.

1.4 Thesis Overview

Chapter 2 introduces the necessary basics (terms, definitions and methods) which are used throughout the work. In Chapter 3 previous work relevant for the thesis will be mentioned. Chapter 4 deals with the developed algorithms based on Lagrangian decomposition for solving the KCMST problem. The experimental results of this algorithms and the test instances used are presented in Chapter 5. Thereby comparing our results with previous ones as well as presenting new results. In Chapter 6 an Evolutionary Algorithm with a suitable solution representation is developed for the KCMST problem and tested. Further possible ways to combine the Lagrangian algorithms with the Evolutionary Algorithm into a Hybrid Lagrangian EA will be described, followed by their experimental results. Details of the implementation are described in Chapter 7. Finally the conclusions on the work are drawn in Chapter 8, together with suggestions for further work.

2 Preliminaries

In this chapter we will describe the necessary preliminaries.

2.1 Combinatorial Optimization Problems

Combinatorial Optimization Problems (COPs) are a special class of optimization problems. They occur in a great variety of applications, e.g. in cutting and packing, routing, scheduling, timetabling, network design and many others, and have a high degree of practical as well as academic relevance.

When solving a COP one seeks to find a solution generally being an integer number, a subset, a permutation or a graph structure of a given basic set (either finite or countably infinite), satisfying specific constraints and having associated the maximum or minimum objective value of all feasible solutions. A more formal definition of a COP, partly based on [5]:

Definition 1. A *Combinatorial Optimization* problem $P = (\mathcal{S}, f)$ can be defined by:

- a vector of variables $x = (x_1, \ldots, x_n)$
- variable domains $D = (D_1, \ldots, D_n)$
- constraints among variables
- an *objective function* f to be minimized or maximized (depending on the problem), where $f : D_1 \times \ldots \times D_n \to \mathbb{R}^+$

The set of all possible feasible assignments is

$$\mathcal{S} = \{x = (x_1, \ldots, x_n) \in D \mid x \text{ satisfies all the constraints}\}$$

The set $\mathcal{S}$ is also called a *search* or *solution space*. Every $s \in \mathcal{S}$ is assigned an *objective value* $f(s)$.
As mentioned above, to solve a COP means finding the best solution, defined as the *globally optimal solution* or *global optimum*.

Definition 2. A solution $s^* \in \mathcal{S}$ is said to be *globally optimal* if, assuming a maximization problem, $\forall s \in \mathcal{S} : f(s^*) \geq f(s)$.

Two classes of algorithms to solve COPs are described in the next sections.

2.2 Exact Algorithms

We will first mention the class of *complete* or *exact algorithms*. They are guaranteed to find the optimal solution for a finite size instance in bounded time and prove its optimality. Before dealing with the implication of this statement some more definitions need to be introduced.

Definition 3. The *time complexity function* for an algorithm expresses its time requirement by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size.

Definition 4. *Big Oh.* A function $f(n)$ is $\mathcal{O}(g(n))$ whenever there exists a constant c such that $|f(n)| \leq c \cdot |g(n)|$ for all values of $n \geq 0$. Thus $c \cdot g(n)$ is an upper bound on $f(n)$.

Definition 5. A *polynomial time algorithm* is an algorithm whose time complexity function is $\mathcal{O}(p(n))$, where p is some polynomial function and n is the size of the instance (or its *input length*). If k is the largest exponent of such a polynomial in n, the corresponding problem is said to be solvable in $\mathcal{O}(n^k)$.

Definition 6. If an algorithm's time complexity function can not be bounded by a polynomial in n, it is called an *exponential time algorithm*.

According to the theory of $\mathcal{NP}$-completeness [17] COPs can be divided in those which are known to be solvable in polynomial time (we will present one in Section 2.6) and those that are not (see one in Section 2.7). The former are said to be *tractable* and the latter *intractable*. Though some important COPs are tractable, the great majority of them are intractable and not considered to be "well-solved".

Hence applying exact algorithms on $\mathcal{NP}$-hard [17] COPs needs, in the worst case, exponential time. Thus only small to moderately sized instances can be solved in reasonable time. These algorithms additionally do not scale well on bigger instances.

Examples for exact algorithms which are often applied on $\mathcal{NP}$-hard problems are Branch-And-Bound (B&B), cutting-plane approaches, Branch-And-Cut (B&C), Branch-And-Prize (B&P) and Branch-And-Cut-And-Prize (BCP). They generally apply clever enumeration techniques based on Linear Programming (see Section 2.8) to efficiently explore the whole search space to yield a better performance than doing an exhaustive search by enumerating all possible solutions.

2.3 Metaheuristics

This section introduces algorithms that are classified as *heuristic algorithms* or *heuristics*. Contrary to the previously mentioned exact algorithms, which guarantee to find the provably optimal solution, but possibly need exponential time to do so, heuristic algorithms aim at reaching good or even near-optimal solutions in shorter and thus reasonable (i.e. in

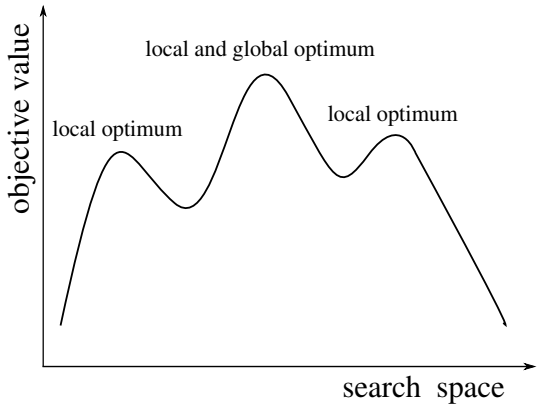


Figure 2.1: Different optima.

polynomial) time, but without optimality guarantee. These algorithms trade optimality for efficiency.

A *metaheuristic*, which was first mentioned in [19], can be seen as a problem independent high-level concept for efficiently exploring the search space by guiding lower-level or subordinate heuristics to find (near-)optimal solutions. It is important that a metaheuristic appropriately balances *diversification* and *intensification* of the search.

One of many classifications can be made between *single point search-based* and *population-based* metaheuristics. Before dealing with them we define some more necessary terms [5], complementing those in Section 2.1. If not stated otherwise we will only consider maximization problems.

Definition 7. A *neighborhood structure* is a function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$, where $2^{\mathcal{S}}$ is the powerset of $\mathcal{S}$, that assigns to every $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$.
$\mathcal{N}(s)$ is called the neighborhood of s.

A neighboring solution s' of s can in general be reached from s by applying a single move, which is for solutions represented as trees usually an edge exchange.

Definition 8. A *locally optimal solution* or *local optimum* with respect to a neighborhood structure $\mathcal{N}$ is a solution s' such that $\forall s \in \mathcal{N}(s') : f(s') \geq f(s)$.

Following this definition each global optimum is also a local optimum. Local and global optima for a maximization problem are illustrated in Figure 2.1.

A good overview of metaheuristics is given in Blum and Roli [5], Fogel and Michalewicz [12] and Gendreau and Potvin [18].

2.3.1 Basic Local Search

The "simplest" member among the single point search-based metaheuristics is the *Basic Local Search* or *iterative improvement*. In the strict sense this basic search does not really belong to the metaheuristics, it is rather a normal *heuristic*), utilized by metaheuristics.

Algorithm 1: Basic Local Search

Input: starting solution s
Result: solution s after local search
repeat
  select $s' \in \mathcal{N}(s)$;
  if $f(s') > f(s)$ **then**
    $s = s'$;
until *termination condition* ;

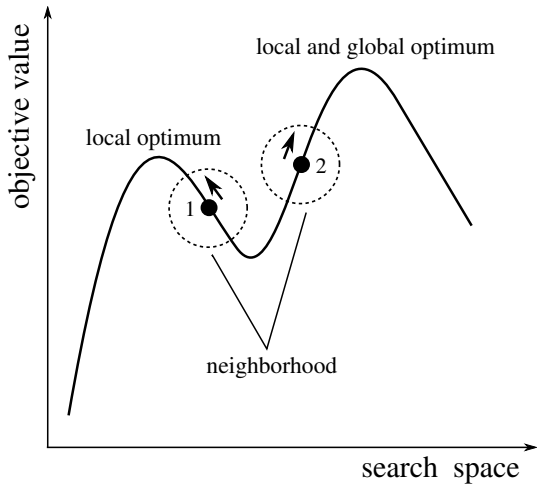


Figure 2.2: Starting solutions and their neighborhood.

The meaning of iterative improvement becomes clear when looking at Algorithm 1. This search only accepts improving solutions. It depends on the neighborhood structure $\mathcal{N}$, how it is processed (select $s' \in \mathcal{N}(s)$) and a given starting solution s, usually created either randomly or by simple construction heuristics. Regarding the processing of $\mathcal{N}(s)$ and accepting a new solution, the most important variants are: (1) *best improvement* scans the whole neighborhood and selects the local optimum, (2) *first improvement* scans the neighborhood in a predefined order and takes the first solution s' with $f(s') > f(s)$ and (3) *random improvement* selects a solution $s' \in \mathcal{N}(s)$ at random and accepts it if $f(s') > f(s)$.

The *termination condition(s)* might be based on a duration limit (either time or overall iterations) or how many iterations have passed without an improvement.

Two starting solutions with their defined neighborhood and the direction of their trajectory can be seen in Figure 2.2. When starting with the first solution (the left one in the figure), chances are high to reach the nearby local optimum, whereas starting solution two (the right one) is heading for the global optimum. Because of this behavior this search is said to do a *hill climbing*.

Basic Local Search is easy to implement and fast, but a major drawback is the inability to overcome local optima. This circumstance is tackled by many more sophisticated single point based-searchs, like Simulated Annealing, Tabu Search, Variable Neighborhood Search, Iterated Local Search and many more [5], which actually belong to the class of metaheuristics.

2.3.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [1] are a class of population-based metaheuristics, and have often been successfully applied to COPs. The idea is to gain such an adaptive behavior like evolutionary processes occurring in nature by modelling them on the computer in a strongly simplified way. Thus the goal of the artificial evolution is to adapt to the environment represented by a specific problem and thereby evolving good solutions.
The general outline of an EA, biased towards a Genetic Algorithm (GA) (a more specific subclass of EAs), is depicted in Algorithm 2, and will be described in the following:

- At the beginning an initial population P is generated, usually with a random constructive heuristic to yield a diversified set of *individuals* or *chromosomes*. The representation of a solution as an individual is called the *genotype*, whereas the solution itself is called the *phenotype*.
- All individuals of P are evaluated,i.e. their *fitness* is determined, corresponding but not necessarily being equal to the objective value of the encoded solution.
- The following steps, comprising one *generation*, are repeated until a termination condition is met, which might be a limit on overall generations or on running time.
  - From the actual population P a set Q_s of individuals is selected as candidates for the next step. The fitness value of an individual reflects the desirability and thus the probability of being chosen, resulting in an *intensification* of the search. The selection process is often implemented by a *tournament selection* of size k, where k individuals are randomly chosen and the fittest of these is taken.
  - In this step the individuals Q_s selected as parents are *recombined* to produce the new individuals (the *offspring*) Q_r. These individuals should be primarily made up of attributes (or genes) of their parents, thus modelling heritability, thereby possibly favoring "good" parts of them (another form of intensification). The concrete implementation highly depends on the genotype.
  - The offspring Q_r has to undergo a mutation operation (corresponding to some extent to natural mutation). With a certain rather small probability each individual is randomly changed to a minor degree. Mutation brings new information into the population and ensures together with recombination a *diversification* of the search. To strengthen an EA one can also use some sort of improvement, mostly local search, instead or in addition of mutation. Such EAs are often

referred to as *Hybrid EAs* or *Memetic Algorithms* [32].
This step results in the final offspring P'.

– After evaluating P' the population of the next generation is selected among the old and new individuals. If the new population consists only of individuals drawn from the offspring it is called *generational replacement*. When using *overlapping populations* then individuals can "survive" a generation, i.e being transferred from the old to the new population. This can be taken even further to so-called *steady-state* or *incremental* strategies, in which usually only one offspring is created and integrated in the population, thereby replacing another chromosome, probably the worst one. An *elitism* strategy is applied if the best solution always survives.
The selection process generally resembles the concept of the "survival of the fittest".

Algorithm 2: Evolutionary Algorithm

```
P ← generate initial population;
Evaluate(P);
repeat
    Q_s ← Selection(P);
    Q_r ← Recombine(Q_s);
    P' ← Mutate(Q_r);
    Evaluate(P');
    P ← SelectSurvivors(P, P');
until termination condition ;
```

Compared to the Basis Local Search an EA has the advantage of conducting a broader search due to the diversification operators and a whole population of solutions. It is thus generally less vulnerable to local optima. By using an adequate intensification strategy it is further able to evolve near-optimal solutions.

To illustrate the applicability and performance of EAs as a representative for metaheuristics and search techniques tailored to a specific problem see Figure 2.3. Not surprisingly the more problem specific knowledge is included in the search method the narrower is its range of application.
Covering this and a lot more about EAs does the book of Eiben and Smith [9].

2.4 Combination of Exact Algorithms and Metaheuristics

A rather new research area is to combine exact algorithms and metaheuristics, thereby using the advantages of both, i.e. obtaining better solutions in shorter time with an additional quality guarantee or occasionally even an optimality-proof.

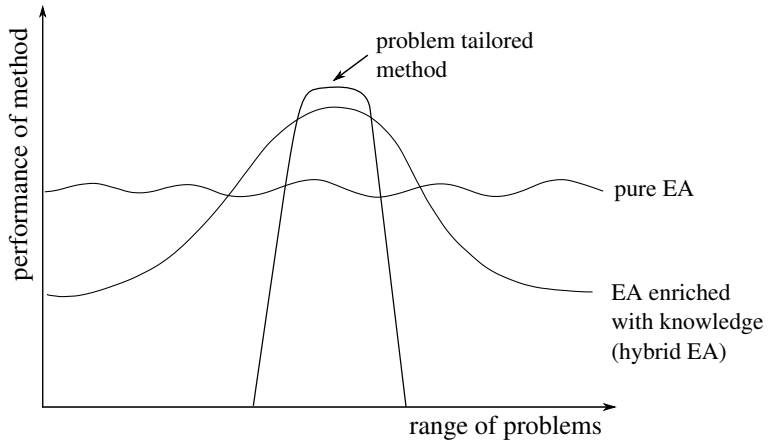


Figure 2.3: Performance of metaheuristics over range of problems.

Puchinger and Raidl [37] gave a general classification scheme for different combinations of metaheuristics and exact algorithms, thus enabling different works to be compared at a high level. The two main categories are (1) *collaborative combination* where the two methods exchange information, but are not part of each other, thereby running (1a) in sequential order or (1b) being executed in a parallel or intertwined way, and (2) *integrative combination* with a distinguished master and at least one integrated slave algorithm, where (2a) exact algorithms are incorporated in metaheuristics or vice versa (2b) metaheuristics are incorporated in exact algorithms. They also presented examples for each combination.

2.5 Graph Theory

A comprehensive introduction and general treatment of this topic is presented in Diestel [7]. We will only present the needed definitions in necessary detail.

Definition 9. A *graph*, or *undirected graph* G consists of a pair of sets $G = (V, E)$, where V is a finite set of vertices or nodes (from thereon the latter term is used), and $E \subseteq V \times V$ is a set of edges. An edge is an unordered pair of distinct nodes.

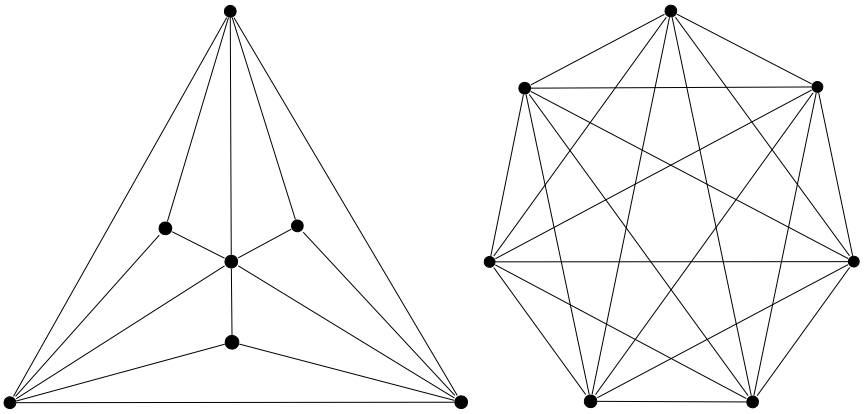
Definition 10. A *plane graph* is one that can be drawn in a plane without crossings of edges.

In the following we will denote a plane graph by $\mathrm{P}_{n,m}$, having n nodes and m edges.

Definition 11. If a plane graph is no longer planar if an additional edge is added (without adding nodes), the graph is said to be a *maximal* or *maximally planar graph*.

Such a graph will be denoted by P_n, having n nodes and for $n \geq 3$ exactly $m = 3 \cdot n - 6$ edges, as an example the graph P_7 is illustrated in Figure 2.4.

Definition 12. A *complete graph* is a graph where every node is connected to all others (except to itself).



(a) P_7 (b) K_7

Figure 2.4: Example for (a) maximal planar and (b) complete graph with $n = 7$.

This graph is denoted by K_n, having n nodes and $m = \frac{n \cdot (n-1)}{2}$ edges, e.g. see Figure 2.4 for the graph K_7.

Definition 13. A *path* is a non-empty graph $P = (V, E)$ consisting of $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\}$, where all x_i are distinct. The nodes x_0 and x_k are linked by P and are called its *ends*, the nodes $x_1, \ldots, x_{k-1}$ are the inner nodes of P. The *length* of a path is its number of edges.

One can also refer to a path by the natural sequence of nodes, therefore resulting in $P = x_0x_1 \ldots x_k$, a path from x_0 to x_k.

Definition 14. Assume $P = x_0 \ldots x_k$ is a path with $k \geq 2$, then the graph $C := P + x_kx_0$ is called a *cycle*, and can again be referred to by its (cyclic) sequence of nodes:
$C = x_0 \ldots x_kx_0$.

Definition 15. A non-empty graph G is called *connected* if any two of its nodes are linked by a path in G, else its called *disconnected*.

Definition 16. A graph G is called a *tree* if any two nodes of G are linked by exactly one path in G. A tree is thus *acyclic*, i.e, it contains no cycles, and connected.

Definition 17. It can be shown that every connected graph $G = (V, E)$ contains a *spanning tree*, which is a subgraph $G_T = (V, T)$ with $T \subseteq E$ whose edges span all nodes, and for which $|V| = |E| + 1$.

2.6 Maximum Spanning Tree Problem

The *Maximum Spanning Tree* (MST) problem, or its complement, the Minimum Spanning Tree problem, arise in many applications as a subproblem (e.g. in the design of communication networks) and are well studied. One either seeks to find a spanning tree whose cost

is minimal or whose profit is maximal. In the following we will only deal with the latter case.
Given is an undirected connected Graph $G = (V, E)$, with a node set V, an edge set $E \subseteq V \times V$, and additional edge profits $p : E \to \mathbb{R}$.
The formulation of the MST problem is:

$$\text{maximize} \quad p(T) = \sum_{e \in T} p(e) \tag{2.1}$$

$$\text{subject to} \quad T \text{ is a spanning tree.} \tag{2.2}$$

The problem is solvable in polynomial time, and two classical greedy algorithms will be presented. The first one is the algorithm of Kruskal [26] (see Algorithm 3). Foremost the edges are sorted according to decreasing profit and a separate tree is created for every node in the graph. Then the edges are consecutively tried to be included in the partial spanning tree, thereby constrained not to induce a cycle. Everytime an edge is included two trees are combined to a single tree which finally results in the maximum spanning tree. Checking if an edge would induce a cycle can be implemented efficiently using a Union-Find data structure, whose operations need constant amortized time, yielding a total running time of $\mathcal{O}(|E| \cdot \log |E|)$, depending only on the sorting of the edges.

The second algorithm is the one of Prim [36] (see Algorithm 4). Contrary to Kruskal's algorithm this one starts with an arbitrary starting node. Then new edges are consecutively added to the partial tree, thereby taking in each step the current most profitable edge connecting a previously unconnected node. This procedure stops when all nodes are connected, yielding the maximum spanning tree. To find the edge with the maximal profit generally a heap is used, for best time complexity either a Fibonacci-Heap [16] or a Pairing-Heap [15]. If this is done, the overall running time is $\mathcal{O}(|V|^2)$.

Since for sparse graphs $|E| \approx \Theta(|V|)$, and for dense graphs $|E| = \Theta(|V|^2)$, Kruskal's algorithm is preferable for the former case, whereas Prim's algorithm usually is the better choice for the latter case.

A good overview is given in Moret and Shapiro [30, 31].

2.7 Knapsack Problem

Another widely known and well-investigated problem is the Knapsack Problem, or KP for short. Given is a set of n items, where every item i has associated a weight $w_i > 0$ and a profit $p_i > 0$, and a capacity $c > 0$. Sought is a subset of all items with a maximal total

Algorithm 3: Kruskal-MST

Input: graph $G = (V, E)$ and $p : E \to \mathbb{R}$
Result: spanning tree T
$S = \emptyset$; $T = \emptyset$; $i = 1$;
sort edges according to decreasing profit: $p(e_1) \geq p(e_2) \geq \ldots \geq p(e_n)$;
forall $v \in V$ **do**
  $S = \{\{v\} \mid v \in V\}$; `// initialize Union-Find data structure`
while $|S| > 1$ **do** `// contains more than one set`
  `//` be $e_i = (u_i, v_i)$ `the next edge`
  if u_i *and* v_i *do not belong to the same set in* S **then**
    $T = T \cup \{(u_i, v_i)\}$;
    unite the sets in S containing u_i and v_i;
  $i = i + 1$;

Algorithm 4: Prim-MST

Input: graph $G = (V, E)$ and $p : E \to \mathbb{R}$
Result: spanning tree T
Select an arbitrary starting node $s \in V$;
$C = \{s\}$;
$T = \emptyset$;
while $|C| \neq |V|$ **do**
  Select an edge $e = (u, v) \in E, u \in C, v \in V \setminus C$ with maximal profit $p(e)$;
  $T = T \cup \{e\}$;
  $C = C \cup \{v\}$;

profit but whose total weight must not exceed the capacity, which is more formally stated as:

$$\text{maximize} \quad \sum_{j \in S} p_j \tag{2.3}$$

$$\text{subject to} \quad \sum_{j \in S} w_j \leq c \tag{2.4}$$

$$S \subseteq \{1, \ldots, n\} \tag{2.5}$$

Trivial cases, like every weight of an item is bigger than c or the sum of all weights is smaller than c, are excluded. There exist many specialized as well as generalized forms of the problem, but we will only deal with the most obvious one, which corresponds to the formulation above. This is the so called *Binary Knapsack Problem* (BKP) or *0/1-KP*, which simply restricts an item to be chosen or not. The solution is presented by a vector x of length n, whereas x_j states if item j is included ($x_j = 1$) or not ($x_j = 0$). Using this representation results in the following formulation:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j \cdot x_j \tag{2.6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j \cdot x_j \leq c \tag{2.7}$$

$$x_j \in \{0, 1\}, \; j = 1, \ldots, n \tag{2.8}$$

Though this problem is generally $\mathcal{NP}$-hard, it is said to be weakly $\mathcal{NP}$-hard because there exists a pseudopolynomial algorithm based on *Dynamic Programming* (DP). Thus even large instances can be efficiently solved.

A comprehensive overview of the different KP variants as well as solution methods is given in Kellerer et al. [25]. An enhanced and currently one of the fastest DP algorithms is presented in Martello et al. [29], which will be used later.

2.8 Linear Programming

In *Linear Programming* (LP) one is given a problem (LP program), formulated as follows:

$$\text{maximize} \quad c^T x \tag{2.9}$$

$$\text{subject to} \quad Ax \leq b \tag{2.10}$$

$$x \in \mathbb{R}^n \tag{2.11}$$

with (2.9) being the linear *objective function* to be maximized, with $c \in \mathbb{R}^n$, and $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ forming the linear *constraints* in (2.10). Integrality constraints can be imposed on a few or even all variables of x, resulting in a *Mixed Integer Program* (MIP) in the former

and in a pure *Integer (Linear) Program* (I(L)P) in the latter case. The important link to COPs is that they can often be formulated as MIPs.

If one restricts the variables in the program to zero or one, this leads to a so called *Binary Linear Program* (BLP) or *0-1 Integer Program* (0-1 IP)

$$\begin{aligned} \text{maximize} \quad & c^T x && (2.12)\\ \text{subject to} \quad & Ax \leq b && (2.13)\\ & x \in \{0,1\}^n && (2.14) \end{aligned}$$

The solution space $\mathcal{S}$ of this 0-1 IP is

$$\mathcal{S} = \{x \in \{0,1\}^n : Ax \leq b\}. \tag{2.15}$$

The *optimal solution value* of a program P will henceforth be denoted as $v(P)$.

Whereas LPs can be solved efficiently using the Simplex or Interior-point method, (M)IPs are in general $\mathcal{NP}$-hard. For a thorough review of LP the reader is referred to Vanderbei [41].

A 0-1 IP will also be the starting point of our work later on in Chapter 4.

2.8.1 Relaxation

Since IPs are in general $\mathcal{NP}$-hard, it is of benefit to deal with a relaxation of the problem at hand, in the following assuming a maximization problem with corresponding program P.

The relaxed program RP of the original program P is a simplification in which some of the constraints are removed. The solution space $\mathcal{S}_R$ of RP is thus larger, i.e. the solution space of the original problem is a subset of the new one, stated as $\mathcal{S} \subseteq \mathcal{S}_R$. In addition to that the objective function f_R of RP dominates (i.e. is better than) those of P over the original solution space, i.e. $\forall x \in \mathcal{S} : f_R(x) \geq f(x)$ and thus also $v(RP) \geq v(P)$. RP is further said to be an *optimistic version* of P.

The main purpose of relaxations is to get upper bounds for the original problem. Although the resulting solution of the relaxed problem might be infeasible for the original problem, it can lend itself to derive a feasible solution (i.e. a lower bound) by applying a problem specific algorithm. In the end there is hopefully a narrow gap between the best lower and upper bound.

One such relaxation is the *Linear Programming Relaxation* (LP Relaxation) or *Continuous Relaxation* (CR), where the integrality constraints of a program P are dropped. Another relaxation will be presented in the next section.

2.9 Lagrangian Relaxation and Decomposition

The following section is based on Beasley [4], Fisher [10, 11], Guignard [20], Frangioni [13] and Lemaréchal [27].

We start with the following IP P:

$$\begin{aligned} \text{maximize} \quad & c^T x && (2.16)\\ \text{subject to} \quad & Ax \leq b && (2.17)\\ & Cx \leq d && (2.18)\\ & x \in \mathbb{Z}^n && (2.19) \end{aligned}$$

with $\{A, B\} \in \mathbb{R}^{m \times n}$, $\{b, d\} \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. Again the objective function (2.16) has to be maximized, thereby considering the constraints (2.17) and (2.18).
Assume that it would be easy to deal with the constraints $Cx \leq d$, but the constraints $Ax \leq b$ are complicating the situation. It would be best to get rid of the latter somehow. This is the time when *Lagrangian Relaxation* (LR) comes into play. We formulate the new relaxed problem LR_λ:

$$\begin{aligned} \text{maximize} \quad & c^T x + \lambda^T (b - Ax) && (2.20)\\ \text{subject to} \quad & Cx \leq d && (2.21)\\ & x \in \mathbb{Z}^n && (2.22) \end{aligned}$$

The slacks of the former complicating constraints have been added to the objective function, thereby replacing the constraints, which are now said to be *dualized*. More precisely the slacks have been added with weights $\lambda_i \geq 0$ for $i = 1, \ldots, m$, which are called Lagrangian multipliers.

It is obvious that $\mathcal{S}_P \subseteq \mathcal{S}_{LR_\lambda}$ holds for every λ, and the objective function of the relaxed program dominates the former one over the original solution space $\mathcal{S}_P$ since the term $\lambda(b - Ax)$ with $\lambda_i \geq 0$ can merely add something to the objective value, thus $v(LR_\lambda) \geq v(P)$ holds, too. So for every positive vector λ the value of $v(LR_\lambda)$ is an upper bound on the optimal value of the original problem P. In order to find the best (i.e. lowest) upper bound we state the problem LR:

$$\min_{\lambda \geq 0} \; v(LR_\lambda) \tag{2.23}$$

This problem is called the *Lagrangian dual* of P. LR is optimized over the dual space of the Lagrangian multipliers, whereas LR_λ was optimized over x.
Ideally it holds that $v(LR) = v(P)$, i.e. the values of both optimal solutions coincide. If this is not the case a so called *duality gap* exists, stating the relative difference between both optimal solutions.

In comparison to the already mentioned LP Relaxation, Lagrangian Relaxation has the advantage of sometimes being able to produce tighter upper bounds, though this is only true if the LR does not have the *Integrality Property*. If the convex hull of the original

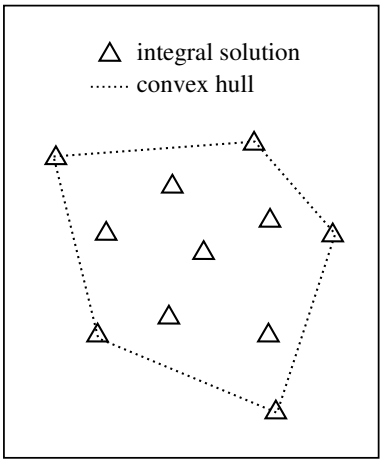


Figure 2.5: The convex hull of a solution set.

solution space regarding the unrelaxed constraints is the same as the LP relaxed solution space, i.e. more formally if $conv\{x \in \mathbb{Z}^n \mid Cx \leq d\} = \{x \in \mathbb{R}^n \mid Cx \leq d\}$, the LR is said to have this property. A simple example of a solution set and the corresponding convex hull is depicted in Figure 2.5.

Denoting the LP Relaxation of program P by LP following can be stated:

$$v(P) \leq v(LR) \leq v(LP).$$

In the worst case the optimal values of the Lagrangian Relaxation and the LP Relaxation are equal.

A special form of Lagrangian Relaxation is the *Lagrangian Decomposition* (LD). It can be applied when there is evidence for two or possibly more intertwined subproblems, which would be easier to solve on their own, because there possibly already exist specialized algorithms.

We will use again the program P stated at the beginning of this section and start with introducing a new variable vector y for the second constraints (now seen as the second problem):

$$\begin{aligned}
\text{maximize} \quad & c^T x && (2.24)\\
\text{subject to} \quad & Ax \leq b && (2.25)\\
& x = y && (2.26)\\
& Cy \leq d && (2.27)\\
& x \in \mathbb{Z}^n && (2.28)\\
& y \in \mathbb{Z}^n && (2.29)
\end{aligned}$$

Since the equality constraints linking x and y (2.26) are introduced too, the program is still equivalent to P. Now these particular constraints can be relaxed in Lagrangian fashion, resulting in program LD_λ:

$$\begin{aligned} \text{maximize} \quad & c^T x - \lambda^T (x - y) && (2.30)\\ \text{subject to} \quad & Ax \leq b && (2.31)\\ & Cy \leq d && (2.32)\\ & x \in \mathbb{Z}^n && (2.33)\\ & y \in \mathbb{Z}^n && (2.34) \end{aligned}$$

A reformulation makes the separation into two independent problems, when considering λ to be constant, explicit:

$$\begin{aligned} \text{(Problem 1)} \quad & \max_x \{(c - \lambda)^T x \mid Ax \leq b, x \in \{0,1\}^n\} + && (2.35)\\ \text{(Problem 2)} \quad & \max_y \{\lambda^T y \mid Cy \leq d, y \in \{0,1\}^n\} && (2.36) \end{aligned}$$

As before we can again state the Lagrangian dual problem LD of LD_λ:

$$\min_\lambda \; v(LD_\lambda) \tag{2.37}$$

Notice that for equality constraints the multipliers need no longer be nonnegative.
Regarding the Integrality Property and the quality of the best upper bound, following can be stated: (1) in case one of the subproblems has the Integrality Property, the best upper bound of LD is equal to the better of the two LR bounds that would result in relaxing either $Ax \leq b$ or $Cx \leq d$, and (2) if both subproblems have the Integrality Property the best upper bound of LD equals the upper bound of the LP relaxation.

Yet missing is a method to solve the Lagrangian dual problem, i.e. to find best suited Lagrangian multipliers. The *Lagrangian function* $z(\lambda) = v(LR_\lambda)$ is a convex function of λ (illustrated in Figure 2.6). Well known and widely applied is the *Subgradient Optimization* originally proposed by Held and Karp [22, 23]. An improved and extended version of this method is the *Volume Algorithm* (VA) [3]. Both will be addressed in Chapter 4.

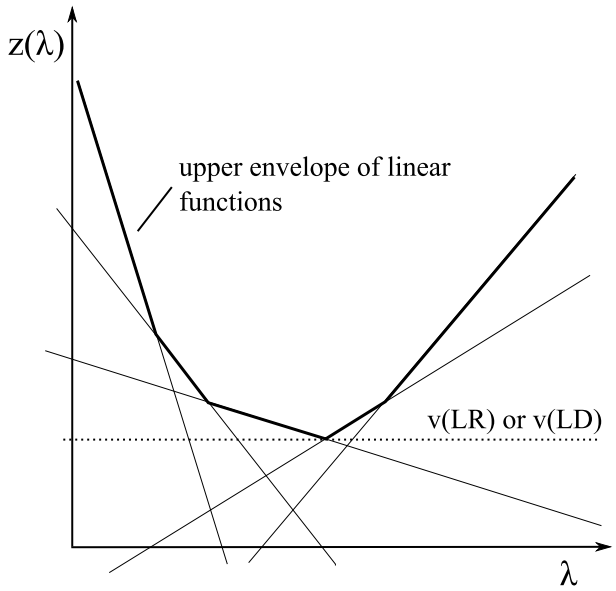


Figure 2.6: Lagrangian function $z(\lambda)$.

3 Previous Work

As already noted in Chapter 1, Yamamato and Kubo [43] were the first to formulate the KCMST problem and they *"...discussed the Lagrangian relaxation, but neither proved $\mathcal{NP}$-hardness nor gave solution algorithms"* [42, page 24] according to Yamada et al. (since the former is only available in Japanese).
The latter authors finally proved $\mathcal{NP}$-hardness by using an appropriately constructed graph (including profits and weights of the edges) and reducing the Knapsack Problem (KP) to the KCMST problem, which completed the proof since KP is itself $\mathcal{NP}$-hard. But it is yet unknown if the problem is either strongly or weakly $\mathcal{NP}$-hard.
They also proposed a Lagrangian relaxation by relaxing the knapsack constraint, leading to:

$$\begin{aligned} &\text{maximize} && L(\lambda, T) = p(T) + \lambda(c - w(T)) && (3.1)\\ &\text{subject to} && T \text{ is a spanning tree.} && (3.2) \end{aligned}$$

This reduces the problem for a given λ to the simple MST problem. They described a bisection method to find the optimal Lagrangian multiplier value (i.e., solving the Lagrangian dual), thus obtaining the best upper bound, and simultaneously derived a lower bound. Further they mentioned a 2-opt local search method (see Algorithm 5) and a

Algorithm 5: Local Search of Yamada et al. [42]

$T \leftarrow$ lower bound from LR;
while $\exists\ T' \in \mathcal{N}(T) : p(T') > p(T)$ **do**
  $T = T'$;

corresponding neighborhood structure. For a spanning tree T the neighborhood $\mathcal{N}(T)$ consists of all feasible trees T' which can be created from T by adding an arbitrary edge not yet included, which induces a cycle and then removing another edge from this cycle. The derived Lagrangian lower bound acts as a starting point.
The authors continue by introducing a decomposition scheme to divide the problem into mutually disjoint subproblems and present a recursive B&B algorithm utilizing this scheme and the bounds created before.
Since the bounds used play a crucial role in B&B algorithms, they propose a shooting method which guesses an artificial lower bound lying between the derived lower and upper bounds. As long as the B&B algorithm fails when using this lower bound, i.e. it terminates without finding the solution to the problem, the guessed value is decreased according to a non-increasing function. This should result in less overall subproblems solved and thus

less computation time.
They conclude their work with numerical experiments, thereby using self-defined instances, both will be addressed later.

Another work dealing with the KCMST problem is reported in Pirkwieser [33]. Therein a B&C algorithm has been devised using an extended local branching framework introduced by Lichtenberger [28]. The algorithm applies three different cuts with increasing complexity and contains a heuristic to derive incumbent solutions. The heuristic itself makes use of a local search that will be used in this work, too. It is described in the next chapter.

Although Haouaria and Siala [21] deal with the Prize Collecting Steiner Tree (PCST) problem the work of this thesis shares some similarities with theirs:

- They are applying a hybrid Lagrangian GA.
- To generate good lower bounds (since it is a minimization problem) they relax the problem using Lagrangian decomposition obtaining a MST and a KP problem.
- The Lagrangian dual is solved with the Volume Algorithm.
- The resulting hybrid GA exploits information from solving the Lagrangian dual.

Apart from the details, among mainly the problem specificity, some aspects of their work are a good point of reference for our work.
Following the scheme presented in Section 2.4 Haouaria and Siala applied a *collaborative combination* with *sequential execution* because both methods, i.e. solving the Lagrangian dual as well as the hybrid GA, are independent of each other and the latter method is started after the former has finished, thus sequential. Since the hybrid GA uses information provided by the solution of the Lagrangian dual it qualifies as a collaborative combination.

4 Lagrangian Decomposition for the KCMST Problem

In this chapter we will apply Lagrangian decomposition to the KCMST problem. Further we will also present two well known methods to solve the Lagrangian dual problem, methods to solve the subproblems resulting from decomposition and finally a simple Lagrangian heuristic to derive incumbent solutions and a local search to improve them.

At first the original KCMST problem, denoted as KCMST-P, is given:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m} p_i x_i && (4.1)\\
\text{s. t.} \quad & x \text{ is a spanning tree} && (4.2)\\
& \sum_{i=1}^{m} w_i x_i \leq c && (4.3)\\
& x \in \{0,1\}^m && (4.4)
\end{aligned}$$

where m is the number of edges and p_i and w_i with $i = 1, \ldots, m$, are the profit and weight values of the edges, respectively. As already mentioned before the problem is a composition of the MST and KP problem, thus we already know the subproblems and it is not hard to spot the starting point for a decomposition. We begin with splitting the variables by introducing new ones for the second constraint (regarding the maximal weight) and equate both variables:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m} p_i x_i && (4.5)\\
\text{s. t.} \quad & x \text{ is a spanning tree} && (4.6)\\
& x = y && (4.7)\\
& \sum_{i=1}^{m} w_i y_i \leq c && (4.8)\\
& x \in \{0,1\}^m && (4.9)\\
& y \in \{0,1\}^m && (4.10)
\end{aligned}$$

It is obvious that this program is still equivalent to KCMST-P.
The next step is to relax equality constraint (4.7) in a Lagrangian fashion (as noted in

Section 2.9) using the Lagrangian multipliers $\lambda \in \mathbb{R}^m$. By doing this we obtain the Lagrangian decomposition of the original problem, which is denoted as KCMST-LD_λ:

$$\max \quad \sum_{i=1}^{m} p_i x_i - \sum_{i=1}^{m} \lambda_i (x_i - y_i) \tag{4.11}$$

$$\text{s. t.} \quad x \text{ is a spanning tree} \tag{4.12}$$

$$\sum_{i=1}^{m} w_i y_i \leq c \tag{4.13}$$

$$x \in \{0,1\}^m \tag{4.14}$$

$$y \in \{0,1\}^m \tag{4.15}$$

Stating KCMST-LD_λ in a more compact way and additionally emphasizing the separated problems yields:

$$\text{(MST)} \quad \max_x \{(p-\lambda)^T x \mid x \text{ is a spanning tree}, x \in \{0,1\}^m\} + \tag{4.16}$$

$$\text{(KP)} \quad \max_y \{\lambda^T y \mid w^T y \leq c, y \in \{0,1\}^m\} \tag{4.17}$$

Due to the properties of Lagrangian relaxation, following can be stated:

$$\forall \lambda \; v(\text{KCMST-}LD_\lambda) \geq v(\text{KCMST-}P) \tag{4.18}$$

Thus, for any λ we obtain an upper bound for the original 0-1 IP.
To find the best (lowest) upper bound we can finally formulate the Lagrangian dual problem:

$$\text{KCMST-}LD = \min_\lambda v(\text{KCMST-}LD_\lambda). \tag{4.19}$$

Now we will consider the strength of the relaxation. Whether the MST problem has the Integrality Property depends on its concrete formulation (we only stated that "x is a spanning tree"), however the KP certainly does not possess it. We therefore expect better upper bounds as if using a simple LP relaxation.

4.1 Solving the Lagrangian Dual Problem

In this section we will describe the application of two well known methods for solving the Lagrangian dual problem to our specific problem KCMST-LD (4.19).

4.1.1 Subgradient Optimization Method

The *Subgradient Optimization* method has been applied to many problems because of its generality. The fact that it is working directly on the constraints makes it simple to adapt to a specific problem. We use the description given in Beasley [4]. The method applied to KCMST-LD is presented in Algorithm 6. In the following we will go through the

Algorithm 6: Subgrad Optimization algorithm

Result: best lower bound z_{LB}, best upper bound z_{UB} and best solution found $bestSol$

```
(sol, p(sol)) ← getInitialSolution(); // see Algorithm 7
bestSol ← sol;
z_LB ← p(sol);
z_UB = ∞;
choose initial values for λ;
initialize T and f accordingly;
t = 0;
subgradSteps = 0;
while z_LB ≠ ⌊z_UB⌋ and subgradSteps ≠ maxSteps do
    t = t + 1;
    (z^t_MST, x^t) ← solveMST(p − λ); // see Section 4.2
    (z^t_KP, y^t) ← solveKP(λ); // see Section 4.3
    z^t = z^t_MST + z^t_KP; // actual upper bound
    LagrangianHeuristic(x^t); // see Section 4.4
    if z^t < z_UB then // better (lower) upper bound found
        if z^t < ⌊z_UB⌋ then
            subgradSteps = 0;
        else
            subgradSteps = subgradSteps + 1;
        z_UB = z^t; // update best upper bound
    else
        subgradSteps = subgradSteps + 1;
    update T and f accordingly;
    v^t = x^t − y^t; // determine actual subgradients
    s = f · (z_UB − T)/||v^t||^2; // determine step size
    λ = λ + sv^t; // update multipliers
```

algorithm and describe relevant parts of it.
The function *getInitialSolution()* (see Algorithm 7) in the first line generates a feasible solution (=spanning tree) which is used as the initial lower bound and to initialize other parameters mentioned later. It uses the function *solveMST* that determines a Maximum Spanning Tree regarding to profit values per edge, defined as input (for further details see Section 4.2). At first the values are calculated from p/w, as a measure of profit per weight, thus favoring edges with high profit and small weight. If this generates a spanning tree whose weight is too mighty, i.e. $w(x) > c$, then a new tree is determined using $1/w$ as profit values, preferring edges with small weight, which finally guarantees that a feasible solution can be found assuming we are given a correct (i.e. solvable) problem instance.

The initial values of λ_i in line 5 are chosen as following:

$$\lambda_i = \frac{3}{4}p_i + \sigma_i, \text{ with } \sigma_i \in \left[0, \frac{p_i}{4}\right] \text{ chosen randomly.} \tag{4.20}$$

Though Beasley [4] stated that the convergence behavior as well as the final result are independent of the initial values of λ, it was observed in preliminary tests (not detailed here) that when using this initialization scheme the method consistently started out with creating smaller upper bounds (=z^t in the algorithm) and thus needing fewer iterations to converge when compared to other schemes tried (e.g. complete random generation).

The termination condition in line 9 is kept simple and intuitive. The first criterion is if $z_{LB} = \lfloor z_{UB} \rfloor$ holds, which means that the provably optimal solution is found. Since we are dealing with integral variables and thus integral objective values of solutions it is valid to use $\lfloor z_{UB} \rfloor$ instead of z_{UB}. The second criterion, $subgradSteps = maxSteps$, assures that the algorithm terminates by restricting the number of consecutive iterations where $z^t \geq \lfloor z_{UB} \rfloor$ holds.

In each iteration the actual upper bound z^t is calculated by solving the two subproblems in line 11 and 12 and adding z^t_{MST} and z^t_{KP}. If this upper bound is better than the best upper bound so far, the latter is updated in line 20. The Lagrangian heuristic applied in line 14 will be described in Section 4.4.
The subgradients v^t are determined in line 24 by calculating the slack $x^t - y^t$. The subgradients determine the direction in which the Lagrangian multipliers λ are altered. The amount of change depends on the *step size* s, but to compute it we have to introduce two more parameters, namely T and f.
The *target value* T acts as substitution for the best lower bound z_{LB} and is initialized with $0.95 \cdot z_{LB}$. Further it is updated every time new information regarding T is derived, more precisely if the problem instance is a plane graph with

$$T = 0.95 \cdot z_{LB} \tag{4.21}$$

or if we deal with a complete (or generally a non-plane) graph using

$$T = 0.475 \cdot (z_{LB} + z_{UB}). \tag{4.22}$$

Additionally if $z_{UB} < 1.05 \cdot T$ holds, i.e. z_{UB} lies within 5% of T, then T is decreased by 5% with the purpose to avoid the convergence getting slower when the two bounds approach each other [4]. This scheme was also developed in preliminary tests and seemed to be satisfying.
The parameter f is initially set to 2, and is multiplied by an assignable parameter $multiply < 1$ (thus is decreased) as soon as a specified amount, given by parameter *steps*, of consecutive iterations without improvement of z_{UB} have passed and $f > 10^{-7}$ holds.
The values of T, f and v^t allow to compute the step size s (line 25) and finally to adapt the Lagrangian multipliers accordingly (line 26).
As can be noticed λ is changed at every iteration, unaffected of the actual improvement. There is further no exploitation of past computations. The method presented next tries to incorporate both.
Results of the Subgradient method are presented in Chapter 5.

Algorithm 7: getInitialSolution()

Input: profit values p and weight values w of all edges
Result: feasible solution x, solution weight $w(x)$ and solution profit $p(x)$
for *i=1 to m* **do** `// m ...number of edges`
  $values_i = p_i / w_i$;
$(x, w(x), p(x)) \leftarrow$ `solveMST(`$values$`)`;
if $w(x) > c$ **then** `// solution` x `is infeasible`
  for *i=1 to m* **do**
    $values_i = 1 / w_i$;
  $(x, w(x), p(x)) \leftarrow$ `solveMST(`$values$`)`;

4.1.2 Volume Algorithm

The *Volume Algorithm* (VA) was introduced in Barahona and Anbil [3]. It is an extension and improvement of the Subgradient Optimization method presented in the previous section and has been applied to many COPs, among them the Steiner Tree Problem [2]. Thus the two algorithms share many similarities, which will be seen in the following description. The VA is depicted in Algorithm 8.
The process of creating an initial solution as well as choosing the initial values of the multipliers is exactly the same as before. The first difference is an iteration 0 (lines 5 and 6) during the initialization phase to generate initial values for z_{UB} (line 7) as well as for both primal vectors x_P and y_P (line 8). These *primal vectors* represent an approximation to a primal solution. The VA determines this values by estimating the volume (hence the

Algorithm 8: Volume Algorithm

Result: best lower bound z_{LB}, best upper bound z_{UB} and best solution found $bestSol$

```
(sol, p(sol)) ← getInitialSolution(); // see Algorithm 7
bestSol ← sol;
z_LB ← p(sol);
choose initial values for λ;
(z^0_MST, x^0) ← solveMST(p − λ); // see Section 4.2
(z^0_KP, y^0) ← solveKP(λ); // see Section 4.3
z_UB = z^0_MST + z^0_KP;
(x_P, y_P) = (x^0, y^0); // initialize primal values
initialize T and f accordingly;
t = 0;
volAlgSteps = 0;
while z_LB ≠ ⌊z_UB⌋ and volAlgSteps ≠ maxSteps do
    t = t + 1;
    v^t = x_P − y_P; // determine actual subgradients
    s = f · (z_UB − T)/||v^t||^2; // determine step size
    λ^t = λ + sv^t; // determine actual multipliers
    (z^t_MST, x^t) ← solveMST(p − λ^t);
    (z^t_KP, y^t) ← solveKP(λ^t);
    z^t = z^t_MST + z^t_KP; // actual upper bound
    LagrangianHeuristic(x^t); // see Section 4.4
    determine actual α;
    (x_P, y_P) = α(x^t, y^t) + (1 − α)(x_P, y_P); // update primal values
    if z^t < z_UB then // better (lower) upper bound found
        if z^t < ⌊z_UB⌋ then
            volAlgSteps = 0;
        else
            volAlgSteps = volAlgSteps + 1;
        z_UB = z^t; // update best upper bound
        λ = λ^t; // update multipliers
    else
        volAlgSteps = volAlgSteps + 1;
    update T and f accordingly;
```

name of the algorithm) below the faces that are active at an optimal dual solution [3], thus they can be regarded as probabilities. As can be seen in line 22 they are a convex combination of the dual solutions produced at each iteration, and depending on the coefficient α. The updating scheme of α is as following, depending itself on the binary parameter *optimalAlpha*:

- If $optimalAlpha = 0$ then the first proposal of [3] is applied, namely setting it to a fixed value for a number of iterations and decreasing it afterwards. So at the beginning we set α to the parameter *alphaStart*. Then after every 100 iterations for plane and every 200 iterations for non-plane graphs (since the latter require more iterations, which will be seen later) it is checked if z_{UB} decreased less than 1%, if so and $\alpha > 10^{-5}$ holds then $\alpha = 0.85 \cdot \alpha$.

- In case $optimalAlpha = 1$ let $v_P = x_P - y_P$, $v^t = x^t - y^t$ and α_{max} be initialized with the value of *alphaStart*. Then $\alpha_{opt} = \min_\alpha ||\alpha v^t + (1-\alpha)v_P||$ is computed using a Golden Section Search [35]. If $\alpha_{opt} < 0$ then $\alpha = \alpha_{max}/10$ otherwise $\alpha = \min\{\alpha_{opt}, \alpha_{max}\}$ is used. Additionally the same decreasing strategy as in case $optimalAlpha = 0$ is used, but with α_{max} altered instead of α. A similar strategy was proposed in [2] and was also used by [21].

Though the VA would make better defined stopping criteria possible [3], we will use the same as before since it proved to work well in preliminary tests.

Another difference is that the actual subgradients are based on the slack of the primal values $x_P - y_P$ (line 14), thus allowing a finer grained subgradient since the primal values are not necessarily integral. Whereas the formula for determining the step size s (in line 15) is the same as well as the updating scheme of the target value T (but since we have an upper bound derived from iteration 0 the update step differentiating between the two graph types can already be used at initialization), we will use another one for parameter f, therefore defining three types of iterations [3]:

- If no improvement was made, this iteration is called *red*.

- If $z^t < z_{UB}$ then $d = v^t(x^t - y^t)$ is computed.
  - If $d < 0$ holds, then a longer step in the direction v^t would have resulted in a smaller value of z^t. This iteration is called *yellow*.
  - If $d \geq 0$ the iteration is called *green* and indicates the need for a larger step size.

The initial value of f is 0.1. If enough consecutive red iterations occurred, determined by the parameter *steps*, and $f > 10^{-8}$ in case of plane and $f > 10^{-6}$ in case of complete graphs (these values were determined in preliminary tests by looking at the value of f occurring at optimal solutions), f is multiplied by the parameter $multiply < 1$. After every green iteration and if $f < 1$ we multiply f by 1.1. A yellow iteration has no effect.

As can be seen, though the actual Lagrangian multipliers λ^t are calculated at every iteration (line 16), the quasi prime Lagrangian multipliers λ are only updated if the former one led to a better upper bound (line 29).

Summarizing, the main differences compared to the Subgradient method are: (1) the computation of primal values and (2) using this convex combination of preceding dual values for the determination of the actual subgradients, (3) an advanced calculation of the step size by using an extended update scheme for the coefficient f and (4) to take only multipliers that led to an improvement.

4.2 Problem 1 - MST

The first of the two subproblems in the Lagrangian decomposition approach is the MST problem already mentioned in Section 2.6. The MST algorithm is given the edge profits $p - \lambda$, i.e. the original profits p are modified by subtracting the Lagrangian multipliers λ. The following MST algorithms have been implemented and tested (parameter $mstAlg$):

0: Kruskal's algorithm (Algorithm 3) with complete presorting of the edges, thus designed for rather sparse graphs and called *Kruskal-P*.

1: Kruskal's algorithm with sorting on demand realized using Heap Sort, designed for dense graphs and called *Kruskal-C*.

2: Prim's algorithm (Algorithm 4) using a Pairing-Heap [15] with dynamic insertion, called *Prim-p-heap*.

3: Prim's algorithm using a Fibonacci Heap [16] with dynamic insertion, called *Prim-f-heap*.

4: Kruskal's algorithm using a Pairing-Heap as a form of on demand sorting, called *KruskalPQ-p-heap*.

5: Kruskal's algorithm using a Fibonacci-Heap as a form of on demand sorting, called *KruskalPQ-f-heap*.

Every instance of Kruskal's algorithm was implemented using a Union-Find data structure. All of these versions of MST algorithms have been mentioned in literature (see [30] for a compact overview).
Since we set a high value on efficiency of the overall algorithm we tried all these algorithms to find the most suitable. The results are presented in Chapter 5.

4.3 Problem 2 - KP

The second subproblem is the Knapsack Problem, first mentioned in Section 2.7. More precisely we deal with the 0/1-KP since each item of the KP corresponding to an edge of

the graph is either selected or not.
We are given the 0/1-KP (we use the same notation as the Lagrangian decomposition):

$$\text{maximize} \quad \sum_{j=1}^{m} \lambda_j \cdot y_j \tag{4.23}$$

$$\text{subject to} \quad \sum_{j=1}^{m} w_j \cdot y_j \leq c \tag{4.24}$$

$$y_j \in \{0,1\},\ j = 1,\ldots,m \tag{4.25}$$

The profit value per item is the actual value of the corresponding Lagrangian multiplier λ_j, whereas the weight w_j is the weight of edge j defined by the problem instance.
To solve this problem we will use the COMBO algorithm described in Martello et al. [29] (see Chapter 7 for further details), a successor of the algorithm presented in [34]. The COMBO algorithm is a sophisticated Dynamic Programming algorithm which was also used in [21].

As suggested in [4] one can include constraints in the Lagrangian relaxed program LR which would have been redundant and thus unnecessary in the original ILP P, but which strengthen the bound attainable from LR. We can apply this idea to the KP subproblem: whereas the complete KCMST problem as well as the MST subproblem restrict their solutions to being a spanning tree, thus having exactly $n-1$ edges selected (with n being the number of nodes), the KP as stated above does not have this property. If we constrain the number of items to an exact sum we obtain the *exact k-item knapsack problem* (E-kKP) [6] which is a special case of the *cardinality constrained knapsack problem*. The resulting E-kKP can be stated as:

$$\text{maximize} \quad \sum_{j=1}^{m} \lambda_j \cdot y_j \tag{4.26}$$

$$\text{subject to} \quad \sum_{j=1}^{m} w_j \cdot y_j \leq c \tag{4.27}$$

$$\sum_{j=1}^{m} y_j = n - 1 \tag{4.28}$$

$$y_j \in \{0,1\},\ j = 1,\ldots,m \tag{4.29}$$

Caprara et al. [6] presented a pseudopolynomial dynamic programming scheme for solving the E-kKP in $\mathcal{O}(nkc)$ time and $\mathcal{O}(k^2c)$ space (when using DP by weights), where n is the number of items, k is the number of items to be chosen and c is the knapsack constraint. When determining the time and space requirements depending on the number of nodes n for the KCMST problem we have to consider that the items are the edges of the graph, thus being in the worst case $m \approx \mathcal{O}(n^2)$ for complete graphs and k and c both depend linearly on n. This yields a running time of $\mathcal{O}(n^4)$ and a space requirement of $\mathcal{O}(n^3)$. Since

these characteristics are not very satisfying and a proper implementation would probably be a thesis on its own, we will use an ILP solver instead. For a comparison of KP versus E-kKP see Section 5.4.2.

4.4 Lagrangian Heuristic and Local Search

A Lagrangian heuristic [4] is responsible for generating feasible solutions to derive better lower bounds. It therefore uses the solution to the Lagrangian dual to generate primal solutions. Ideally the method is simple enough, i.e. computationally not very demanding, to be applied each iteration. Furthermore a local search (probably problem specific) can be used to derive even better bounds. Since it is consecutively applied to different solutions it acts like a multistart local search.
We will use the heuristic illustrated in Algorithm 9.
It receives the solution to the MST subproblem and checks if it is feasible. If not, the

Algorithm 9: Lagrangian heuristic

```
Input: actual solution x of the MST subproblem
if w(x) ≤ c then // solution x is feasible
    if z_LB/z_UB > klsMinGap then // gap between z_LB and z_LB is tight
    enough
        x' ← LocalSearch(x); // see Algorithm 10
        if f(x') > f(bestSol) then
            bestSol = x';
            z_LB = p(x');
    else if f(x) > f(bestSol) then
        bestSol = x;
        z_LB = p(x);
```

heuristic does nothing, since we only deal with yet feasible solutions. It has been tried to randomly repair the solutions and run the local search mentioned below on it, but this did not result in a noticeable improvement, so we will skip this option. However if the solution is already feasible the ratio of the lower and upper bound is calculated, which serves as a measure for the remaining gap, and compared with parameter $klsMinGap$ (with $0 \leq klsMinGap \leq 1$). If the quasi gap lies below or is equal to the parameter value the current best solution $bestSol$ as well as the lower bound z_{LB} are updated if the yet feasible solution x is better, i.e., if $f(x) > f(bestSol)$.
Otherwise if the ratio is higher than $klsMinGap$ a local search–henceforth denoted by KLS (from KCMST Local Search)–is applied to x and produces another solution x' being at least as good as x, thus $f(x') \geq f(x)$ is assured. This resulting solution then updates $bestSol$ and z_{LB} if required.
The introduction of the parameter $klsMinGap$ prevents the KLS from being applied too

early in the process where good solutions are neither required nor easy to obtain. The idea is to save the local search until the Lagrangian multipliers are supposed to be of good quality [13], thus intensifying the heuristic search at the end of the solution process. The local search KLS is presented in Algorithm 10. The general idea is the same as in [42], to exchange an edge with another one and thereby retaining a spanning tree. The procedure is to include an edge not present in the solution, thus inducing a cycle in the tree, and then to remove an edge from that cycle, see Figure 4.1. One has to decide about which edge is tried to be included and given that edge, which one to remove from the introduced cycle. The latter is clear after stating when an intermediate tree T', resulting from one exchange step on the basis of the tree T, is better:

- if T' has a higher profit value as T, i.e. when $p(T') > p(T)$ or
- if T' has the same profit assigned, but weighs less than T, i.e $p(T') = p(T)$ and $w(T') < w(T)$.

The first proposition is obvious. The second one considers the knapsack character of the problem by preferring solutions with less total weight, which are subsequently easier to improve. This is implemented in the algorithm by searching for an edge to be deleted in the induced cycle having either less profit or the same profit but more weight as the edge to be included (line 9). Since the maximal weight c given by the problem instance may not be exceeded after exchange, the removed edge must satisfy a minimum weight constraint, denoted by w_{min} in the algorithm (calculated in line 4).

What remains to be defined is the method of selecting the edge to be included, denoted by the function *getNextEdge()* in line 2. This is an important part of KLS, since it is responsible for guiding the search. We used three such selection processes, which are determined by the parameter $klsType$:

0: Select an edge not active in the current solution at random.

1: Like above plus excluding all edges that were selected and active so far, i.e. select edges at most once and do not reconsider already removed edges.

2: At the beginning of KLS the edges are sorted according to descending order defined by their current modified profit values $p'_i = p_i - \lambda_i$ (being equal the values used to solve the MST subproblem of that iteration) resulting in $p'(e_1) \geq p'(e_2) \geq \ldots \geq p'(e_m)$. Then in every retry of KLS the next less profitable edge not active in the current solution is selected. This results in a greedy search where every edge is considered at most once.

In case of $klsType = 2$ the applied method to sort the edges depends on the graph type of the instance, being once again complete presorting for plane graphs and sorting on demand using Heap Sort for non-plane graphs. As indicated in preliminary tests using sorting on demand instead of complete presorting for non-plane graphs resulted in a remarkable speed improvement.

Algorithm 10: Local Search (KLS)

Input: feasible solution x represented as tree T, graph $G = (V, E)$
Result: probably improved solution x'

while $retries < klsRetries$ **do**
  $e_{new} \leftarrow$ `getNextEdge()`;
  $w_{max} = w(e_{new})$;
  $w_{min} = w(T) + w(e_{new}) - c$;
  $p_{min} = p(e_{new})$;
  $e_{remove} = e_{new}$;
  determine cycle $C \in T \cup \{e_{new}\}$; `// there is exactly one cycle in` $T \cup \{e_{new}\}$
  foreach $e \in C$ **do**
    if $(w(e) \geq w_{min}) \wedge (p(e) < p_{min} \vee (p(e) = p_{min} \wedge w(e) > w_{max}))$ **then**
      $w_{max} = w(e)$;
      $p_{min} = p(e)$;
      $e_{remove} = e$; `// save worse edge`
  if $e_{remove} \neq e_{new}$ **then** `// replaceable edge found`
    $T = T \setminus \{e_{remove}\} \cup \{e_{new}\}$;
    $retries = 0$;
  else
    $retries = retries + 1$;
$x' \leftarrow T$;
return x';

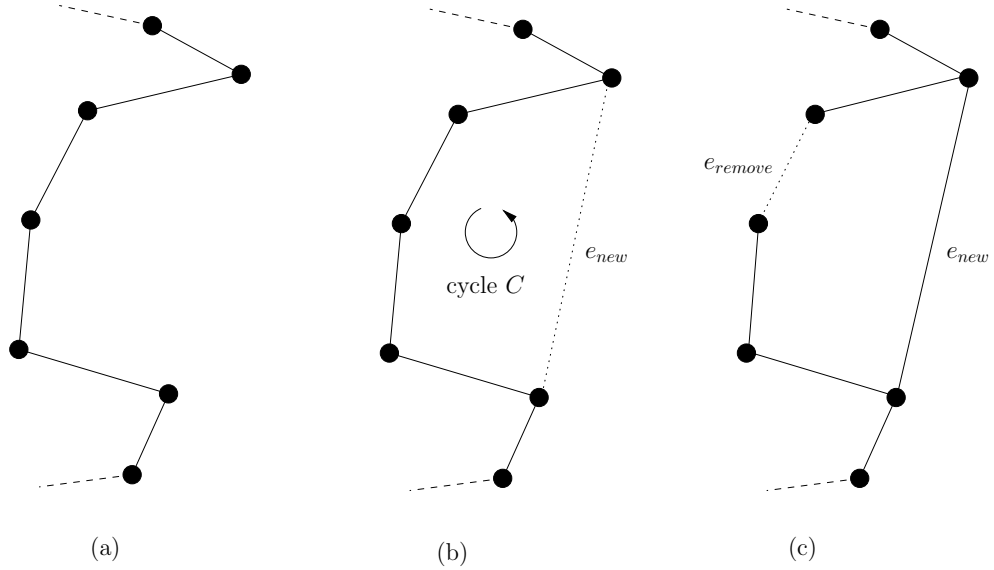


Figure 4.1: Exchanging two edges in KLS. (a) given part of the spanning tree, (b) try to include edge e_{new}, thereby inducing cycle C and (c) found replaceable (i.e. worse) edge e_{remove} to be removed and finally include c_{new}.

Furthermore it was investigated if the approximate primal solution, available when using the Volume Algorithm, could lend itself to derive better solutions. Unfortunately neither the direct solution generated by calling $solveMST(x_P)$, i.e. determine the MST by using the probability as some sort of profit and thus preferring edges which are likely to be in the optimal primal solution, nor using the approximate values to sort the edges in case of $klsType = 2$ turned out to be beneficial.
Regarding the target value T, mentioned in the algorithm sections, one can either decide to use the locally improved solutions for updating T or not, which is determined by the binary parameter $useLBfromKLS$.

5 Experimental Results of Lagrangian Decomposition

This chapter introduces the test instances, presents various experimental results and compares them to previous ones.
If not stated otherwise the test environment consists for the rest of this thesis of a Pentium M 1.6GHz with 1.25GB RAM using GCC in version 4.0.3. All attributes which will be used are listed in Table 5.1.

5.1 Used Test Instances

An instance is determined by the number of nodes n, the number of edges m, the knapsack capacity c and tuples of the form $(e_i, v_j, v_k, p(e_i), w(e_i)),\ i = 1, \ldots, m$, stating that the i-th edge connects nodes v_j and v_k, has profit $p(e_i)$ and weight $w(e_i)$.
All test instances can be categorized according to graph type and correlation type. We use three graph types (see Section 2.5 for a definition): plane graphs, maximal plane graphs and complete graphs. The *correlation type* states the correlation between weight $w(e)$ and profit $p(e)$ of an edge e. Three correlations were defined in [42]:

- uncorrelated:
  $p(e)$ and $w(e)$ are independently and uniformly distributed over $[1, 100]$
- weakly correlated:
  $w(e)$ are independently and uniformly distributed over $[1, 100]$ and
  $p(e) := \lfloor 0.8 * w(e) + v(e) \rfloor$, $v(e)$ is uniformly random over $[1, 20]$
- strongly correlated:
  $w(e)$ are independently and uniformly distributed over $[1, 100]$ and
  $p(e) := \lfloor 0.9 * w(e) + 10 \rfloor$.

The type of correlation is in the following denoted as the first letter (u, w or s) being subscripted.

We will also set the *knapsack capacity* c for all instances as defined in [42]:

- $c = 35 \cdot n$ for plane and maximal plane graphs and
- $c = 20 \cdot n - 20$ for complete graphs.

Attribute	Description
$graph$	The used graph.
$corr$	The type of *correlation* between weight and profit: u = uncorrelated, w = weakly correlated and s = strongly correlated.
$graph_{corr}$	The combination of both (see above).
c	The knapsack capacity.
$t[s]$	Avg. running time in seconds.
$iter$	Avg. number of iterations until a stopping criterion is met.
LB	Avg. best found lower bound (i.e., feasible solution with highest objective value).
UB	Avg. best found upper bound.
$\%\text{-}gap$	$= \frac{UB-LB}{LB} \cdot 100\%$, i.e., avg. relative difference between LB and UB against LB.
$\sigma_{\%\text{-}gap}$	The standard deviation of $\%\text{-}gap$.
Opt	How many instances out of ten were solved to provable optimality, i.e. it was checked if $LB = \lfloor UB \rfloor$.
w_{LB}	Avg. weight of the best feasible solution found ($=LB$).

Table 5.1: Attributes used in the evaluation.

5.1.1 Test Set 1

The instances used by Yamada et al. [42] consist of plane graphs (not being maximal plane) ranging from 50 to 1000 nodes and complete graphs between 20 and 200 nodes. The structure of the plane graphs was fixed by them, e.g. see Figure 5.1 for a visualization of the graph $P_{200,560}$, whereas the structure of all complete graphs is unambiguous. Since they gave us their instance generator we tried to reproduce their graphs at the best possible rate, but of course the computed random values for the profits and weights are not the same. We also randomly generated 10 instances per combination of graph and correlation type to be tested, among them the same combinations as they used as well as for each correlation type all remaining larger graphs to have an extended test set, see Table 5.2 for all instances.

5.1.2 Test Set 2

We generated further instances for more extensive testing by an own instance generator (see Chapter 7) which can in addition to complete graphs also produce maximal plane graphs. The second test set is presented in Table 5.3. Like before we randomly generated 10 instances per graph and correlation type.

From now on if we loosely speak of an instance we always actually mean the set of 10 randomly generated instances of that type, which will be referred to by *graph* in addition

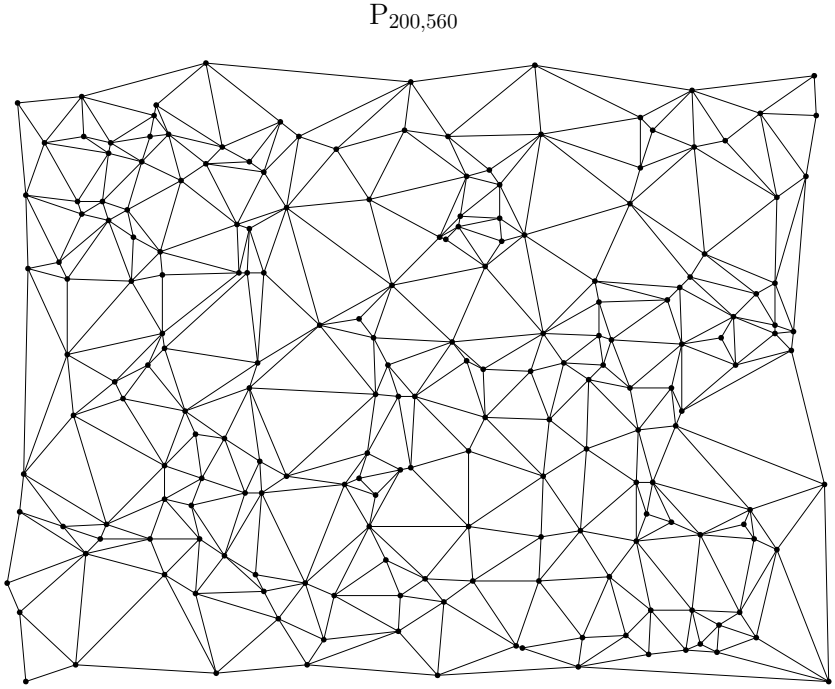


Figure 5.1: The structure of the graph $P_{200,560}$.

plane graphs			**complete graphs**		
graph	*corr*	*c*	*graph*	*corr*	*c*
$P_{50,127}$	u,w,s	1750	K_{20}	w,s	380
$P_{100,260}$	u,w,s	3500	K_{30}	s	580
$P_{200,560}$	u,w,s	7000	K_{40}	u,w,s	780
$P_{400,1120}$	u,w,s	14000	K_{60}	u,w,s	1180
$P_{600,1680}$	u,w,s	21000	K_{80}	u,w,s	1580
$P_{800,2240}$	u,w,s	28000	K_{100}	u,w,s	1980
$P_{1000,280}$	u,w,s	35000	K_{120}	u,w,s	2380
			K_{140}	u,w,s	2780
			K_{160}	u,w,s	3180
			K_{180}	u,w,s	3580
			K_{200}	u,w,s	3980

Table 5.2: Instances of test set 1.

maximal plane graphs			large complete graphs		
graph	*corr*	*c*	*graph*	*corr*	*c*
P_{2000}	u,w,s	70000	K_{250}	u,w,s	4980
P_{4000}	u,w,s	140000	K_{300}	u,w,s	5980
P_{6000}	u,w,s	210000	K_{350}	u,w,s	6980
P_{8000}	u,w,s	280000	K_{400}	u,w,s	7980
P_{10000}	u,w,s	350000	K_{450}	u,w,s	8980
P_{12000}	u,w,s	420000	K_{500}	u,w,s	9980

Table 5.3: Instances of test set 2.

with *corr* or mostly by $graph_{corr}$ alone. The knapsack capacity c is left out and can be looked up in the mentioned table of the corresponding test set.

5.2 Comparing the MST Algorithms

Now we will analyze the performance of the different variants of the MST algorithms described in Section 4.2 when applied in our Lagrangian algorithm (using the corresponding settings described later in Section 5.4). To do this we will run all variants (by changing the parameter $mstAlg$) on different graphs (all 10 instances each), record the time spent of the MST algorithm and compare the average values over all 10 instances per graph. Thereby the number of iterations of the Lagrangian algorithm is fixed to 600 for maximal plane graphs and to 1200 for complete graphs. The limit is due to have the same test conditions per graph type since taking higher or no limits at all could lead to convergence and thus falsifying the results.
The average time of each variant for a collection of maximal plane and complete graphs is presented in Table 5.4. It can be seen that the average times are very different and considering efficiency it is of great importance to choose the right variant for a specific graph type. The variant with Kruskal's algorithm using a priority queue takes the most time and is thus uninteresting, except that using a Pairing-Heap instead of a Fibonacci-Heap leads to a significant improvement. A difference when using either of these heaps can be noticed in Prim's algorithm too, whereby the Pairing-Heap is again the priority queue of choice. This is consistent with findings reported in Stasko and Vitter [40] and in Fredman [14], suggesting to take the Pairing-Heap instead of the Fibonacci-Heap. Prim's algorithm using a Pairing-Heap performs best for complete graphs, and will be henceforth used as the default variant for these graphs. When comparing the results of the variants of Kruskal's algorithm, it can be stated that the variant with complete presorting of all edges is the fastest for maximal plane graphs (and assuming for plane graphs too) and will be the default one for these graphs. Though the variant with sorting on demand makes quite a difference compared to complete presorting when used for complete graphs, the before mentioned variant of Prim's algorithm performs better.

Instance		**avg. time of MST algorithm [s]**					
graph	*corr*	Kruskal-P	Kruskal-C	KruskalPQ-f-heap	KruskalPQ-p-heap	Prim-f-heap	Prim-p-heap
P_{2000}	u	**0.78**	1.95	5.42	2.85	2.35	1.55
	w	**0.79**	1.99	5.59	2.89	2.21	1.57
	s	**0.87**	1.99	5.41	2.86	2.20	1.60
P_{6000}	u	**3.07**	6.77	20.44	10.29	10.18	7.57
	w	**3.12**	6.76	20.79	10.47	10.06	7.57
	s	**3.41**	6.80	20.87	10.61	10.09	7.66
P_{10000}	u	**6.07**	12.00	40.78	20.07	19.30	14.73
	w	**6.14**	12.26	41.14	20.60	19.37	14.80
	s	**6.40**	12.25	41.28	20.34	19.42	14.79
K_{200}	u	3.75	1.97	9.88	7.02	1.27	**1.11**
	w	2.87	1.60	8.99	6.38	1.23	**1.16**
	s	3.56	1.49	8.26	6.30	1.29	**1.14**
K_{300}	u	9.82	4.70	27.87	18.44	3.18	**2.97**
	w	8.32	4.15	28.47	18.58	3.11	**2.83**
	s	8.91	3.70	26.74	17.74	3.08	**2.83**
K_{400}	u	18.68	8.06	72.99	54.64	5.55	**5.25**
	w	16.58	7.80	71.87	51.98	5.57	**5.34**
	s	16.76	7.01	67.89	50.11	5.66	**5.21**

Table 5.4: Required times of different MST algorithms when used within the Lagrangian algorithm.

5.3 Experimental Results of the Subgradient Optimization Method

Solving KCMST-LD with the Subgradient Optimization method will henceforth be denoted as KCMST-LD_{SO}, if additionally the local search KLS is used it will be termed KCMST-$LD_{SO\&KLS}$.
Two of the remaining parameters of the algorithm are set to the values suggested in [4], which seem to be a good choice here, too. Thus the parameter *multiply* is set to 0.5 and *steps* to 30.
For **plane graphs** *maxSteps* is fixed to 500, whereas the configuration of KLS depends on the correlation type and was determined in trial runs:

- uncorrelated: the best variant of KLS, though not improving much, was to set $klsType = 0$ and $klsRetries = 200$
- weakly correlated: again $klsType = 0$ with $klsRetries = 200$, this time having more effect
- strongly correlated: here we chose again $klsType = 0$ but less retries with $klsRetries = 100$, which gives excellent results.

The parameter $klsMinGap$ was fixed for all plane graphs to 0.99, and only "naturally" occurring feasible solutions are used for updating the target value T by setting $useLB\text{-}fromKLS = 0$.

The results of KCMST-LD_{SO} as well as KCMST-$LD_{SO\&KLS}$ on plane graphs using these parameters are presented in Table 5.5. We will first examine the results of KCMST-LD_{SO}. In case of uncorrelated graphs there is a tendency to get better solutions as the graph size increases, even solving all instances to optimality for $P_{800,2240u}$ and $P_{1000,2800u}$. The opposite is true for strongly correlated graphs, there the solutions get worse with increasing graph size. Constantly good results are obtained in case of weak correlation. Although %-*gap* is already small, so is sometimes the number of optimal solutions *Opt*. Therefore we apply KLS and can see that the results are improving for nearly all graphs. The most striking increase in solution quality can be noticed for the strongly correlated graphs, henceforth all instances are solved to optimality. Also improved are the results for some of the weakly correlated graphs, most notably around the size of 400 nodes. For $P_{800,2240w}$ and $P_{1000,2800w}$ no improvement was possible. Most of the uncorrelated graphs are not improved at all, or otherwise not significantly.
The bad results obtained on the smallest graphs are due to the Subgradient Optimization method sometimes being unable to derive the optimal upper bound, thus even though using KLS can narrow the gap and actually find the optimal solution, the number of provably optimal solutions stays the same. This was detected after running the algorithm a few times on these instances and comparing the upper bounds with the solutions obtained by the Branch-And-Cut (B&C) algorithm used in [33].

The configuration for **maximal plane graphs** was $maxSteps = 300$, which is less than before but seemed to be enough (except for P_{10000s} where it was not possible to derive the optimal upper bound for one instance, so we consistently used $maxSteps = 500$ for all instances of this graph), and a uniform KLS setting with $klsType = 0$, $klsRetries = 100$ and $klsMinGap = 0.995$. Test runs suggested to use improved solutions from KLS for updating T, thus setting $useLBfromKLS = 1$. The results of KCMST-LD_{SO} as well as KCMST-$LD_{SO\&KLS}$ on maximal plane graphs are presented in Table 5.6. Without KLS the worst results are clearly obtained for strongly correlated graphs, whereas uncorrelated graphs yield the best results. The results on weakly correlated graphs are also quite good but a bit worse than in case of no correlation. Again the relative gap %-*gap* is very small for all graphs but good values of *Opt* are missing. This is different when additionally applying KLS at the end of the process. Though this increases the running time remarkably, in many cases a lot more optimal solutions are found. Again the degree of improvement is roughly proportional to the strength of correlation. There was only one graph, P_{12000u}, where the results without KLS are slightly better.

By using KCMST-$LD_{SO\&KLS}$ the results on the different correlation types are quite identical, showing no clear advantage for a specific correlation, which is apparent when summing up the optimal solutions per correlation type (see Table 5.7).

It was not possible to find suitable parameter values to reasonably solve complete graph instances, except a few small ones. With increasing graph size the Subgradient Optimization method took more and more iterations and was unable to derive good bounds. So we will skip these instances for this algorithm.
Since the Volume Algorithm does not have this problem, and performs, as will soon be seen, on the other instances also very well, we will from now on concentrate more on this method, thereby going into more detail.

Instance	**KCMST-LD$_{\mathbf{SO}}$**								**KCMST-LD$_{\mathbf{SO\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
$P_{50,127u}$	0.23	763	3558.4	3560.7	0.06539	0.08993	3	1747.8	0.28	779	3559.0	3560.7	0.04758	0.04916	3	1748.3
$P_{50,127w}$	0.12	613	2062.9	2064.3	0.06710	0.11296	6	1748.9	0.26	604	2063.6	2064.3	0.03357	0.050591	6	1749.7
$P_{50,127s}$	0.11	636	2051.7	2052.2	0.02438	0.04739	7	1749.4	0.17	506	2052.2	2052.2	0	0	10	1750.0
$P_{100,260u}$	0.13	590	7222.9	7223.4	0.00676	0.01317	7	3498.1	0.29	605	7223.0	7223.4	0.00540	0.00945	7	3498.6
$P_{100,260w}$	0.13	586	4168.0	4168.3	0.00724	0.01165	7	3499.5	0.34	611	4168.0	4168.3	0.00724	0.01165	7	3499.9
$P_{100,260s}$	0.19	710	4115.1	4115.5	0.00972	0.01699	7	3499.5	0.17	570	4115.5	4115.5	0	0	10	3500.0
$P_{200,560u}$	0.20	642	14896.9	14897.3	0.00268	0.00471	7	6999.4	0.53	622	14896.9	14897.3	0.00268	0.00471	7	6999.2
$P_{200,560w}$	0.22	627	8431.5	8432.0	0.00595	0.01012	7	6999.7	0.32	467	8432.0	8432.0	0	0	10	7000.0
$P_{200,560s}$	0.29	821	8243.9	8244.3	0.00484	0.00624	6	6999.7	0.32	626	8244.3	8244.3	0	0	10	7000.0
$P_{400,1120u}$	0.43	753	29734.8	29735.8	0.00337	0.00448	5	13999.3	1.09	758	29735.1	29735.8	0.00236	0.00357	6	13999.4
$P_{400,1120w}$	0.39	637	16794.5	16794.9	0.00238	0.00417	7	13999.5	0.64	503	16794.9	16794.9	0	0	10	14000.0
$P_{400,1120s}$	0.43	734	16500.0	16500.3	0.00182	0.00408	8	13999.8	0.62	651	16500.3	16500.3	0	0	10	14000.0
$P_{600,1680u}$	0.60	747	44836.2	44836.7	0.00112	0.00158	6	20999.6	1.30	674	44836.4	44836.7	0.00067	0.00107	7	20999.7
$P_{600,1680w}$	0.52	654	25157.8	25158.1	0.00120	0.00269	8	20999.6	1.15	631	25157.9	25158.1	0.00080	0.00252	9	20999.9
$P_{600,1680s}$	0.82	981	24754.1	24755.4	0.00526	0.00541	4	20998.5	0.92	692	24755.4	24755.4	0	0	10	21000.0
$P_{800,2240u}$	0.58	573	59814.5	59814.5	0	0	10	27999.8	1.31	543	59814.5	59814.5	0	0	10	27999.7
$P_{800,2240w}$	0.62	597	33540.4	33540.5	0.00030	0.00094	9	27999.8	1.49	596	33540.4	33540.5	0.00030	0.00094	9	28000.0
$P_{800,2240s}$	1.15	1072	33006.4	33007.8	0.00424	0.00409	3	27998.3	1.22	704	33007.8	33007.8	0	0	10	28000.0
$P_{1000,2800u}$	0.72	575	74835.6	74835.6	0	0	10	34999.8	1.49	539	74835.6	74835.6	0	0	10	34999.9
$P_{1000,2800w}$	0.87	672	41980.7	41980.9	0.00048	0.00101	8	35000.0	1.90	653	41980.7	41980.9	0.00048	0.00101	8	35000.0
$P_{1000,2800s}$	1.39	1035	41262.8	41263.8	0.00240	0.00226	4	34998.8	1.64	740	41263.8	41263.8	0	0	10	35000.0

Table 5.5: Results of KCMST-LD$_{\mathrm{SO}}$ and KCMST-LD$_{\mathrm{SO\&KLS}}$ on all *plane* graphs.

Instance	**KCMST-LD$_{\mathbf{SO}}$**								**KCMST-LD$_{\mathbf{SO\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
P_{2000u}	1.58	649	147799.5	147799.6	0.00007	0.00022	9	69999.7	2.85	671	147799.5	147799.6	0.00007	0.00022	9	69999.7
P_{2000w}	1.78	678	85570.5	85570.8	0.00035	0.00110	9	69999.5	2.78	663	85570.8	85570.8	0	0	10	69999.9
P_{2000s}	2.57	962	82522.0	82523.3	0.00157	0.00222	4	69998.5	3.09	762	82523.3	82523.3	0	0	10	70000.0
P_{4000u}	3.64	706	294871.9	294872.1	0.00006	0.00012	8	139999.5	6.38	720	294871.9	294872.1	0.00006	0.00012	8	139999.7
P_{4000w}	4.28	807	170957.6	170958.1	0.00030	0.00050	7	139999.3	7.07	797	170958.0	170958.1	0.00006	0.00018	9	139999.9
P_{4000s}	5.54	1021	165048.9	165051.5	0.00157	0.00165	3	139997.5	7.75	894	165051.3	165051.5	0.00012	0.00025	8	140000.0
P_{6000u}	5.97	756	441977.7	441978.1	0.00009	0.00016	7	209999.4	9.67	669	441978.1	441978.1	0	0	10	210000.0
P_{6000w}	6.62	810	256318.1	256318.4	0.00012	0.00019	7	209999.7	10.93	777	256318.2	256318.4	0.00008	0.00016	8	209999.9
P_{6000s}	8.78	1047	247589.7	247592.2	0.00100	0.00086	3	209997.1	12.08	899	247592.0	247592.2	0.00008	0.00016	8	209999.9
P_{8000u}	8.82	825	589446.7	589447.1	0.00007	0.00011	7	279999.3	14.77	773	589447.0	589447.1	0.00002	0.00006	9	279999.5
P_{8000w}	9.64	853	341903.8	341904.4	0.00018	0.00032	7	279999.4	15.18	786	341904.1	341904.4	0.00009	0.00028	9	280000.0
P_{8000s}	12.42	1055	330117.2	330122.1	0.00147	0.00097	2	279995.4	17.87	959	330121.9	330122.1	0.00006	0.00012	8	279999.9
P_{10000u}	12.15	854	737449.8	737450.7	0.00012	0.00015	5	349998.9	18.50	751	737450.6	737450.7	0.00001	0.00003	9	349999.7
P_{10000w}	13.47	899	427405.8	427407.2	0.00032	0.00051	5	349998.7	21.72	862	427406.9	427407.2	0.00006	0.00009	7	349999.8
P_{10000s}	20.09	1349	412635.4	412643.6	0.00198	0.00314	1	349993.7	25.36	937	412643.6	412643.6	0	0	10	350000.0
P_{12000u}	14.96	828	885117.6	885117.8	0.00002	0.00004	8	419999.4	23.97	801	885117.5	885117.8	0.00003	0.00004	7	419999.2
P_{12000w}	18.40	973	512985.4	512987.3	0.00038	0.00055	4	419997.9	26.65	825	512987.3	512987.3	0	0	10	419999.9
P_{12000s}	21.85	1126	495164.1	495169.7	0.00112	0.00125	2	419993.8	31.04	943	495169.5	495169.7	0.00004	0.00008	8	420000.0

Table 5.6: Results of KCMST-LD$_{\mathrm{SO}}$ and KCMST-LD$_{\mathrm{SO\&KLS}}$ on all *maximal plane* graphs.

5.4 Experimental Results of the Volume Algorithm

When the VA is used to solve the problem, we will denote the whole algorithm by KCMST-LD_{VA}, and like before, when KLS is used too, by KCMST-$LD_{VA\&KLS}$.
We will fix the parameter *multiply* to 0.67 and *steps* will be set to 20, as it was done in [2] and [21], which seemed to be robust values. Contrary to the Subgradient Optimization method the same values are suitable regardless of the graph type. The advanced update scheme for the parameter α showed no advantage, merely a little overhead in running time, so we will always use the simple one by setting $optimalAlpha = 0$.

When dealing with **plane graphs** the same parameters selected for the Subgradient Optimization method for these graphs will be used, including the settings in case of KLS is applied. In Table 5.8 are given the results of KCMST-LD_{VA} and KCMST-$LD_{VA\&KLS}$ with this configuration. The results of KCMST-LD_{VA} and KCMST-LD_{SO} (Table 5.5) look quite the same, and all what was said before holds here too, also the same weakness when applied to small plane graphs, i.e. being unable to find the optimal upper bound though already having the optimal lower bound. The only really notable difference is that less strongly correlated graphs are solved to optimality by KCMST-LD_{VA}.
When comparing the results of KCMST-$LD_{VA\&KLS}$ with those of KCMST-$LD_{SO\&KLS}$ (Table 5.5), then it is seen that the former method produces more provably optimal solutions at the larger graphs with 800 and 1000 nodes, though only a total of three, whereas the latter method generally takes less iterations and thus requires less times.

Since we are more interested in optimal solutions and the difference of the running time is anyway not large, we will prefer KCMST-$LD_{VA\&KLS}$. The running time of this method on plane graphs is presented in the upper plot of Figure 5.2, showing for all correlation types a more or less linear dependence on the graph size.

The parameter settings of KCMST-LD_{SO} and KCMST-$LD_{SO\&KLS}$ for **maximal plane graphs** will also be used for KCMST-LD_{VA} and KCMST-$LD_{VA\&KLS}$, since these values turned out to be suitable here too (including to use $maxSteps = 500$ instead of $maxSteps = 300$ for P_{10000s}). The only difference is not to use locally improved solutions for updating the target value T. As both methods exhibit very small values of $\%$-*gap* on all maximal plane graphs and the average running times are again comparable (the variant with Subgradient Optimization is once again faster, but it is only a question of a few seconds) we will oppose the number of optimal solutions obtained, which is done in Table 5.7. The result is quite interesting, revealing that when using no KLS the method of choice is KCMST-LD_{SO}, whereas KCMST-$LD_{VA\&KLS}$ performs slightly better among the variants with KLS, although it is worse on weakly correlated graphs. A possible reason why KCMST-$LD_{VA\&KLS}$ performs sometimes better than KCMST-$LD_{SO\&KLS}$ may be because it applies KLS to solutions being a little worse than in the latter method, thus leaving more room for improvement instead of using it on probably locally optimal solutions. An evidence for this are the mentioned fewer optimal solutions of KCMST-LD_{VA} compared to KCMST-LD_{SO}.

corr	$\sum Opt$ **of KCMST-LD** using			
	SO	**VA**	**SO&KLS**	**VA&KLS**
u	**44**	43	52	**56**
w	**39**	26	**53**	47
s	**15**	12	52	**56**
$\sum$	98	81	157	159

Table 5.7: Number of provably optimal solutions on *maximal plane* graphs.

In the end none of the two methods is clearly better than the other. It seems that KCMST-LD$_{\text{VA\&KLS}}$ is to prefer for uncorrelated and strongly correlated maximal plane graphs and KCMST-LD$_{\text{SO\&KLS}}$ in case of weak correlation. The average running time of KCMST-LD$_{\text{VA\&KLS}}$ depending on the number of nodes is displayed in the lower plot of Figure 5.2, where there is clearly shown to be a linear dependence for all correlation types. The progression of the lower and upper bound for three of the largest maximal plane graphs (P$_{12000u1}$, P$_{12000w1}$ and P$_{12000s1}$) is visualized at the left side of Figure 5.4. As can be seen the algorithm shows the characteristic saw-tooth pattern for the upper bound with a slower convergence at the end. The lower bound bears a certain resemblance to this progress, but being updated less often with increasing correlation.

When the VA is applied to **complete graphs** we set $maxSteps = 1000$, thus allowing a longer period of none or small improvement. First tests suggested that the variant of KLS where information from solving the Lagrangian dual is used to guide the local search is superior to the other two variants, thereby using the current Lagrangian multipliers denoted by λ^t in the VA (see Algorithm 4.1.2) to derive the modified profit values mentioned in Section 4.4. Therefore when using KLS we choose $klsType = 2$, $klsRetries = 100$ and $klsMinGap = 0.99$. Contrary to plane graphs we also update the target value T with solutions produced by KLS, i.e. we set $useLBfromKLS = 1$.

The results of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on all complete graphs of test set 1 are given in Tables 5.10 and 5.11. The hardest instances to solve (when looking at the resulting %-*gap*) for KCMST-LD$_{\text{VA}}$ up to 100 nodes are the strongly correlated graphs and above the weakly correlated graphs, although they take less time than the former graphs. Almost all uncorrelated and strongly correlated instances are solved to optimality when $n \geq 120$, except for K$_{180u}$, but there was only 1 out of 10 instances not solved provably optimal.

When running KCMST-LD$_{\text{VA\&KLS}}$ on these instances then %-*gap* is always at least as good (i.e. low) as without KLS applied, in fact in most cases even being zero. Furthermore the weakly correlated instances with $n \geq 120$ are no longer a problem. It was again found that all instances of uncorrelated graphs and one of K$_{20w}$ which were not solved to optimality, except of K$_{60u5}$, was due to the VA being unable to derive the optimal upper bound. This was again checked by running the algorithm a few times on these instances and comparing the upper bounds with the solutions obtained in [33]. Fortunately this sort

of flaw occurs seldom and rather only on small graphs (actually supposed to be easier to solve), but as we will see in Section 5.4.2, there is a possible way to circumvent this to a certain degree.

We will use the same settings as for complete graphs for the **large complete graphs** of test set 2. The results are presented in Table 5.12. We will begin by looking at the results of KCMST-LD$_{\text{VA}}$. This time all uncorrelated graphs are solved to optimality, and the algorithm persists to deliver the worst results on the weakly correlated graphs. The results on the strongly correlated graphs are twofold, being very satisfying up to 400 nodes, but quite bad for $n = 450$ and $n = 500$, particularly the values of Opt.
Additionally applying KLS, which is done in KCMST-LD$_{\text{VA\&KLS}}$, yields superb results on all large complete graphs. Finally the algorithm delivers for all 10 instances per graph the optimal bounds, except for one of K$_{300w}$. The effects of KLS will be addressed in more detail in the next section.
The running time of KCMST-LD$_{\text{VA}}$ on all uncorrelated complete graphs (since using KLS brought no improvement) as well as the running times of KCMST-LD$_{\text{VA\&KLS}}$ on all weakly and strongly correlated complete graphs are illustrated in Figure 5.3. The curves for complete and large complete graphs are looking very similar. It can be seen that the times of uncorrelated and weakly correlated graphs are nearly equal and showing a running time about roughly proportional to $n^{1.5}$, but bearing in mind that KLS is used for the latter but not former graphs. The strongly correlated graphs clearly demand more iterations and yielding a running time that seems to be proportional to n^2.
The course of the lower and upper bound for three of the largest complete graphs, namely K$_{500u1}$, K$_{500w1}$ and K$_{500s1}$ are represented at the right side of Figure 5.4. Contrary to plane graphs there is a rather smooth progression towards the optimal upper bound. Apart from the beginning there is no big change in the course of the upper bound. Also unlike in the case of plane graphs the algorithm starts out with extremely high values for the upper bound, especially for the uncorrelated case. This was the reason to use another updating scheme for the target value T for complete graphs, because otherwise the algorithm would go against this by reducing the parameter f already too much at the beginning and yield worse results.

Instance	**KCMST-LD$_{\mathbf{VA}}$**								**KCMST-LD$_{\mathbf{VA\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
$P_{50,127u}$	0.19	983	3558.5	3560.7	0.06256	0.08970	3	1747.6	0.30	976	3559.0	3560.7	0.04758	0.04916	3	1748.3
$P_{50,127w}$	0.15	745	2063.2	2064.3	0.05280	0.07975	6	1749.3	0.23	751	2063.6	2064.3	0.03357	0.05059	6	1749.7
$P_{50,127s}$	0.16	815	2051.3	2052.2	0.04392	0.06281	5	1749.4	0.12	573	2052.2	2052.2	0	0	10	1750.0
$P_{100,260u}$	0.17	801	7222.9	7223.4	0.00676	0.01317	7	3497.8	0.37	817	7222.9	7223.4	0.00676	0.01317	7	3498.1
$P_{100,260w}$	0.17	732	4167.9	4168.3	0.00967	0.01694	7	3499.6	0.36	724	4168.0	4168.3	0.00724	0.01165	7	3499.7
$P_{100,260s}$	0.23	829	4115.1	4115.5	0.00972	0.01254	6	3499.5	0.18	641	4115.5	4115.5	0	0	10	3500.0
$P_{200,560u}$	0.31	869	14896.7	14897.3	0.00398	0.00560	6	6999.1	0.55	822	14896.9	14897.3	0.00268	0.00471	7	6999.2
$P_{200,560w}$	0.28	730	8431.9	8432.0	0.00119	0.00376	9	7000.0	0.36	634	8432.0	8432.0	0	0	10	7000.0
$P_{200,560s}$	0.35	877	8243.6	8244.3	0.00849	0.01285	6	6999.2	0.32	685	8244.3	8244.3	0	0	10	7000.0
$P_{400,1120u}$	0.55	880	29735.0	29735.8	0.00271	0.00383	6	13999.5	1.15	905	29735.1	29735.8	0.00236	0.00320	6	13999.4
$P_{400,1120w}$	0.49	802	16794.3	16794.9	0.00358	0.00642	7	13999.6	0.77	721	16794.9	16794.9	0	0	10	14000.0
$P_{400,1120s}$	0.45	757	16500.2	16500.3	0.00061	0.00192	9	13999.8	0.58	713	16500.3	16500.3	0	0	10	14000.0
$P_{600,1680u}$	0.79	934	44836.2	44836.7	0.00111	0.00117	5	20999.5	1.52	854	44836.4	44836.7	0.00067	0.00107	7	20999.8
$P_{600,1680w}$	0.65	779	25158.0	25158.1	0.00040	0.00126	9	20999.8	1.29	788	25158.0	25158.1	0.00040	0.00126	9	21000.0
$P_{600,1680s}$	0.85	996	24754.3	24755.4	0.00444	0.00586	5	20998.6	0.88	745	24755.4	24755.4	0	0	10	21000.0
$P_{800,2240u}$	0.79	766	59814.5	59814.5	0	0	10	27999.7	1.59	716	59814.5	59814.5	0	0	10	27999.8
$P_{800,2240w}$	0.92	854	33540.2	33540.5	0.00089	0.00199	8	27999.6	1.76	762	33540.5	33540.5	0	0	10	28000.0
$P_{800,2240s}$	1.30	1215	33005.3	33007.8	0.00757	0.00501	0	27997.5	1.13	731	33007.8	33007.8	0	0	10	28000.0
$P_{1000,2800u}$	0.99	764	74835.6	74835.6	0	0	10	34999.9	2.08	764	74835.6	74835.6	0	0	10	34999.8
$P_{1000,2800w}$	1.21	899	41980.0	41980.9	0.00215	0.00379	6	34999.2	1.88	762	41980.9	41980.9	0	0	10	35000.0
$P_{1000,2800s}$	1.50	1128	41262.6	41263.8	0.00288	0.00189	2	34999.0	1.46	753	41263.8	41263.8	0	0	10	35000.0

Table 5.8: Results of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on all *plane* graphs.

Instance	**KCMST-LD$_{\mathbf{VA}}$**								**KCMST-LD$_{\mathbf{VA\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
P_{2000u}	2.32	867	147799.4	147799.6	0.00014	0.00029	8	69999.6	3.26	813	147799.6	147799.6	0	0	10	69999.7
P_{2000w}	2.42	883	85570.1	85570.8	0.00081	0.00109	6	69999.4	3.29	808	85570.7	85570.8	0.00012	0.00037	9	70000.0
P_{2000s}	2.97	1045	82520.9	82523.3	0.00290	0.00309	2	69998.1	2.87	815	82523.3	82523.3	0	0	10	70000.0
P_{4000u}	4.64	854	294872.0	294872.1	0.00003	0.00009	9	139999.4	7.37	835	294872.0	294872.1	0.00003	0.00009	9	139999.8
P_{4000w}	5.44	1040	170957.1	170958.1	0.00060	0.00048	3	139998.8	7.77	907	170957.7	170958.1	0.00024	0.00050	8	139999.8
P_{4000s}	6.10	1071	165048.9	165051.5	0.00157	0.00163	2	139997.9	7.92	916	165051.4	165051.5	0.00006	0.00018	9	140000.0
P_{6000u}	8.16	953	441977.5	441978.1	0.00013	0.00022	6	209999.2	12.46	898	441978.1	441978.1	0	0	10	210000.0
P_{6000w}	9.12	1033	256316.7	256318.4	0.00067	0.00084	4	209998.6	12.66	934	256318.3	256318.4	0.00004	0.00012	9	210000.0
P_{6000s}	9.94	1094	247588.6	247592.2	0.00145	0.00187	2	209996.4	12.68	950	247592.2	247592.2	0	0	10	210000.0
P_{8000u}	11.15	906	589446.9	589447.1	0.00004	0.00008	8	279999.5	19.58	975	589446.9	589447.1	0.00004	0.00008	8	279999.4
P_{8000w}	13.89	1102	341901.7	341904.4	0.00080	0.00086	3	279996.9	19.49	981	341904.0	341904.4	0.00012	0.00020	7	279999.6
P_{8000s}	14.22	1087	330117.3	330122.1	0.00144	0.00145	3	279994.9	17.02	887	330122.0	330122.1	0.00003	0.00009	9	280000.0
P_{10000u}	15.92	969	737450.2	737450.7	0.00007	0.00012	7	349999.5	24.63	956	737450.6	737450.7	0.00001	0.00003	9	349999.8
P_{10000w}	16.31	964	427406.4	427407.2	0.00019	0.00039	7	349998.9	25.51	1021	427406.9	427407.2	0.00006	0.00009	7	349999.7
P_{10000s}	23.42	1383	412640.1	412643.6	0.00084	0.00087	1	349996.7	26.61	1025	412643.6	412643.6	0	0	10	350000.0
P_{12000u}	21.67	1056	885117.0	885117.8	0.00008	0.00010	5	419998.5	29.20	921	885117.8	885117.8	0	0	10	419999.7
P_{12000w}	23.27	1102	512985.4	512987.3	0.00038	0.00048	3	419998.0	32.69	1033	512986.9	512987.3	0.00008	0.00013	7	419999.8
P_{12000s}	25.83	1148	495164.0	495169.7	0.00114	0.00138	2	419994.5	34.38	1019	495169.5	495169.7	0.00004	0.00008	8	419999.9

Table 5.9: Results of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on all *maximal plane* graphs.

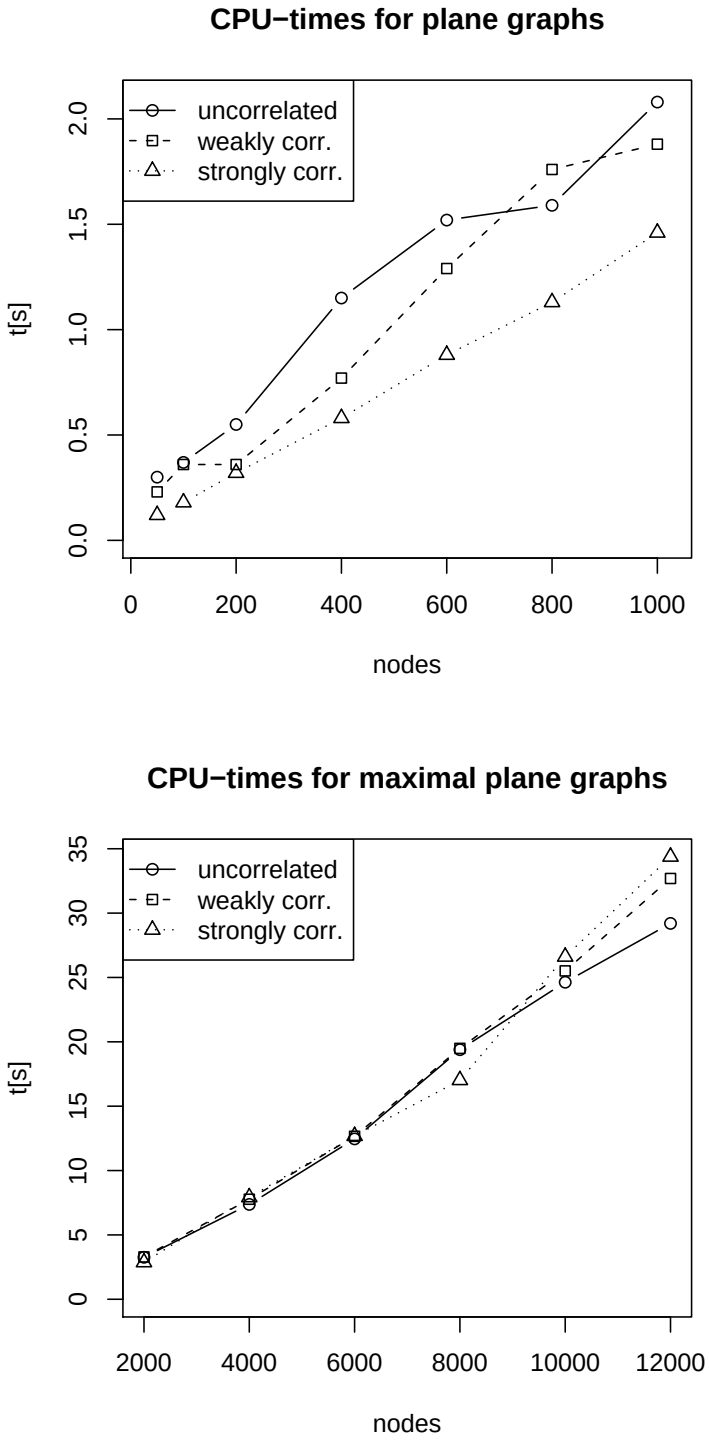


Figure 5.2: CPU-times of KCMST-$LD_{VA\&KLS}$ on plane and maximal plane graphs.

Instance	**KCMST-LD$_{\mathbf{VA}}$**								**KCMST-LD$_{\mathbf{VA\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
K_{20w}	0.11	720	618.9	619.0	0.01701	0.05379	9	379.5	0.12	698	618.9	619.0	0.01701	0.05379	9	379.5
K_{20s}	0.22	960	528.6	528.9	0.05689	0.09160	7	379.7	0.09	635	528.9	528.9	0	0	10	380.0
K_{30s}	0.31	1016	809.2	809.5	0.03712	0.05976	7	579.7	0.16	717	809.5	809.5	0	0	10	580.0
K_{40u}	0.23	880	3669.3	3669.5	0.00550	0.01159	8	778.9	0.28	884	3669.3	3669.5	0.00550	0.01159	8	778.6
K_{40w}	0.24	737	1320.6	1320.7	0.00755	0.02387	9	779.9	0.19	613	1320.7	1320.7	0	0	10	780.0
K_{40s}	0.34	902	1089.9	1090.1	0.01838	0.05812	9	779.7	0.28	782	1090.1	1090.1	0	0	10	780.0
K_{60u}	0.58	1164	5673.3	5673.8	0.00886	0.01250	6	1178.7	0.72	1189	5673.4	5673.8	0.00710	0.00916	6	1177.9
K_{60w}	0.51	891	2017.6	2018.0	0.01987	0.04188	8	1179.8	0.40	676	2018.0	2018.0	0	0	10	1180.0
K_{60s}	1.72	1504	1648.6	1650.5	0.11539	0.12638	4	1177.9	0.57	903	1650.5	1650.5	0	0	10	1180.0
K_{80u}	0.60	858	7672.8	7672.8	0	0	10	1578.7	0.69	847	7672.8	7672.8	0	0	10	1579.2
K_{80w}	0.81	863	2720.4	2720.5	0.00368	0.01163	9	1579.7	0.67	732	2720.5	2720.5	0	0	10	1580.0
K_{80s}	3.94	1889	2208.3	2211.2	0.13146	0.12378	2	1577.2	1.16	1118	2211.2	2211.2	0	0	10	1580.0
K_{100u}	1.07	1062	9698.0	9698.1	0.00103	0.00325	9	1978.0	1.27	1055	9698.0	9698.1	0.00103	0.00325	9	1977.5
K_{100w}	1.10	879	3421.3	3421.4	0.00292	0.00923	9	1979.6	1.02	759	3421.4	3421.4	0	0	10	1980.0
K_{100s}	2.46	1770	2768.1	2771.9	0.13784	0.25039	6	1976.3	2.05	1348	2771.9	2771.9	0	0	10	1980.0

Table 5.10: Results of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on all *complete* graphs (part 1/2).

Instance	**KCMST-LD$_{\mathbf{VA}}$**								**KCMST-LD$_{\mathbf{VA\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
K_{120u}	1.37	1012	11701.2	11701.2	0	0	10	2378.9	1.65	1052	11701.2	11701.2	0	0	10	2378.7
K_{120w}	2.78	1527	4123.3	4124.4	0.02669	0.02415	3	2378.7	1.65	871	4124.3	4124.4	0.00243	0.00768	9	2380.0
K_{120s}	3.38	1845	3332.0	3332.0	0	0	10	2380.0	2.99	1505	3332.0	3332.0	0	0	10	2380.0
K_{140u}	2.08	1184	13721.0	13721.0	0	0	10	2777.2	2.38	1162	13721.0	13721.0	0	0	10	2778.2
K_{140w}	3.29	1351	4824.1	4825.1	0.02073	0.02763	5	2778.7	2.29	913	4825.1	4825.1	0	0	10	2780.0
K_{140s}	5.10	2096	3892.0	3892.0	0	0	10	2780.0	4.77	1756	3892.0	3892.0	0	0	10	2780.0
K_{160u}	2.88	1260	15727.9	15727.9	0	0	10	3176.8	3.19	1213	15727.9	15727.9	0	0	10	3177.9
K_{160w}	4.44	1443	5524.7	5525.7	0.01810	0.02413	5	3178.4	3.27	979	5525.7	5525.7	0	0	10	3180.0
K_{160s}	7.56	2379	4452.0	4452.0	0	0	10	3180.0	6.60	1933	4452.0	4452.0	0	0	10	3180.0
K_{180u}	4.31	1488	17729.2	17729.4	0.00113	0.00357	9	3572.8	4.95	1470	17729.3	17729.4	0.00056	0.00177	9	3576.7
K_{180w}	5.32	1384	6225.6	6226.9	0.02089	0.02839	6	3578.0	4.11	1011	6226.9	6226.9	0	0	10	3580.0
K_{180s}	10.09	2480	5012.0	5012.0	0	0	10	3580.0	9.63	2202	5012.0	5012.0	0	0	10	3580.0
K_{200u}	5.55	1502	19739.4	19739.4	0	0	10	3975.4	6.11	1446	19739.4	19739.4	0	0	10	3977.5
K_{200w}	7.52	1560	6926.5	6928.0	0.02166	0.02745	5	3978.1	5.49	1053	6928.0	6928.0	0	0	10	3980.0
K_{200s}	13.63	2655	5572.0	5572.0	0	0	10	3980.0	12.86	2331	5572.0	5572.0	0	0	10	3980.0

Table 5.11: Results of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on all *complete* graphs (part 2/2).

Instance	**KCMST-LD$_{\mathbf{VA}}$**								**KCMST-LD$_{\mathbf{VA\&KLS}}$**							
$graph_{corr}$	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
K_{250u}	10.79	1746	24747.2	24747.2	0	0	10	4976.7	11.81	1667	24747.2	24747.2	0	0	10	4979.1
K_{250w}	14.80	1822	8928.3	8930.3	0.02240	0.02361	4	4977.6	11.05	1241	8930.3	8930.3	0	0	10	4980.0
K_{250s}	28.85	3252	6972.0	6972.0	0	0	10	4980.0	25.54	2744	6972.0	6972.0	0	0	10	4980.0
K_{300u}	19.20	2001	29770.6	29770.6	0	0	10	5972.6	20.65	1905	29770.6	29770.6	0	0	10	5977.9
K_{300w}	28.44	2315	10729.1	10734.1	0.04662	0.05370	1	5974.2	20.86	1522	10734.0	10734.1	0.00093	0.00294	9	5980.0
K_{300s}	54.03	4080	8372.0	8372.0	0	0	10	5980.0	48.64	3394	8372.0	8372.0	0	0	10	5980.0
K_{350u}	30.42	2187	34793.5	34793.5	0	0	10	6972.0	34.22	2220	34793.5	34793.5	0	0	10	6977.8
K_{350w}	44.23	2420	12530.9	12538.1	0.05751	0.07766	2	6970.6	32.20	1640	12538.1	12538.1	0	0	10	6980.0
K_{350s}	90.76	4540	9771.1	9772.0	0.00922	0.02915	9	6979.0	81.96	4054	9772.0	9772.0	0	0	10	6980.0
K_{400u}	47.99	2499	39818.0	39818.0	0	0	10	7968.3	53.77	2449	39818.0	39818.0	0	0	10	7978.8
K_{400w}	61.72	2456	14339.6	14342.2	0.01813	0.01650	3	7976.7	48.34	1801	14342.2	14342.2	0	0	10	7980.0
K_{400s}	151.61	5392	11172.0	11172.0	0	0	10	7980.0	125.91	4675	11172.0	11172.0	0	0	10	7980.0
K_{450u}	74.63	2940	44841.5	44841.5	0	0	10	8969.8	79.80	2805	44841.5	44841.5	0	0	10	8977.1
K_{450w}	87.57	2619	16143.0	16146.0	0.01860	0.02066	4	8975.7	74.30	2052	16146.0	16146.0	0	0	10	8980.0
K_{450s}	230.77	6092	12561.8	12572.0	0.08128	0.09883	3	8973.6	188.55	5317	12572.0	12572.0	0	0	10	8980.0
K_{500u}	101.52	3239	49860.1	49860.1	0	0	10	9960.6	116.91	3211	49860.1	49860.1	0	0	10	9964.7
K_{500w}	128.94	2938	17945.2	17950.0	0.02675	0.02287	2	9973.8	107.67	2295	17950.0	17950.0	0	0	10	9979.9
K_{500s}	309.53	6545	13955.8	13972.0	0.11618	0.10502	2	9968.1	271.84	5800	13972.0	13972.0	0	0	10	9980.0

Table 5.12: Results of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on all *large complete* graphs.

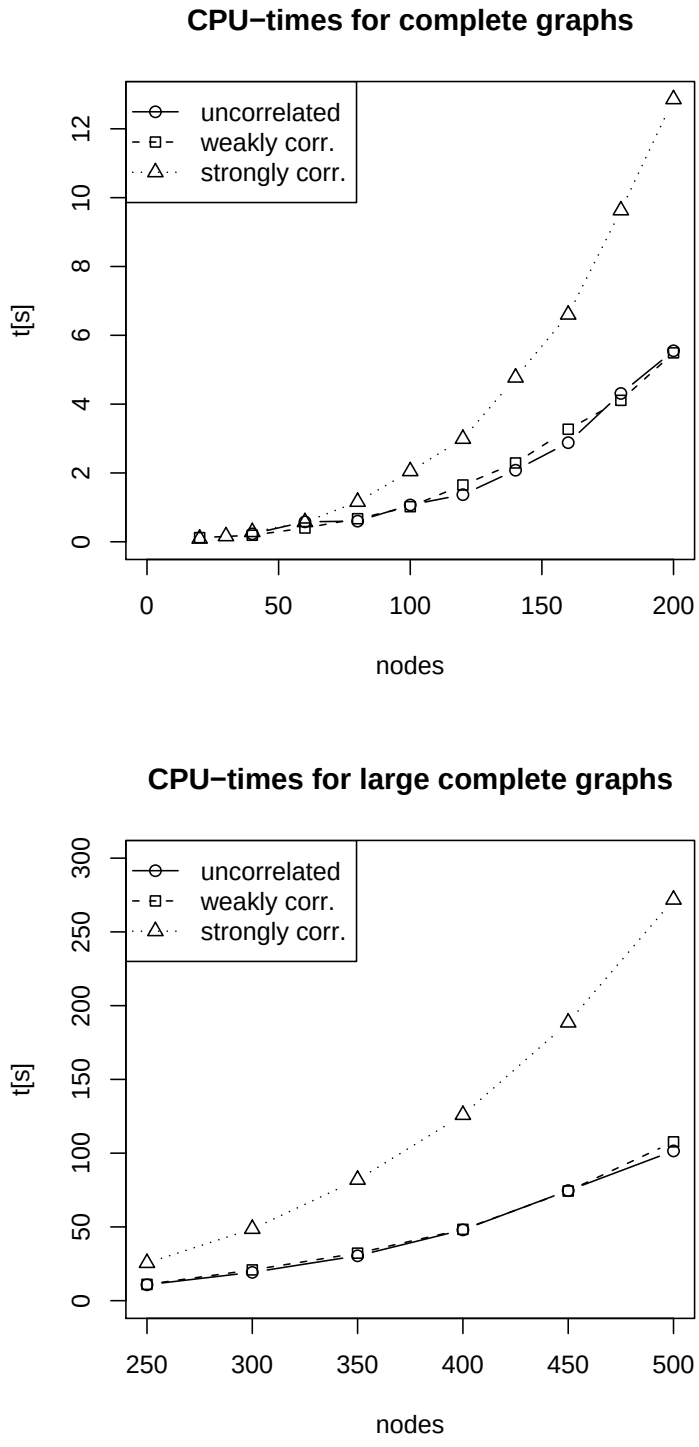


Figure 5.3: CPU-times of KCMST-LD_{VA} on uncorrelated and KCMST-$LD_{VA\&KLS}$ on weakly and strongly correlated complete and large complete graphs.

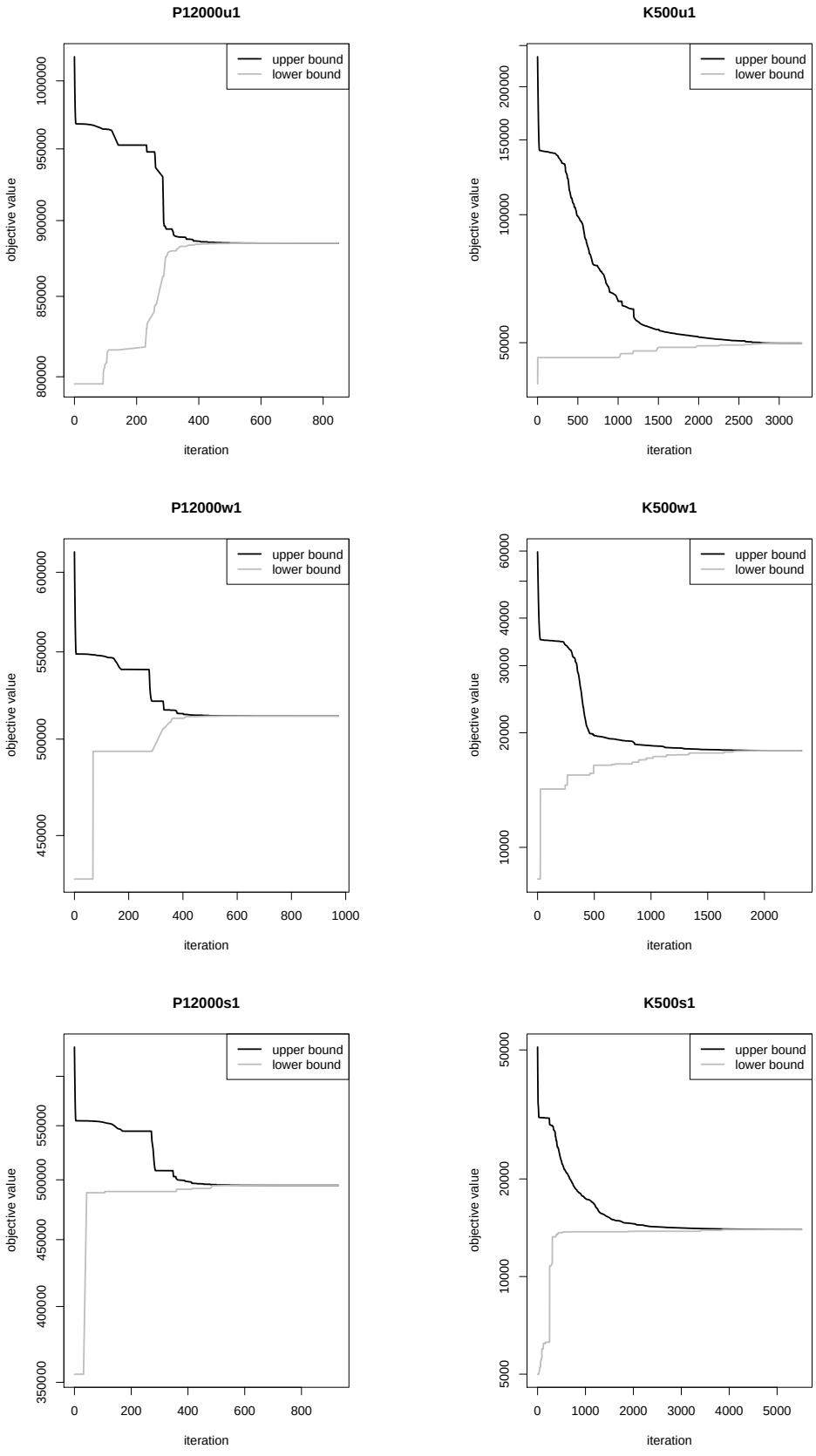


Figure 5.4: Plot of the progression of lower and upper bound on some of the largest instances of maximal plane and complete graphs (with logarithmized y-axis).

5.4.1 Effect of KLS

Now we will consider the effects of the local search KLS in more detail for the graphs of test set 2, since these are larger and thus harder to solve (except seemingly the smallest graphs, which will be discussed in the next section). To see the difference of the results with and without applying KLS more clearly, we present histograms of the running time and boxplots of %-*gap*, the latter giving more information than by stating the average gap and its standard deviation.

First we look at the results for the maximal plane graphs which are illustrated in Figures 5.5, 5.6 and 5.7 for the uncorrelated, weakly and strongly correlated graphs, respectively. In case of maximal plane graphs we have used pure random search with $klsType = 0$. As already mentioned and what can now be clearly seen is that the running time is always increased when using KLS. This increase is on average about 50% for uncorrelated, 40% for weakly and 30% for strongly correlated graphs. When examining the solutions more closely that act as starting solutions for KLS, i.e. the feasible solutions generated during the Lagrangian decomposition, then it turns out that all of them are already very good (i.e. have a high objective value). Anyhow KLS is able to even improve the nearly optimal incumbent uncorrelated solutions, where on average an increase of profit about 0.007% is achieved. There is a bit more room for improvement for the other two correlation types, leading to an average increase of 0.035% for weakly and 0.05% for strongly correlated graphs. To conclude, for maximal plane graphs KLS generally leads to lower bounds that are either provably optimal ($LB = \lfloor UB \rfloor$) or it yields a very tight bound with $LB = \lfloor UB \rfloor - 1$.

The plots for the large complete graphs are presented in Figures 5.8, 5.9 and 5.10 for each correlation type. For this graph type we used guided KLS with $klsType = 2$. As already mentioned before the algorithm solves all uncorrelated graphs to optimality even without applying KLS, so using it does not improve %-*gap* or *Opt* but merely increases the running time at most about 15%. The starting solutions are again very good, so only an average increase of profit of about 0.25% occurs.
The effect on weakly correlated graphs is quite different. There the application of KLS reduces the running time from 15% to 40%, getting less with increasing graph size. This reduction is due to faster convergence because the optimal lower bound is found earlier or in most cases at all, which happens for all instances but one of K_{300w}. This time the average increase of profit is about 5%.
Finally applying KLS on strongly correlated graphs leads again to a decrease of running time between 10% and 18%, which is for graphs up to 400 nodes the only positive effect, except of solving 10 instead of 9 instances to optimality for K_{350s}. However for the two remaining largest graphs K_{450s} and K_{500s} the number of optimal solutions *Opt* is increased substantially, henceforth solving all 10 instances per graph to optimality. The average relative increase of profit by KLS is the highest of all graphs, and ranges from about 13% for K_{250s} to about 23% for K_{500s}. This is due to the starting solutions having the worst objective value so far, but this circumstance seemingly helps in finding the best solution,

since especially for strongly correlated complete graphs it is often found after applying KLS to a rather bad feasible solution and thereby increasing the profit up to 40%.
In summary, applying KCMST-$LD_{VA\&KLS}$ on complete graphs, with the variant of KLS guided by information from solving the Lagrangian dual, almost guarantees to find the optimal solution.

5.4.2 KP vs. E-kKP

As mentioned in Section 4.3 it is also possible to use a special variant of the knapsack problem, the exact k-item knapsack problem, to solve the Lagrangian dual. This may lead to a strengthened Lagrangian relaxation, which will be investigated in this section. Therefore CPLEX 10.0, a commercial ILP solver from ILOG[1] is used to solve the E-kKP. KCMST-LD_{VA} using E-kKP solved by CPLEX will be applied to those instances where KCMST-LD_{VA} using KP solved by the COMBO algorithm was unable to derive the optimal upper bound, among mostly uncorrelated graphs. The tests with CPLEX are run on a Pentium IV having 2.8GHz and 2GB RAM using GCC 3.4. Since the computing environment stated at the beginning of this chapter turned out to be comparable with this one (the former is in fact only weaker to a small degree) we will use the results already obtained before for comparison. Both results of KCMST-LD_{VA} are opposed in Table 5.4.2. We always applied the new variant to all 10 instances per graph to get a better picture of it, apart from the largest complete graphs where only the relevant instances are solved. When using CPLEX the running time is often hundred times more than with the COMBO algorithm, but our interest concerns mainly the relative gap and especially the upper bound. Therefore, when comparing the average upper bounds UB the variant using E-kKP produces lower ones for nearly all graphs, the only exceptions being the graphs $P_{200,560w}$ and the single instance K_{100u4}, where it stays the same. Though the decrease is only small it sometimes suffices to increase the number of instances solved to provable optimality, since the lower bounds were seen to be optimal in most cases, only a corresponding upper bound was missing, as is clearly seen for the graph $P_{50,127u}$. Moreover it happens that the decrease of the upper bound is accompanied by an increase of the lower bound, thus having so to speak a positive influence on the whole process, for example when applied to the single instance K_{180u4}. Nevertheless even when using E-kKP the algorithm is unable to find all optimal upper bounds, especially some instances of the small graphs remain to be a problem.
The boxplot of %-*gap* of these results (except the two single instances) is shown in Figure 5.11.

All in all using E-kKP instead of KP does not strengthen the Lagrangian relaxation much, and regarding the few better lower bounds it was already shown that using KCMST-$LD_{VA\&KLS}$ instead of KCMST-LD_{VA} generally delivers better lower bounds too. Further keeping in mind the drastic increase in running time when using E-KKP by solving it with

[1] see http://www.ilog.com/products/cplex/ (last checked on October 10, 2006)

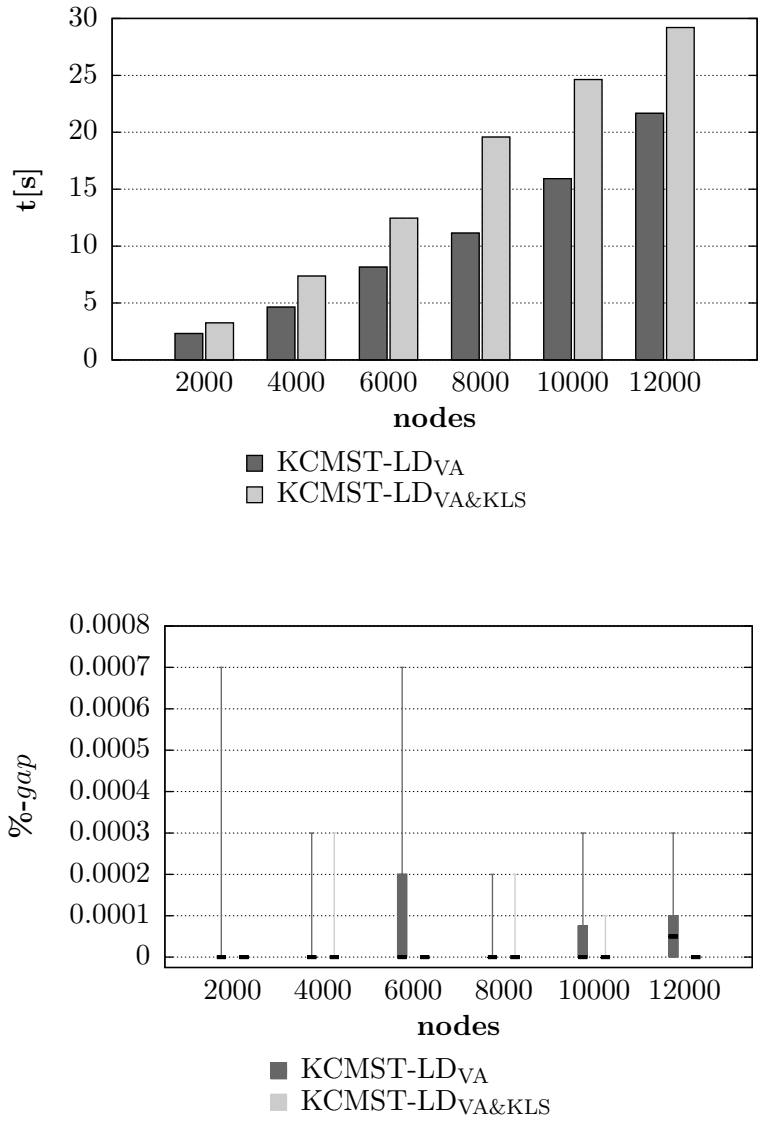


Figure 5.5: Average time and %-*gap* of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on *maximal plane uncorrelated* graphs.

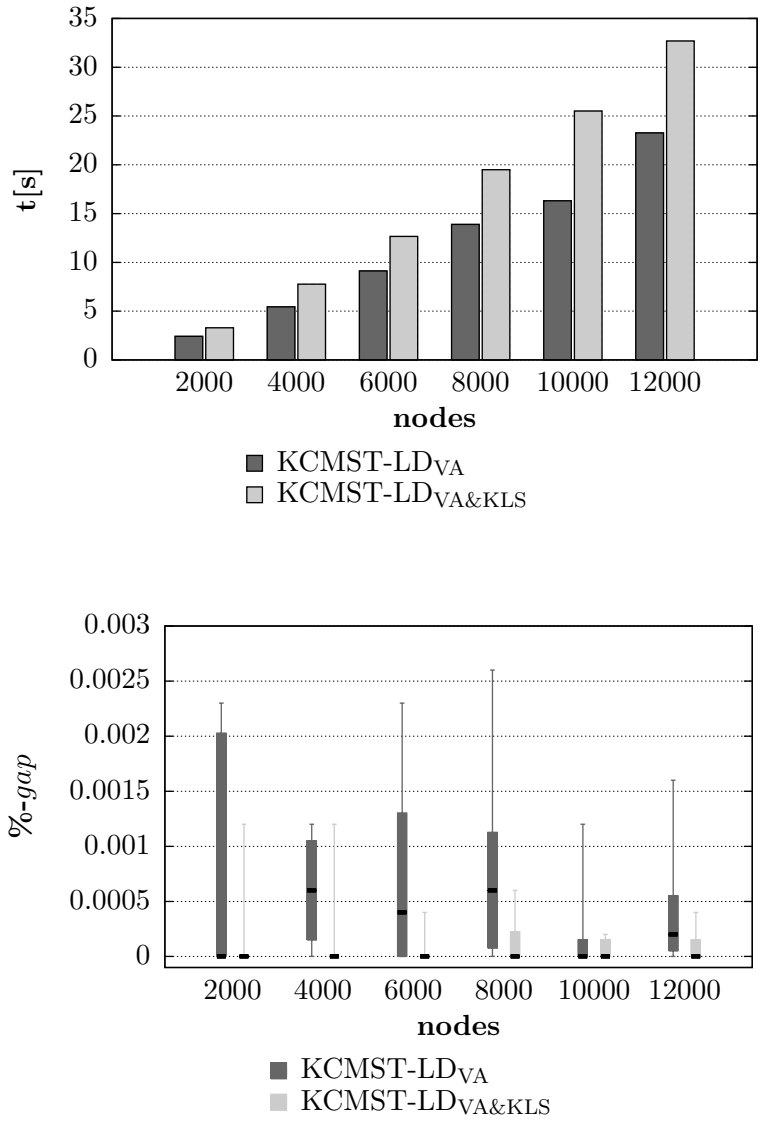


Figure 5.6: Average time and %-*gap* of KCMST-LD$_{VA}$ and KCMST-LD$_{VA\&KLS}$ on *maximal plane weakly correlated* graphs.

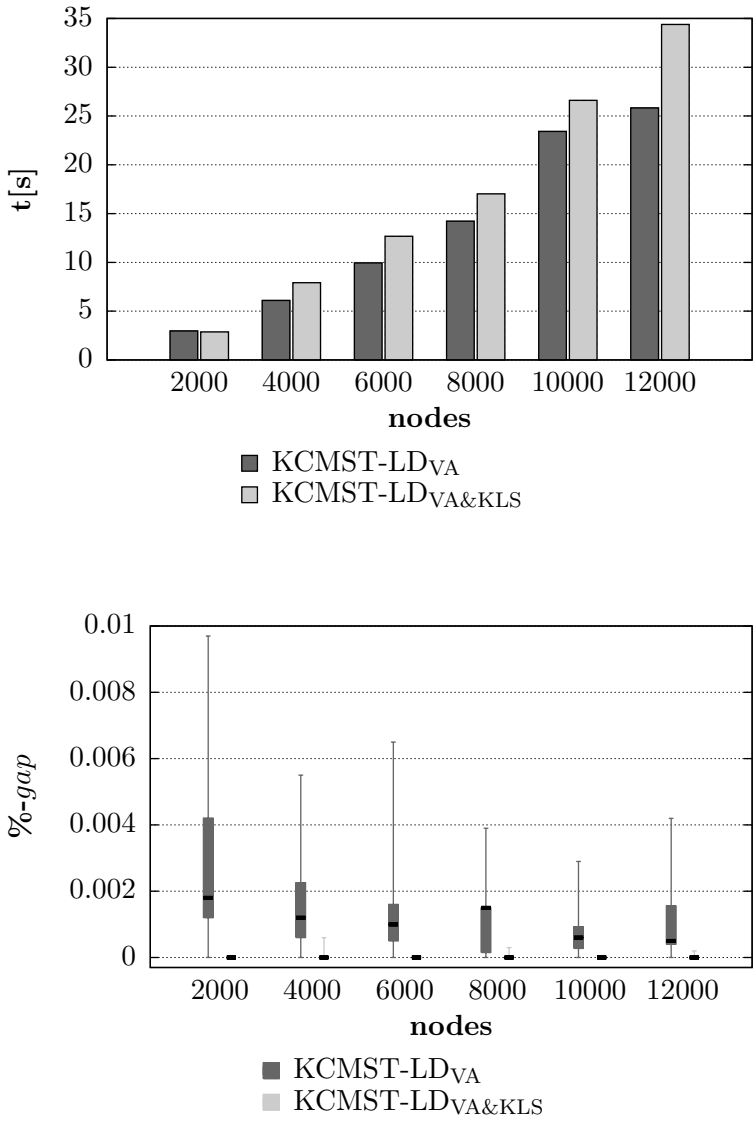


Figure 5.7: Average time and %-*gap* of KCMST-LD$_{\mathrm{VA}}$ and KCMST-LD$_{\mathrm{VA\&KLS}}$ on *maximal plane strongly correlated* graphs.

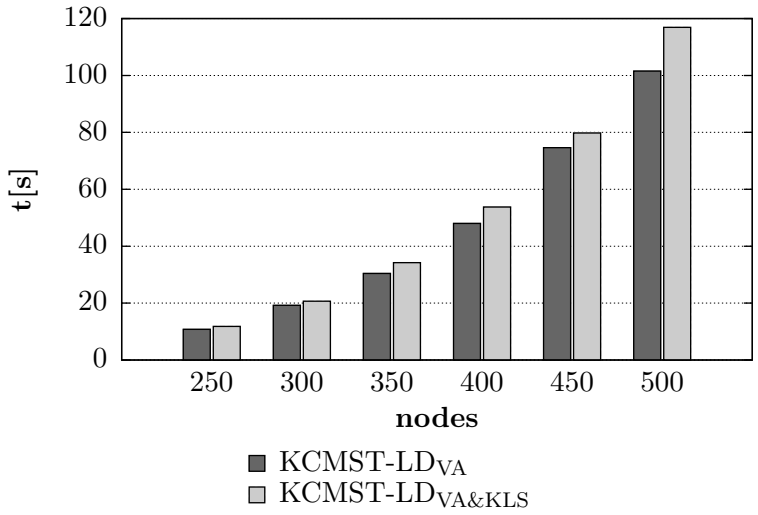


Figure 5.8: Average time of KMCST-LD$_{VA}$ and KMCST-LD$_{VA\&KLS}$ on *large complete uncorrelated* graphs (boxplot of %-*gap* was omitted since all instances were solved to optimality).

CPLEX, we will stick to solving KP with the COMBO algorithm. Although if necessary one could use the former variant to possibly guarantee better bounds and thus a smaller %-*gap* in case only a few optimal solutions are found with KP, which occurs for our instances especially on rather small uncorrelated graphs.

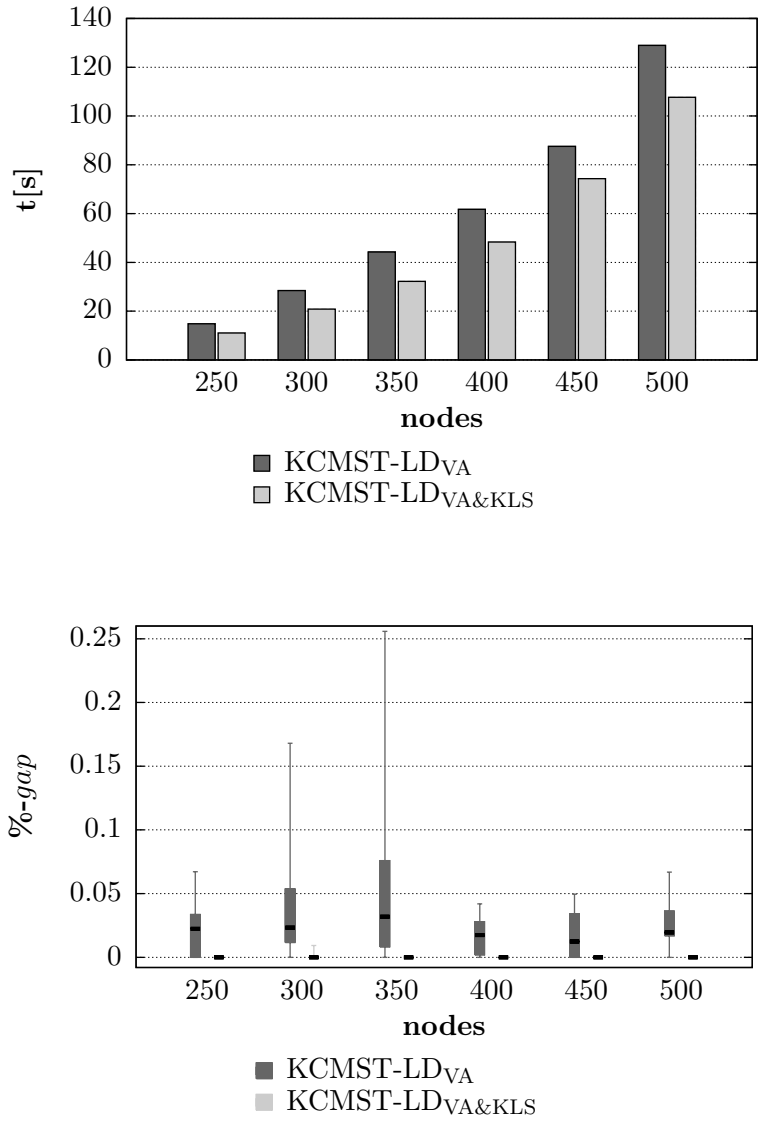


Figure 5.9: Average time and %-*gap* of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on *large complete weakly correlated* graphs.

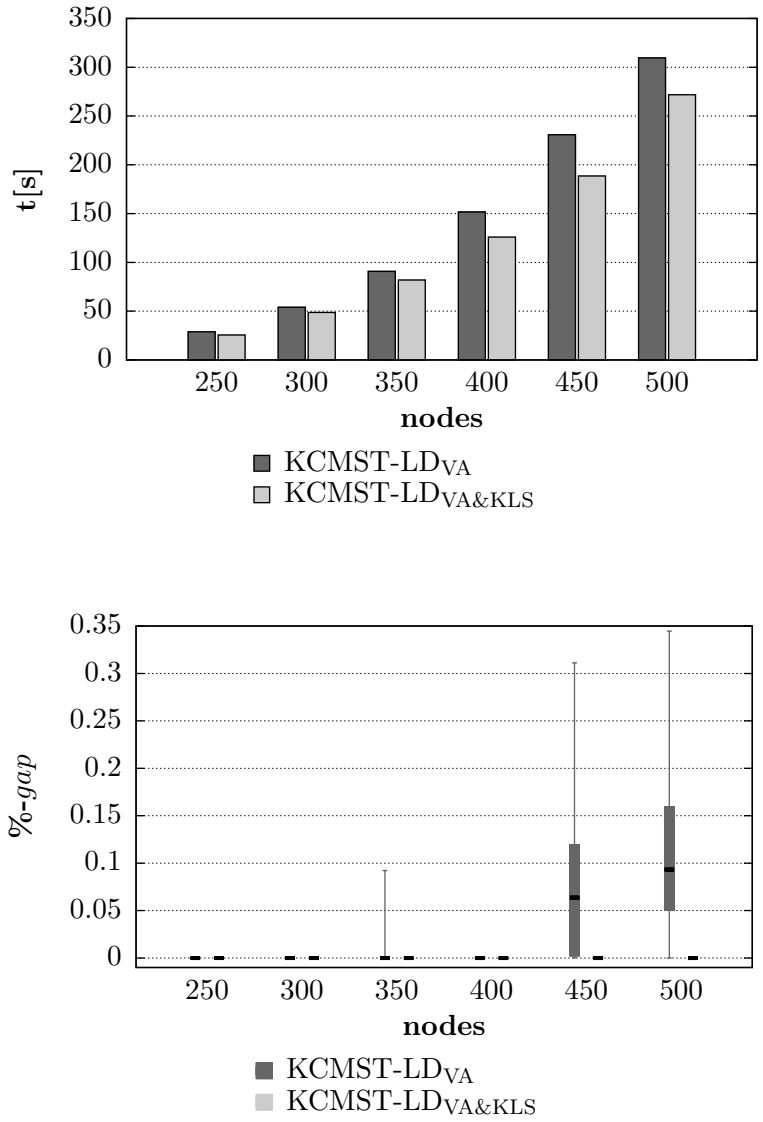


Figure 5.10: Average time and %-*gap* of KCMST-LD$_{\text{VA}}$ and KCMST-LD$_{\text{VA\&KLS}}$ on *large complete strongly correlated* graphs.

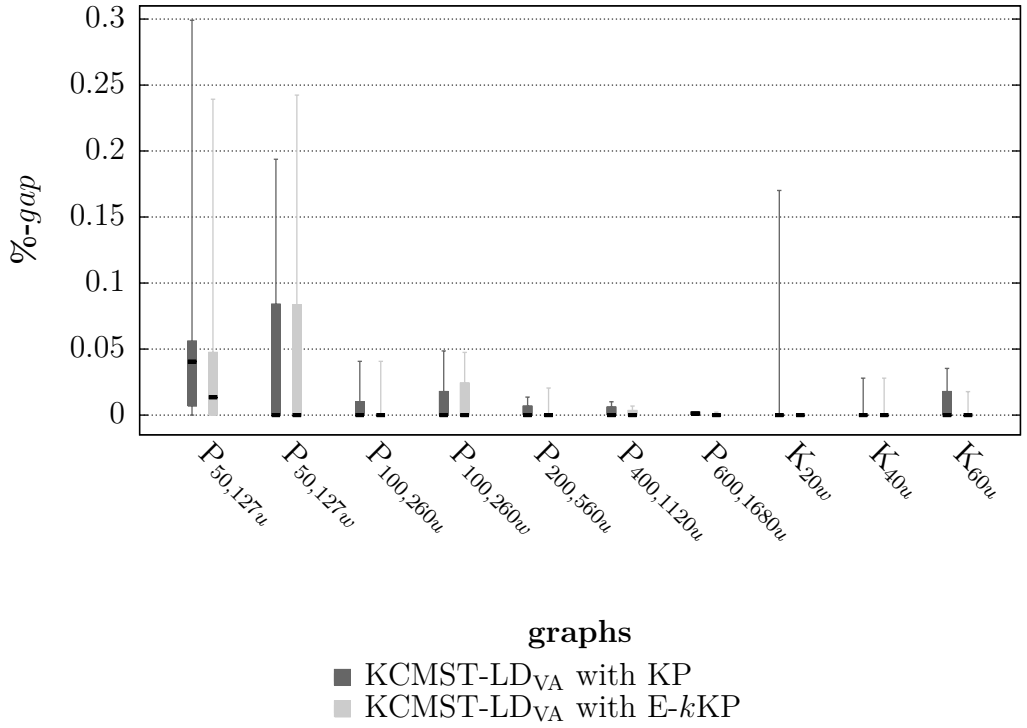


Figure 5.11: Boxplot of %-*gap* of KCMST-LD_{VA} using either KP or E-kKP on selected graphs.

Instance	**B&C**	**KCMST-LD$_{\mathbf{VA}}$**															
	(see [33])	**using KP (COMBO)**								**using E-kKP (CPLEX)**							
$graph_{corr}$	UB	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}	$t[s]$	$iter$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	w_{LB}
$P_{50,127u}$	3559.0	0.19	983	3558.5	3560.7	0.06256	0.08970	3	1747.6	17.71	817	3558.5	3559.9	0.04041	0.07342	5	1747.6
$P_{50,127w}$	2063.7	0.15	745	2063.2	2064.3	0.05280	0.07975	6	1749.3	26.06	732	2063.1	2064.2	0.05294	0.08343	6	1749.7
$P_{100,260u}$	7223.0	0.17	801	7222.9	7223.4	0.00676	0.01317	7	3497.8	32.34	719	7222.9	7223.3	0.00540	0.01308	8	3497.8
$P_{100,260w}$	4168.0	0.17	732	4167.9	4168.3	0.00967	0.01694	7	3499.6	40.25	750	4167.8	4168.3	0.01199	0.01688	6	3499.7
$P_{200,560u}$	14896.9	0.31	869	14896.7	14897.3	0.00398	0.00560	6	6999.1	62.93	798	14896.8	14897.2	0.00272	0.00659	8	6999.8
$P_{400,1120u}$	29735.5	0.55	880	29735.0	29735.8	0.00271	0.00383	6	13999.5	114.05	902	29735.2	29735.7	0.00169	0.00239	6	13999.6
$P_{600,1680u}$	44836.4	0.79	934	44836.2	44836.7	0.00111	0.00117	5	20999.5	109.78	802	44836.3	44836.4	0.00022	0.00069	9	20999.7
K_{20w}	618.9	0.11	720	618.9	619.0	0.01701	0.05379	9	379.5	17.68	578	618.9	618.9	0	0	10	379.6
K_{40u}	3669.3	0.23	880	3669.3	3669.5	0.00550	0.01159	8	778.9	51.85	789	3669.3	3669.4	0.00279	0.00882	9	779.3
K_{60u}	5673.5	0.58	1164	5673.3	5673.8	0.00886	0.01250	6	1178.7	110.65	860	5673.5	5673.6	0.00177	0.00559	9	1178.3
K_{100u4}	9690	2.15	1889	9690	9691	0.01032	-	-	1977	1239	1779	9690	9691	0.01032	-	-	1980
K_{180u4}	17723	7.76	2386	17722	17724	0.01128	-	-	3556	1048	1215	17723	17723	0	-	-	3572

Table 5.13: Results of KCMST-LD$_{\text{VA}}$ using either KP or E-kKP on selected graphs.

5.5 Comparison to Previous Results

Now we will compare the new results of the Lagrangian decomposition using the VA with all results achieved before that are known and available to the author.
To be able to compare our results, especially the %-*gap*, with those presented in [42] we must define some more attributes, since they compared the bounds derived through Lagrangian relaxation with the optimal solution found by their B&B method. They gave the relative error of the Lagrangian upper bound (Err_U), of the Lagrangian lower bound (Err_L^b) and the 2-opt lower bound ($Err_L^\#$). We will formulate two more measures using these:

$$\%\text{-}gap^b = \frac{Err_U + Err_L^b}{100 - Err_L^b} \cdot 100\% \tag{5.1}$$

which is the relative difference between $\underline{p^b}$ and $\overline{p}$, the Lagrangian lower and upper bound, respectively, and

$$\%\text{-}gap^\# = \frac{Err_U + Err_L^\#}{100 - Err_L^\#} \cdot 100\% \tag{5.2}$$

which is the relative difference between the 2-opt lower bound $\underline{p^\#}$ and again the Lagrangian upper bound $\overline{p}$ (for definitions of Err_U, Err_L^b, $Err_L^\#$, $\underline{p^b}$, $\underline{p^\#}$ and $\overline{p}$ see [42]).
The first gap, %-gap^b, can be compared with our %-*gap* when the local search KLS is not used, whereas %-$gap^\#$ is comparable with our gap when using KLS.
As they gave no CPU-time of their Lagrangian relaxation we will list the times given for the B&B method (now denoted by $t[s]$). Additionally we will state how many out of ten instances were solved by them within 2000 CPU seconds (now denoted by *Opt*). Since they used a IBM RS/6000 44P Model 270 workstation on a different set of randomly generated instances, only a comparison of the order of magnitude of the values is possible.
We will also give the best results obtained by the B&C method presented in [33], stating the average running time and the number of optimal solutions out of ten, again solved within 2000 CPU seconds. The test set 1, presented in this work, is an extension of the instances used in [33], so the same generated instances were used for testing. Additionally the computing environment was the same as for the CPLEX tests done in the previous section, thus being comparable to the newest results achieved in this thesis.

The results on plane graphs are opposed in Table 5.14 and for complete graphs in Table 5.15. Now we will look at the results of every correlation type of both graph types.
Since our developed algorithm is for some instances unable to derive the optimal upper bound, the optimality of many actually optimal solutions cannot be proven. This occurs especially for uncorrelated plane graphs up to $P_{600,1680u}$. For these graphs the B&B algorithm of Yamada and colleagues is suited best, but for the graphs $P_{800,2240u}$ and $P_{1000,2800u}$ our algorithm outperforms the other two, being extremely fast and finding all optimal solutions. For the graphs $P_{50,127w}$ and $P_{100,260w}$ we have again seemingly worse results due to non-optimal upper bounds, though for the former graph are actually found 9 and for the latter even all 10 optimal solutions. All in all KCMST-$LD_{VA\&KLS}$ performs best for

the remaining weakly correlated plane graphs, as well as clearly for all strongly correlated plane graphs. The CPU-times of the compared methods on plane graphs are shown in Figure 5.12.
Though for all instances of uncorrelated complete graphs but one (K_{60u5}) the optimal solution was found by KCMST-$LD_{VA\&KLS}$, they can again not always be proven to be optimal. For these rather rare cases the B&C algorithm in [33] is to prefer, otherwise KCMST-$LD_{VA\&KLS}$ yields good results. In case of weakly correlated complete graphs only two solutions are not optimal, but one of K_{20w} merely lacks the corresponding optimal upper bound and K_{120w9} was found out to be solved to optimality in 7 out of 10 testruns. When only considering the number of provably optimal solutions then the B&C method is the winner, whereas when also taking the running time into account then KCMST-$LD_{VA\&KLS}$ is a good alternative.
Finally when looking at the results on the strongly correlated graphs our new algorithm clearly outperforms the other two, like for strongly correlated plane graphs.
When comparing the relative gaps %-gap^b and %-$gap^{\#}$ of the Lagrangian relaxation used in [42] with the values of %-gap of KCMST-LD_{VA} and KCMST-$LD_{VA\&KLS}$, respectively, clearly reveals an advantage of our method, producing a much tighter gap, in fact being several orders of magnitude smaller. The second great advantage over the B&B as well as the B&C algorithm is the small running time, which is especially apparent as the graphs are getting bigger, as can be seen in Figure 5.13. The only disadvantage is the examined fact that the derived upper bounds are in some cases not optimal, therefore reducing the number of possible provably optimal solutions.
However, our new algorithm is likely the only one of these which can be (and in fact was) successfully applied to the instances of test set 2, being to date unrivalled for such large graphs.

Instance	**Yamada et al.** [42]				**Pirkwieser** [33]		**KCMST-LD$_{\mathbf{VA}}$**			**KCMST-LD$_{\mathbf{VA\&KLS}}$**		
	LR		B&B		B&C							
$graph_{corr}$	%-gap^{b}	%-$gap^{\#}$	$t[s]$	Opt	$t[s]$	Opt	%-gap	$t[s]$	Opt	%-gap	$t[s]$	Opt
$P_{50,127u}$	0.9482	0.4541	0.43	10	0.14	10	0.06256	0.19	3	0.04578	0.30	3
$P_{100,260u}$	0.5866	0.2689	1.78	10	0.72	10	0.00676	0.17	7	0.00676	0.37	7
$P_{200,560u}$	0.4116	0.1879	5.46	10	5.06	10	0.00398	0.31	6	0.00268	0.55	7
$P_{400,1120u}$	0.1283	0.0704	24.44	10	68.04	10	0.00271	0.55	6	0.00236	1.15	6
$P_{600,1680u}$	0.1212	0.0541	75.25	10	286.51	10	0.00111	0.79	5	0.00067	1.52	7
$P_{800,2240u}$	0.2962	0.1249	466.37	10	773.67	7	0	0.79	10	0	1.59	10
$P_{1000,2800u}$	0.1660	0.0733	592.77	10	1875.37	2	0	0.99	10	0	2.08	10
$P_{50,127w}$	4.3720	1.2433	0.81	10	0.44	10	0.05280	0.15	6	0.03357	0.23	6
$P_{100,260w}$	2.9264	0.6037	2.71	10	1.29	10	0.00967	0.17	7	0.00724	0.36	7
$P_{200,560w}$	1.0640	0.2663	13.11	10	5.75	10	0.00119	0.28	9	0	0.36	10
$P_{400,1120w}$	0.8188	0.1839	47.15	10	40.84	10	0.00358	0.49	7	0	0.77	10
$P_{600,1680w}$	0.8240	0.1676	371.84	8	301.06	10	0.00040	0.65	9	0.00040	1.29	9
$P_{800,2240w}$	0.4257	0.1038	509.22	5	976.50	8	0.00089	0.92	8	0	1.76	10
$P_{50,127s}$	10.2825	0.1610	2.84	10	0.09	10	0.04392	0.16	5	0	0.12	10
$P_{100,260s}$	19.8980	0.2656	405.45	8	0.28	10	0.00972	0.23	6	0	0.18	10
$P_{200,560s}$	-	-	-	-	3.13	10	0.00849	0.35	6	0	0.32	10
$P_{400,1120s}$	-	-	-	-	39.67	10	0.00061	0.45	9	0	0.58	10
$P_{600,1680s}$	-	-	-	-	250.85	10	0.00444	0.85	5	0	0.88	10

Table 5.14: Comparing results on *plane* graphs (see Table 5.8) with previous ones.

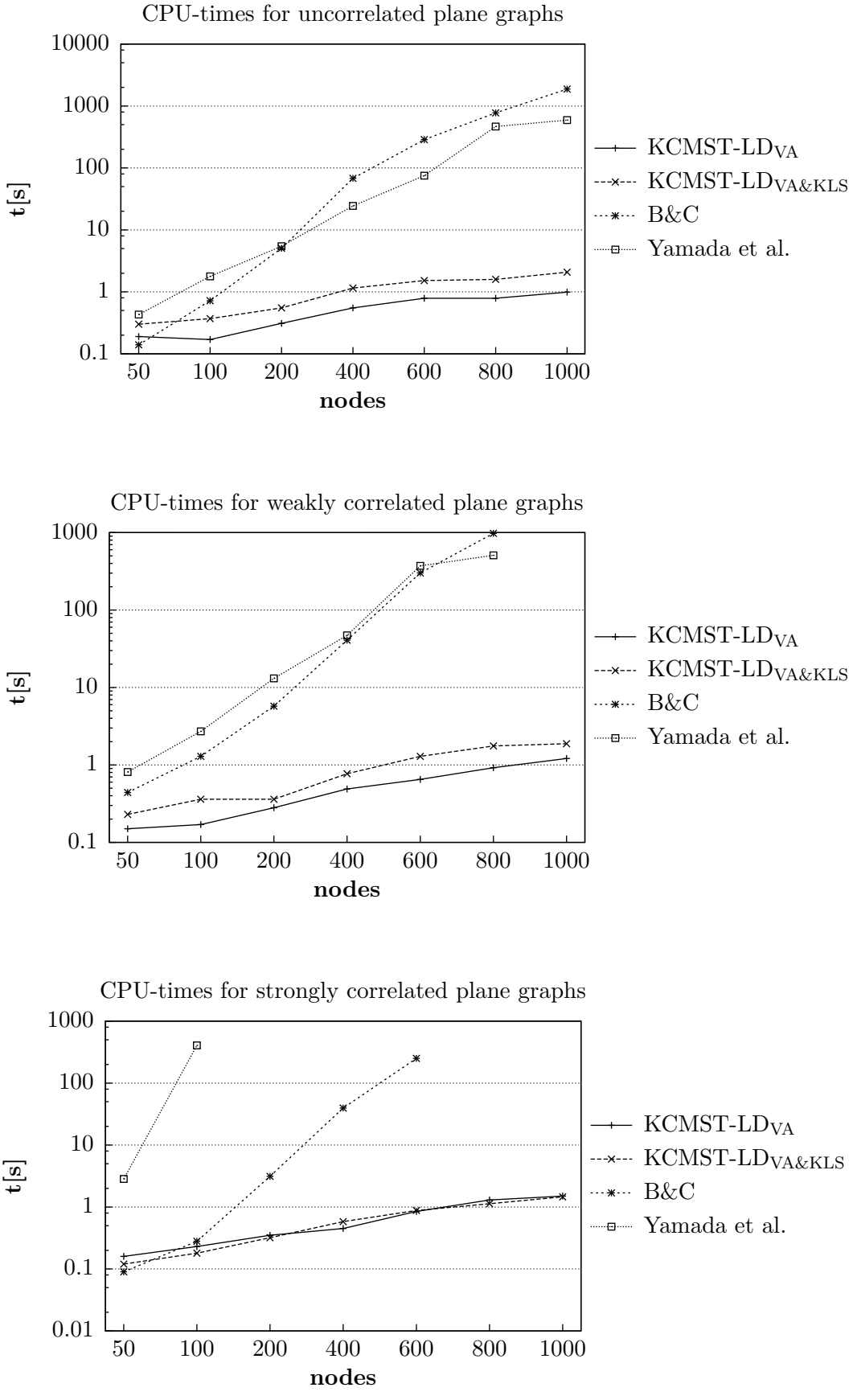


Figure 5.12: CPU-times of the different methods for plane graphs.

Instance	**Yamada et al.** [42]				**Pirkwieser** [33]		**KCMST-LD$_{\mathbf{VA}}$**			**KCMST-LD$_{\mathbf{VA\&KLS}}$**		
	LR		B&B		B&C							
$graph_{corr}$	%-gap^b	%-$gap^{\#}$	$t[s]$	Opt	$t[s]$	Opt	%-gap	$t[s]$	Opt	%-gap	$t[s]$	Opt
K_{40u}	0.2509	0.1061	0.87	10	0.44	10	0.00550	0.23	8	0.00550	0.28	8
K_{60u}	0.3901	0.1074	1.89	10	2.42	10	0.00886	0.58	6	0.00710	0.72	6
K_{80u}	0.2727	0.1303	6.54	10	2.67	10	0	0.60	10	0	0.69	10
K_{100u}	0.1488	0.0433	12.48	10	7.43	10	0.00103	1.07	9	0.00103	1.27	9
K_{120u}	0.1223	0.0427	23.69	10	11.82	10	0	1.37	10	0	1.65	10
K_{140u}	0.0561	0.0226	60.95	10	20.48	10	0	2.08	10	0	2.38	10
K_{160u}	0.0897	0.0388	476.26	10	22.63	10	0	2.88	10	0	3.19	10
K_{180u}	0.1011	0.0452	636.54	10	49.29	10	0.00113	4.31	9	0.00056	4.95	9
K_{200u}	0.0405	0.0172	375.26	10	84.70	10	0	5.55	10	0	6.11	10
K_{20w}	6.1869	0.9917	0.25	10	0.07	10	0.01701	0.11	9	0.01701	0.12	9
K_{40w}	4.2625	0.5203	1.17	10	0.93	10	0.00755	0.24	9	0	0.19	10
K_{60w}	5.7005	0.5292	6.09	10	3.64	10	0.01987	0.51	8	0	0.40	10
K_{80w}	4.9704	0.3436	38.15	10	4.73	10	0.00368	0.81	9	0	0.67	10
K_{100w}	2.4133	0.1729	377.61	8	14.02	10	0.00292	1.10	9	0	1.02	10
K_{120w}	3.7977	0.2066	451.06	8	30.83	10	0.02669	2.78	3	0.00243	1.65	9
K_{140w}	-	-	-	-	39.23	10	0.02073	3.29	5	0	2.29	10
K_{160w}	-	-	-	-	62.65	10	0.01810	4.44	5	0	3.27	10
K_{20s}	22.1222	0.3791	0.53	10	0.04	10	0.05689	0.22	7	0	0.09	10
K_{30s}	17.0329	0.3222	99.63	10	0.11	10	0.03712	0.31	7	0	0.16	10
K_{40s}	9.4927	0.1377	226.30	6	0.29	10	0.01838	0.34	9	0	0.28	10
K_{60s}	-	-	-	-	2.49	10	0.11539	1.72	4	0	0.57	10
K_{80s}	-	-	-	-	8.94	10	0.13146	3.94	2	0	1.16	10
K_{100s}	-	-	-	-	74.82	10	0.13784	2.46	6	0	2.05	10

Table 5.15: Comparing results on *complete* graphs (see Tables 5.10 and 5.11) with previous ones.

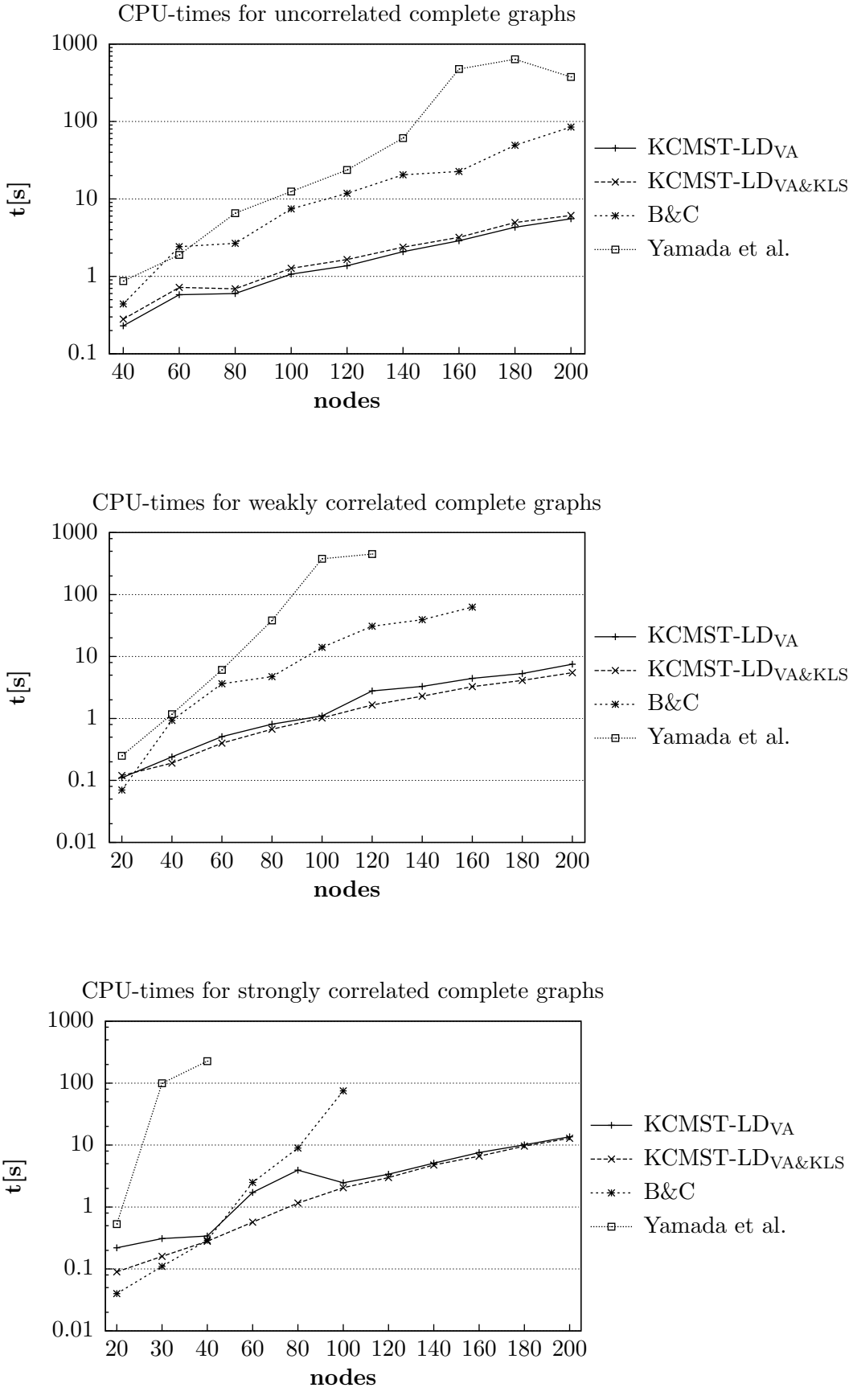


Figure 5.13: CPU-times of the different methods for complete graphs.

6 Hybrid Lagrangian Evolutionary Algorithm for the KCMST Problem

This chapter introduces a suitable coding for spanning trees, an EA for the KCMST problem, building upon this coding, and presents two ways to combine the EA with the Lagrangian algorithms presented in Chapter 4, resulting in a Hybrid Lagrangian EA.

6.1 The Edge-Set Coding Scheme

The kind of genotype, i.e the representation of a solution, is probably the most important decision when designing an EA. It is a decisive factor regarding the performance of the whole algorithm, furthermore all operators must be laid out accordingly. An adequate representation for spanning trees is a so called edge-set introduced in Julstrom and Raidl [24], see Figure 6.1 for an example. This *coding scheme* was shown by them to have favorable properties compared to others used for this kind of problems so far. They also

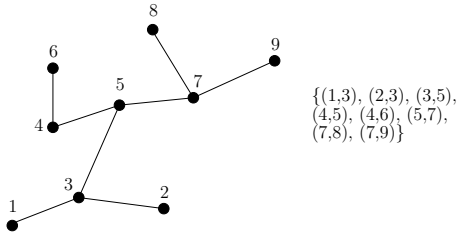


Figure 6.1: Example of an edge-set representation of a spanning tree.

described variations of Kruskal's and Prim's algorithm as well as a random-walk algorithm for the purpose of generating random spanning trees (RSTs) for a given graph. These algorithms can be used to create an initial diversified population. Further they presented appropriate operators, among them the recombination of two spanning trees T_1 and T_2 represented as edge-sets. The idea is to merge both edge-sets and generate a spanning tree using again one of the RST algorithms, i.e. $T_3 = RST(T_1 \cup T_2)$. Using only parental edges ensures strong heritability, but the constraints of the problem at hand may enforce to use other edges as well, i.e. one of $E - (T_1 \cup T_2)$. They mentioned the possibility to favor common edges, i.e. which appear in both parents, by always including them in the offspring.
The two variants of mutation are either mutation by insertion, i.e. include a new edge and remove another from the induced cycle (thus termed insertion-before-deletion), or

mutation by deletion, i.e. removing an edge, determine the separated components and reconnect them by a new edge (called deletion-before-insertion).
Additionally they described how to include heuristics in the EA based on probabilities according to the edge-costs, thus probably improving its performance. We will present the possible applications proposed by them:

- One can favor low-cost edges in the initialization phase by considering them first in the RST algorithm, though this variant is only possible in case of Kruskal's algorithm. But one has to ensure diversified solutions. This method is more detailed in the next section.
- When applying recombination one can select the (remaining) edges to be included in the offspring according to their costs. This can be done via a tournament selection or even selecting them in a greedy fashion.
- Finally the edge-costs can also be utilized by the insertion-before-deletion mutation. There the unused edges to be inserted can be determined either via a tournament or by using their cost-based rank and identify a rank by sampling a suitable random variable, thus applying a rank-based selection. They concluded from experimental results that the latter variant is more effective than a tournament. It will be explained in the next section.

They also developed an EA for the degree-constrained minimum spanning tree problem (d-MST), thereby including the above mentioned heuristics and further designing the operators to generate only feasible solutions.

6.2 Evolutionary Algorithm for the KCMST Problem

Our EA, henceforth called KCMST-EA, is based on that described in [24] for the d-MST problem, though adapted to the KCMST problem and extended where necessary. Since we deal with a maximization problem we introduce edge-profits $p'(e) > 0$ (not to be confused with the profit values $p(e)$ given by a problem instance) instead of the edge-costs mentioned before. How these edge-profits are determined will be explained later, for now we just assume that they exist and that preferable edges of an instance have a high profit and vice versa.

We apply a steady state EA with elitism, i.e. in each iteration two parents are selected and recombined to generate one offspring which replaces in our case the worst chromosome of the population, thereby always keeping the best chromosome in the population. To facilitate diversity no duplicate solutions are allowed. The fitness of a chromosome is simply its objective value, i.e. $\sum_{e \in T} p(e)$. Now the initialization phase as well as the operators of KCMST-EA are described.

Initialization: We will only use the Kruskal-RST algorithm, since we want to use a profit-based heuristic. By applying the suggestion of Julstrom and Raidl [24] adapted to

our problem, we start with sorting the edges according to decreasing profit. The first tree is built using this edge sequence, then the number of permuted edges is increased with each tree using the formula

$$k = \alpha(i-1)n/P, \tag{6.1}$$

where P is the population size, $i = 1, \ldots, P$ is the index of the next tree and α (set by the program parameter *heuinik*) controls the degree of bias towards high-profit edges. The higher α the more diversified is the initial population. Thus the determined amount of the most profitable edges are permuted before running the Kruskal-RST algorithm. If the generated tree T is infeasible because it does not satisfy the weight constraint, thus $w(T) > c$, it is randomly repaired. Thereby edges of the set $E-T$, i.e. the remaining edges of the graph are selected at random to be included and it is tried to remove another edge from the induced cycle which has more weight assigned, to derive a tree weighing less. In case two ore more edges in the cycle have the same maximum weight, those with minimal $p(e)$ is chosen to be removed, thus favoring edges with higher profit. This is repeated until the tree is feasible, which is guaranteed to happen.

Recombination: Here we use again the Kruskal-RST algorithm, because of the simpler data structures applied. The recombination works like following. Given two parent solutions T_1 and T_2, one can decide whether to include the edges $T_1 \cap T_2$ appearing in both parents immediately in the offspring solution T_3 or not, by setting the binary parameter $prefib$. The remaining edges, either $(T_1 \cup T_2) - (T_1 \cap T_2)$ or $T_1 \cup T_2$ in case no common edges were included, are examined in a specific order, defined by the parameter *crotype*:

0: In random order, favoring no edges over others, thus performing a random crossover.

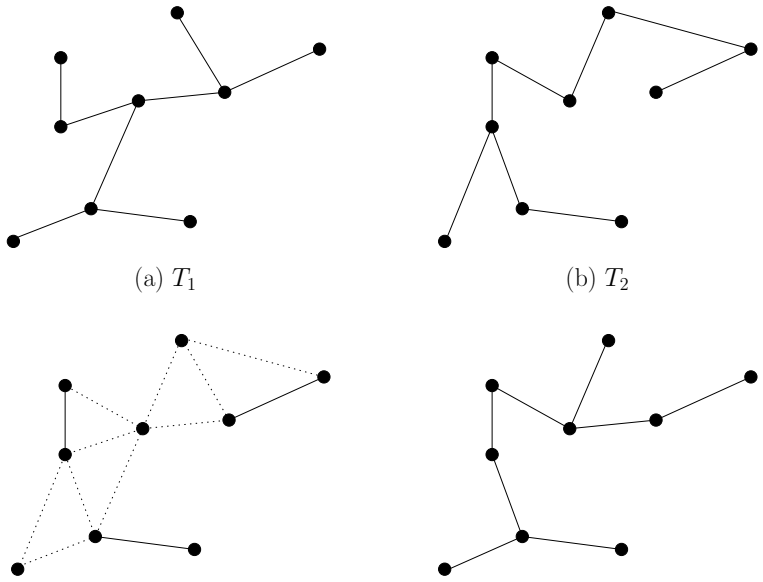
1: In a greedy order, examining them according to decreasing profit $p'(e)$.

2: Via a tournament with desired size *heucrok*, preferring edges with higher profit $p'(e)$.

An example for a recombination where all common parental edges are included in the offspring is illustrated in Figure 6.2.

Whereas it is guaranteed that a feasible solution can be found using exclusively the merged edges of both parents (since both were feasible at least two feasible solutions can be derived of the set), it is not assured that the method presented so far accomplishes this. It may happen that the generated tree T_3 is infeasible, i.e. it does not satisfy the weight constraint, and thus $w(T_3) > c$. If this is the case the solution is randomly repaired like described for the initialization phase, but this time selecting the edges only of the set $(T_1 \cup T_2) - T_3$ at random. So it is ensured that the offspring consists only of parental edges, yielding a strong heritability.

Mutation: Using the mutation operator should bring new information into a solution and consequently into the whole population, thus diversifying the search to a certain degree. We will only apply the insertion-before-deletion mutation, but with different selection-schemes regarding the edge(s) to be included, given by the parameter *heumut*:



(a) T_1 (b) T_2

(c) $T_1 \cap T_2$ among $T_1 \cup T_2$ (d) $T_3 = RST(T_1 \cup T_2)$

Figure 6.2: Given the parent solutions (a) T_1 and (b) T_2, a offspring (d) T_3 is generated of the merged edges $T_1 \cup T_2$ using the Kruskal-RST algorithm, thereby including the common edges (c) $T_1 \cap T_2$.

0: Choose them with a uniform probability favoring none of them.

1: Use a rank-based selection, by considering the edges in sorted order according to decreasing profit $p'(e)$, thus having a profit-based ranking, and sampling a random variable

$$R = \lfloor |\mathcal{N}(0, \beta n)| \rfloor \bmod m + 1 \tag{6.2}$$

where $\mathcal{N}(0, \beta n)$ is a normally distributed random variable with mean 0 and standard deviation βn. It holds that $R \in [1, m]$ with $m = |E|$. This scheme's bias towards high-profit edges is adjustable via the parameter β, making the probability distribution more uniform with increasing size of β, which is set by the program parameter *heumutk*.

When an edge e is selected for inclusion, another edge e' from the induced cycle C must be removed from the tree T. This edge is determined in the following way: examine the edges of $C - \{e\}$ in random order, thus using no heuristic, and check if the weight constraint would still be satisfied after exchanging both edges, which is the case if

$$w(e') \geq w(T) + w(e) - c. \tag{6.3}$$

If no such edge e' is found, the mutation fails and T is left unchanged, otherwise replace e' with e, thereby using the first edge satisfying (6.3).

Local Search: Contrary to the mutation operator the local search leads to an intensification of KCMST-EA's search. The local search applied is in general equal to KLS (Algorithm 10 described in Section 4.4), it either chooses the edges randomly ($klsTypeEA = 0$) or greedily by examining them in the sequence according to decreasing profit $p'(e)$ ($klsTypeEA = 1$).

KCMST-EA implements the schema of the basic EA depicted in Algorithm 2 in Section 2.3 using the operators just described, in addition of probably applying a local search operator after mutation.
If having no other information regarding an edge's desirability, the often referred edge-profits are determined by the ratio of profit to weight given by an instance:

$$\forall e \in E : p'(e) = p(e)/w(e) \tag{6.4}$$

as was already used for generating the initial feasible solution for the Lagrangian algorithms.

6.2.1 Experimental Results of the KCMST-EA

We ran tests to investigate if the heuristics based on these straightforward profit values presented above are of benefit. Thereby KCMST-EA was applied with a population of size 100 ($popsize = 100$) and run for 10^5 iterations ($tgen = 100000$) either without using any profit-based heuristics, i.e. randomly permuting all edges for every generated tree when initializing ($heuinik = 0$), using random-crossover ($crotype = 0$) and selecting the edges in case of mutation at random ($heumut = 0$ and $pmut = -1$), or utilizing the profits wherever applicable. In the latter case, denoted by KCMST-EA$_{\mathrm{H}}$ we prefer profitable edges when creating the initial population ($heuinik = 1.5$), determine the order of examined parental edges via a binary tournament selection ($crotype = 2$ and $heucrok = 2$), select the edges for mutation using the rank-based selection ($heumut = 1$ with $heumutk = 1.5$ and $pmut = -1$) and apply the greedy variant of KLS ($klsTypeEA = 1$ with $klsRetriesEA = 50$) with a probability of $plocim = 0.2$. These parameters, except of the population size and the local search, were also used in [24]. Since the inclusion of common edges in the offspring does not make use of any heuristic it is applied in both runs ($prefib = 1$).

The results on some plane and complete graphs are presented in Table 6.1. The fitness (or objective value) of the best chromosome found, averaged over the 10 instances per graph, is denoted by *sol*. The mean difference of *sol* between KCMST-EA and KCMST-EA$_{\mathrm{H}}$ is given by δ_{sol}. Further a paired t-test using a 95% confidence interval was made to investigate if this difference is significant, giving the *p-value* which states the probability that this specific difference occurred assuming the null hypothesis, i.e. both means are equal, would be true. The results of KCMST-LD refer to those of KCMST-LD$_{\mathrm{VA\&KLS}}$ presented earlier, except for uncorrelated complete graphs, where the results of KCMST-LD$_{\mathrm{VA}}$ are given.

The variant utilizing the profit-based heuristics produced only for uncorrelated and weakly

Instance	**KCMST-LD**		**KCMST-EA**		**KCMST-EA$_{\mathbf{H}}$**		δ_{sol}	**t-Test**
$graph_{corr}$	LB	$t[s]$	sol	$t[s]$	sol	$t[s]$		p-$value$
$P_{100,260u}$	7222.9	0.38	7216.4	14.71	7222.0	21.11	0.08%	2.29%
$P_{100,260w}$	4167.9	0.36	4162.8	15.20	4165.5	21.91	0.06%	1.76%
$P_{100,260s}$	4115.5	0.18	4113.2	18.18	4111.9	25.88	-0.03%	0.63%
$P_{200,560u}$	14896.9	0.55	14851.7	28.77	14884.1	39.45	0.22%	2.07%
$P_{200,560w}$	8432	0.36	8402.0	30.32	8409.0	41.24	0.08%	11.69%
$P_{200,560s}$	8244.3	0.32	8236.2	37.25	8231.4	49.18	-0.06%	0.00%
$P_{400,1120u}$	29735.1	1.15	29594.7	61.09	29671.3	78.37	0.26%	0.13%
$P_{400,1120w}$	16794.9	0.77	16682.5	64.26	16718.7	81.23	0.22%	0.14%
$P_{400,1120s}$	16500.3	0.58	16476.6	71.47	16469.1	98.19	-0.04%	0.74%
K_{100u}	9680.0	1.07	9672.9	17.55	9508.3	21.55	-1.70%	0.00%
K_{100w}	3421.4	1.02	3383.9	16.75	3179.2	22.05	-6.04%	0.00%
K_{100s}	2771.9	2.05	2760.3	23.21	2707.4	25.52	-1.91%	0.00%
K_{200u}	19739.4	5.55	19551.6	32.60	19302.2	46.89	-1.27%	0.00%
K_{200w}	6928.0	5.49	6755.9	32.55	6328.7	43.01	-6.32%	0.00%
K_{200s}	5572.0	12.86	5542.5	41.61	5431.7	49.99	-1.99%	0.00%
K_{300u}	29770.6	19.20	29161.0	46.24	29114.4	76.78	-0.16%	14.27%
K_{300w}	10734.0	20.86	10349.6	48.79	9778.4	68.08	-5.52%	0.00%
K_{300s}	8372	48.64	8321.2	77.25	8153.6	78.89	-2.01%	0.00%

Table 6.1: Results of KCMST-EA with and w/o heuristics based on profits of (6.4).

correlated plane graphs provably better solutions. For all other graphs it is better not to use any heuristics based on these profits. This was especially apparent when generating the initial population, where the average fitness for weakly and strongly correlated complete graphs is doubled (or even more) when using no heuristic. Thus we can conclude that these straightforward profit values determined by (6.4) are rather misleading and in generally not to prefer. Unsurprisingly tests using the greedy edge selection in recombination ($crotype = 1$) showed consistently even worse results. After comparing the better solutions of either KCMST-EA or KCMST-EA$_{\mathrm{H}}$ with the optimal ones, it can be stated that the average relative difference is mostly less than 1%, being in general even smaller for all plane graphs and a bit higher for weakly correlated complete graphs (up to 3.58%). Although this might be a satisfying result for some applications, the Lagrangian algorithm clearly generates better solutions.
Finally when looking at the running time, it reveals the disadvantageous time consumption of KCMST-EA, especially for plane graphs. This is because the algorithm's performance highly depends on the size of the used edge-set. For this reason it is far more suitable for complete graphs which have a relatively small edge-set contrary to plane or even maximal plane graphs with several hundreds or thousands of nodes and accordingly huge spanning trees.

6.3 Combining KCMST-LD and KCMST-EA to a Hybrid Lagrangian EA

Having seen that KCMST-EA is inferior to KCMST-LD, we will now investigate if it is beneficial to combine them. Thereby we will consider two *collaborative combinations*, running both algorithms either in *sequential* order starting with KCMST-LD or *intertwined*, i.e. they are consecutively executed, each one for a specified amount of iterations. The resulting algorithm is a Hybrid Lagrangian EA, which will be denoted by HLEA. Depending on the type of combination the algorithms exchange useful information. All possible exchanges are illustrated in Figure 6.3 and will be described in the following:

- **Best solution:** If KCMST-LD finds a solution with highest objective value so far, i.e. a new best solution, it can be transferred to KCMST-EA by including it as a chromosome using the edge-set representation. So the latter algorithm can use this solution to recombine it with others.

- **Upper bound:** A derived (better) upper bound $\lfloor UB \rfloor$ of KCMST-LD can be used to update the objective value for termination *tobj* used by KCMST-EA, which terminates as soon as the fitness of the best chromosome is equal to or greater than this value. Provided that the upper bound is optimal enables KCMST-EA to stop if the best solution is found, thus prevents unnecessary iterations.

- **Lower bound:** If KCMST-EA generates a chromosome with highest fitness up to now, this fitness value can be used to update the best known lower bound of KCMST-

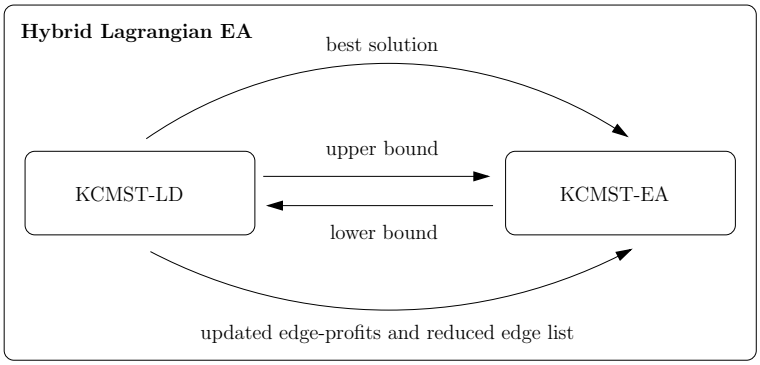


Figure 6.3: Possible information exchange between Lagrangian and Evolutionary Algorithms.

LD. Like before this update can lead to an earlier termination. It can only be applied in case of intertwined execution.

- **Edge-profits:** Since the profits determined by (6.4) were shown to be rather misleading, especially for complete graphs, the edge-profits used by KCMST-EA are updated on the basis of the Lagrangian multipliers:

$$p'_i = p_i - \lambda_i. \tag{6.5}$$

  These values were already successfully used for the local search KLS in case of complete graphs.

- **Reduced edge list:** We build a set of edges EL while running KCMST-LD by merging the edges of all derived feasible solutions:

$$EL = \bigcup_{e \in T} \text{feasible solution } T \text{ derived by KCMST-LD} \tag{6.6}$$

  and subsequently restrict KCMST-EA to work on this reduced set. This might help to intensify the search in the most promising regions of the search space, though this depends on the quality of the merged solutions. At the best EL contains all edges necessary to build an optimal solution. Using this procedure is restricted to sequential execution.

  This utilization of primal information produced by Lagrangian decomposition was also presented in [21].

In the **sequential execution** KCMST-LD is run first, possibly followed by KCMST-EA if the profit of the actual best solution does not match the latest upper bound. The scheme of the sequential HLEA is shown in Algorithm 11. In case KCMST-EA is applied, then prior to generating the initial population the edge-profits are set using the latest Lagrangian multipliers, which should lead to a strengthening of all profit-based heuristics.

Further the set of edges which are considered is possibly reduced. Additionally *tobj* is set to the best upper bound and the best solution of KCMST-LD is integrated as a chromosome into the population of KCMST-EA. After this KCMST-EA is run until it either finds the optimal solution or terminates otherwise, e.g. because a maximal number of iterations is reached.

Algorithm 11: Sequential Hybrid Lagrangian EA

```
initialize KCMST-LD;
run KCMST-LD;
if LB = ⌊UB⌋ then // provably optimal solution found
    return;
for i = 1 to m do
    p'_i = p_i − λ_i; // set edge-profits of KCMST-EA
tobj = ⌊UB⌋; // set objective value for termination of KCMST-EA
possibly reduce edge list E to EL;
initialize KCMST-EA;
integrate best solution of KCMST-LD as chromosome into population of KCMST-EA;
run KCMST-EA;
```

Applying an **intertwined execution** is straightforward, too. The Intertwined HLEA is depicted in Algorithm 12. As already mentioned before the two algorithms are run consecutively and exchange information accordingly. This includes integrating a new best solution of KCMST-LD into the population of KCMST-EA and updating *tobj* with the actual best upper bound. Further the edge-profits of KCMST-EA are updated using the actual Lagrangian multipliers after each run of KCMST-LD. Every time KCMST-EA finds a new best solution (i.e. chromosome), its objective value (i.e. fitness) is used to set the best known lower bound utilized by KCMST-LD.

The intertwined HLEA is terminated if either of the two algorithms finds a provably optimal solution. In case KCMST-LD terminates because its maximal amount of consecutively non-improving steps is reached, KCMST-EA is run one last time for $iterEA$ iterations.

6.3.1 Experimental Results of the Sequential-HLEA

Since for nearly all complete graphs of test set 1 and 2, except of two instances, optimal (though not always proven optimal) solutions were found by KCMST-LD$_{\text{VA\&KLS}}$ (see previous chapter), we will not consider to run the Sequential-HLEA on them. Instead we focus on the maximal plane graphs. A comparison of results obtained so far indicated that the upper bounds are optimal contrary to some of the solutions. Further it was found that the best solution of an instance (whether optimal or not) was in most cases exclusively built

Algorithm 12: Intertwined Hybrid Lagrangian EA

```
initialize KCMST-LD and KCMST-EA;
repeat
    run KCMST-LD for iterLD iterations;
    if LB = ⌊UB⌋ then // provably optimal solution found
        break;
    if KCMST-LD found new best solution then
        integrate solution as chromosome into population of KCMST-EA;
    if ⌊UB⌋ < tobj then
        tobj = ⌊UB⌋; // update objective value for termination of KCMST-EA
    for i = 1 to m do
        p'_i = p_i − λ_i; // update edge-profits of KCMST-EA
    run KCMST-EA for iterEA iterations;
    if KCMST-EA found new best solution then
        update lower bound of KCMST-LD with fitness value;
until termination condition ;
```

of edges which occurred in previous solutions, thus the precondition for the application of edge list reduction is given.

The parameters for KCMST-LD are those of KCMST-LD$_{\text{VA\&KLS}}$ as in previous tests on these graphs, i.e. $maxSteps = 300$ (only for P_{10000s} setting $maxSteps = 500$), $useLB\text{-}fromKLS = 0$, $klsType = 0$, $klsRetries = 100$, and $klsMinGap = 0.995$. KCMST-EA will be run with all heuristics and KLS, thus setting $heuinik = 1.5$, $crotype = 2$, $prefib = 1$, $pmut = -1$, $heumut = 1$, $plocim = 0.2$, $klsTypeEA = 0$ and $klsRetriesEA = 50$. The population size is again 100 and as termination condition we bound the maximal number of iterations to 5000 ($tcgen = 5000$) for graphs up to 8000 nodes and to 10000 ($tcgen = 10000$) for the two remaining larger graphs. Finally the parameters of HLEA for sequential execution and reduced edge list for KCMST-EA are $execOrder = 0$ and $reducedEL = 1$.

The amount of possible edge list reduction, i.e. $(|E| - |EL|)/|E| * 100\%$, is stated by red. The number of KCMST-EA iterations, averaged only over those instances where it was applied, is given by $iter_{\text{EA}}$ and Opt_{EA} denotes the number of optimal solutions (among Opt) which were found by KCMST-EA. The remaining attributes were already introduced in Table 5.1. The results are presented in Table 6.2. We will compare them with those of KCMST-LD$_{\text{VA\&KLS}}$ given in Table 5.9.

Applying the Sequential-HLEA instead of KCMST-LD improves most notably the results on weakly correlated graphs, by now solving all instances to provable optimality. Moreover the highest possible edge list reduction, about 45%, can be noticed on these graphs, seemingly intensifying the search of the algorithm as intended. The results on uncorrelated graphs are only improved to a small degree, but they have already been very satisfying before. The execution of KCMST-EA after KCMST-LD rather seems to effectively com-

pensate for the unsteady success of KLS applied in KCMST-LD. The uncorrelated graphs can be reduced about 39%. Nearly no improvement is obtained on the strongly correlated graphs, which is first of all because KCMST-LD with KLS applied solved almost all of them to optimality. In the rare case when this was not possible, KCMST-EA found the optimal solution of half of these instances (precisely for 3 out of 5). KCMST-EA does not seem to be very succesful on strongly correlated maximal plane graphs, at least not when it comes to the hard task of finding the optimal solution. This is probably mainly due to the lowest possible edge list reduction, which is only about 23%.

Instance	**Sequential-HLEA**										
$graph_{corr}$	$t[s]$	$iter$	red	$iter_{\mathrm{EA}}$	LB	UB	$\%\text{-}gap$	$\sigma_{\%\text{-}gap}$	Opt	Opt_{EA}	w_{LB}
P_{2000u}	4.34	816	38%	2188	147799.6	147799.6	0	0	10	1	69999.9
P_{2000w}	6.29	856	44%	2001	85570.8	85570.8	0	0	10	3	69999.9
P_{2000s}	3.33	816	20%	0	82523.3	82523.3	0	0	10	0	70000.0
P_{4000u}	12.42	853	39%	5000	294872.0	294872.1	0.00003	0.00009	9	0	139999.7
P_{4000w}	12.93	985	43%	1283	170958.1	170958.1	0	0	10	3	139999.9
P_{4000s}	7.53	887	23%	0	165051.5	165051.5	0	0	10	0	140000.0
P_{6000u}	23.45	959	39%	2674	441978.1	441978.1	0	0	10	2	210000.0
P_{6000w}	20.93	980	45%	1130	256318.4	256318.4	0	0	10	3	210000.0
P_{6000s}	15.62	937	25%	1325	247592.2	247592.2	0	0	10	1	210000.0
P_{8000u}	17.88	892	39%	0	589447.1	589447.1	0	0	10	0	279999.8
P_{8000w}	26.97	919	46%	3503	341904.4	341904.4	0	0	10	1	280000.0
P_{8000s}	39.37	922	23%	3968	330122.0	330122.1	0.00003	0.00009	9	1	280000.0
P_{10000u}	56.66	1029	39%	1877	737450.7	737450.7	0	0	10	4	349999.9
P_{10000w}	61.62	1048	44%	1681	427407.2	427407.2	0	0	10	5	349999.9
P_{10000s}	26.82	1019	23%	0	412643.6	412643.6	0	0	10	0	350000.0
P_{12000u}	55.54	1008	39%	1468	885117.8	885117.8	0	0	10	3	419999.8
P_{12000w}	77.05	1037	45%	2147	512987.3	512987.3	0	0	10	4	419999.9
P_{12000s}	141.99	1044	23%	8225	495169.6	495169.7	0.00002	0.00006	9	1	419999.9

Table 6.2: Results of Sequential-HLEA on all *maximal plane* graphs.

All in all the Sequential-HLEA clearly improves the results on maximal plane graphs when compared to those of KCMST-LD, since it solves 177 of 180 instances provably optimal (by contrast KCMST-LD$_{\text{SO\&KLS}}$: 157 and KCMST-LD$_{\text{VA\&KLS}}$: 159). Furthermore it accomplishes to derive superb solutions unregarded the type of correlation. The latter was not so by solely using KCMST-LD$_{\text{VA\&KLS}}$. Another test run of the Sequential-HLEA with applying the settings of KCMST-LD$_{\text{SO\&KLS}}$ yielded notable worse results on the strongly correlated graphs. This is because the latter variant was already shown to find fewer optimal solutions as the variant with VA as well because KCMST-EA is weakest on these graphs, which leads to an unfavorable combination. Thus the Sequential-HLEA is better applied with the Volume Algorithm, at least on these instances.

Though KCMST-EA is generally slowest on plane graphs having many nodes, the assigned edge-profits and the edge list reduction yield an intensified search, mostly leading to (relatively) few necessary iterations until the optimal solution is found. This allows to bind the maximal number of iterations by a reasonably small value, thereby also decreasing the maximal time overhead in case no optimal solution can be found.

6.3.2 Experimental Results of the Intertwined-HLEA

The proposed Intertwined-HLEA will be applied on some complete and big complete graphs. Although it is not very likely that there is an improvement over the results of KCMST-LD$_{\text{VA\&KLS}}$, we will investigate the behavior and the effects of the hybrid algorithm, especially the performance of KCMST-EA. The latter algorithm will once again be used without and with the profit-based heuristics.

The settings of KCMST-LD are those of KCMST-LD$_{\text{VA\&KLS}}$ as supplied before for (large) complete graphs: $maxSteps = 1000$, $useLBfromKLS = 1$ and in case of weak or strong correlation we set $klsType = 2$, $klsMinGap = 0.99$ and $klsRetries = 100$, whereas for uncorrelated graphs KLS will not be used. The variant of KCMST-EA and KCMST-EA$_{\text{H}}$ (now denoted as KCMST-EA$_{\text{H-TX}}$ because of the tournament crossover) without and with applying profit-based heuristics, respectively, is equal to those described in Section 6.2.1. The only exception is that due to the worse initial population (i.e. having a low average fitness) created for weakly and strongly correlated graphs when using the straightforward profits of (6.4), we always set $heuinik = 0$ for these two correlation types. Unlike to uncorrelated graphs, where setting $heuinik = 1.5$ as suggested in [24] generally yields an initial population with an average fitness value increased up to 50%. A third run over all 10 instances per graph will be made using the same settings as KCMST-EA$_{\text{H-TX}}$ but a greedy crossover, i.e. setting $crotype = 1$, which will be denoted as KCMST-EA$_{\text{H-GX}}$. The intertwined execution is selected with $execOrder = 0$, and in each cycle KCMST-LD is performed for 100 iterations ($iterLD = 100$) and KCMST-EA for 300 ($iterEA = 300$).

The results according to these settings are given in Table 6.3. In addition to former attributes we state again $iter_{\text{EA}}$, but this time it is the number of iterations of KCMST-EA averaged over all 10 instances since it is obviously applied everytime, *sol* is the average fitness of the best chromosomes generated by KCMST-EA, whereas solutions integrated

from KCMST-LD do not count as being generated. If additionally a superscript number is given, i.e. sol^n, then n states how many of the optimal solutions were found by KCMST-EA. The information exchange is illustrated by stating the values $s_{\mathrm{l,e}}$ and $s_{\mathrm{e,l}}$, which are the average number of solutions transferred from KCMST-LD to KCMST-EA and the average number of solutions (chromosomes) generated by KCMST-EA that updated the lower bound of KCMST-LD, respectively. Thus only one of the latter chromosomes can be the best one generated.

Since the upper bounds were already given in Tables 5.10, 5.11 and 5.12 and all provably optimal solutions were found (except for K_{100u4} because of the non-optimal upper bound) we omit these values.

With increasing exploitation of the edge-profits in KCMST-EA via the heuristics, the fitness of its average best solutions is increasing, too. At the same time $s_{\mathrm{l,e}}$ decreases, i.e. fewer solutions can be transferred from KCMST-LD to KCMST-EA since the latter algorithm generates good solutions on its own. In fact these solutions are increasingly more often than not better than those of KCMST-LD, which is seen by the notable increase of $s_{\mathrm{e,l}}$. The positive effect of the heuristic is also apparent when considering the number of optimal solutions found by KCMST-EA: none when using no heuristic at all, 11 with rank-based selection for mutation and tournament selection for recombination and 23 when using greedy edge selection for recombination instead.

Whereas KCMST-EA clearly benefits from the intertwined execution by receiving better solutions, and probably use them to generate even better ones, and by utilizing the updated edge-profits, the only advantage for KCMST-LD would be to receive the optimal lower bound from KCMST-EA before deriving it itself. Although this occurs for some instances, the number of average iterations never decreased significantly. Furthermore the overall running time is increasing on all graphs, at most about 20% to 30%. None the less applying the Intertwined-HLEA has a positive effect, namely better solutions right from the start for weakly and strongly correlated graphs and generally more updates of the best lower bound for all graphs. Thus the lower bound is steadily higher when applying the Intertwined-HLEA instead of solely KCMST-LD, which would clearly be an advantage if a time limit were given. This effect is illustrated in Figure 6.4, again for three of the largest complete graphs. Thereby showing the resulting lower bound of the applied variants of the Intertwined-HLEA, using different settings for KCMST-EA, on the left side and the comparison of the corresponding lower bound of KCMST-LD (using the settings as described above) and those of the best performing Intertwined-HLEA run on the right side (whereas e.g. Intertwined-$\mathrm{HLEA}_{\mathrm{H\text{-}GX}}$ means using KCMST-$\mathrm{EA}_{\mathrm{H\text{-}GX}}$).

Instance		**Intertwined-HLEA**																	
		using KCMST-EA						**using KCMST-EA$_{\text{H-TX}}$**						**using KCMST-EA$_{\text{H-GX}}$**					
$graph_{corr}$	LB	$t[s]$	$iter$	$iter_{\text{EA}}$	sol	$s_{\text{l,e}}$	$s_{\text{e,l}}$	$t[s]$	$iter$	gen	sol	$s_{\text{l,e}}$	$s_{\text{e,l}}$	$t[s]$	$iter$	gen	sol	$s_{\text{l,e}}$	$s_{\text{e,l}}$
K_{100u}	9698.0	1.60	1040	2970	8483.6	8	2	1.95	1082	3120	9693.3[1]	3	8	2.05	1044	3060	9695.4[3]	2	8
K_{100w}	3421.4	1.56	765	2130	3266.2	6	4	1.78	758	2160	3391.3	3	6	1.70	748	2100	3401.2[1]	2	7
K_{100s}	2771.9	2.91	1340	3840	2743.3	5	7	3.26	1324	3810	2766.8	4	9	3.17	1330	3810	2767.9	2	11
K_{200u}	19739.4	6.97	1533	4440	17800.1	11	3	8.43	1528	4410	19735.8[4]	4	14	8.18	1444	4173	19737.7[3]	4	14
K_{200w}	6928.0	7.97	1154	3270	6730.4	8	5	8.27	1083	3120	6886.6	3	8	8.00	1080	3120	6891.8	2	9
K_{200s}	5572.0	16.04	2297	6720	5524.6	9	10	17.92	2328	6840	5563.4	7	14	17.91	2356	6930	5564.0	6	18
K_{300u}	29770.6	23.02	2025	5910	28515.2	13	5	25.51	2054	6030	29767.5[1]	4	17	25.56	1943	5679	29769.1[5]	4	19
K_{300w}	10734.1	24.11	1446	4170	10479.0	9	9	26.08	1470	4260	10680.3	4	11	26.08	1462	4230	10681.9	3	11
K_{300s}	8372.0	57.62	3423	10110	8313.1	11	18	62.96	3537	10470	8357.1	9	16	62.45	3490	10300	8362.0[3]	7	27
K_{400u}	39818.0	57.68	2615	7710	38970.9	15	7	59.02	2523	7440	39815.9	4	22	60.13	2546	7500	39816.1[3]	3	25
K_{400w}	14342.2	57.90	1800	5250	13837.7	13	9	60.98	1814	5310	14264.8	5	14	60.27	1792	5220	14278.2	4	14
K_{400s}	11172.0	149.22	4738	14070	11081.4	7	25	155.59	4735	14095	11142.9[1]	8	16	155.85	4753	14078	11151.0[1]	7	39
K_{500u}	49860.1	117.30	3120	9210	48929.1	15	10	122.25	3139	9262	49858.0[4]	5	27	127.48	3204	9450	49839.1[1]	5	29
K_{500w}	17950.0	125.75	2330	6840	17548.6	13	14	130.53	2321	6810	17834.7	6	17	129.44	2278	6660	17873.4	5	18
K_{500s}	13972.0	314.93	5793	17220	13869.3	11	28	322.02	5759	17085	13925.4	8	19	320.91	5715	16955	13961.8[3]	7	43

Table 6.3: Results of Intertwined-HLEA on some *(big) complete* graphs.

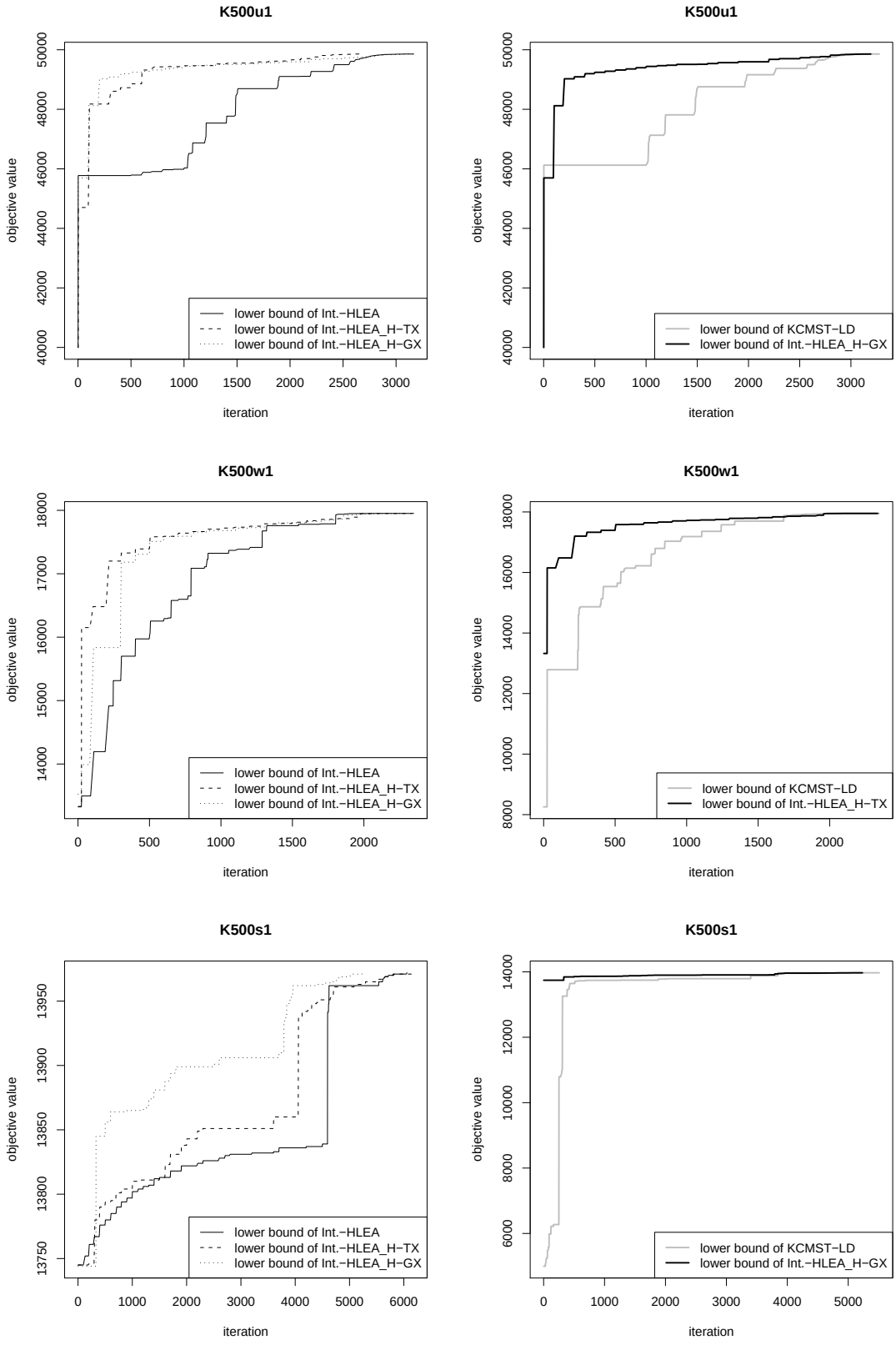


Figure 6.4: Lower bounds of Intertwined-HLEA using different variants of KCMST-EA and comparison of best variant to KCMST-LD against iterations of the latter.

7 Implementation Details

Everything was implemented in C++ using the GCC 4.0.3 compiler on a Pentium M 1.6GHz with 1.25GB RAM.
LEDA [39], a C++ class library for efficient data types and algorithms, was used whenever possible. Further EAlib [38], a problem-independent C++ library suitable for the development of efficient metaheuristics for COPs, was especially used to implement the Evolutionary Algorithm (KCMST-EA), as well as for parameter handling and random number generation.
A plane graph is represented by a 2-dimensional hash map and a complete graph by a 2-dimensional array, both holding for a pair of node indices the corresponding edge index.

7.1 Adapted COMBO Algorithm

The COMBO algorithm is available online[1] provided by David Pisinger. Since the Lagrangian multipliers $\lambda_i,\ i = 1, \ldots, m$ are double-values, but the algorithm only works with integer values of profits and weights, the former values have to be transformed to a suitable integer value. To obtain a certain degree of precision the double-values are multiplied by 10^6 for plane graphs and 10^8 for complete graphs and transformed to an integer-value. These multiplier values are different because of the varying number of nodes of plane and complete graphs, which corresponds to the number of items which will be chosen. Setting them too small was shown to probably result in the inability to derive the optimal upper bound, so we chose rather big values. Yet this conversion has the side effect of having high profit values afterwards which can lead to an overflow in the course of the computation. Due to this the COMBO algorithm must be adapted to handle bigger values by introducing larger variable types for some macros and declarations. Another small change of the item structure is necessary, to include the item number initialized with the index of the item, because it is needed afterwards to set the variable vector $y \in \{0, 1\}^m$ corresponding to the items. All these changes applied to the original file *combo.c* are shown in Listing 7.1, of course the header file must be changed correspondingly, too.

7.2 Class Diagram

The class diagram of all classes involved is shown in Figure 7.1. For the sake of completeness the classes of EAlib are included too, see [38] for further details. We give a short description of the created classes:

[1] see http://www.diku.dk/~pisinger/codes.html (last checked on October 10, 2006)

```
...
/* section 'macros'- long instead of int: */
#define NO(a,p)          ((long) ((p) - (a)->fitem + 1))
#define DIFF(a,b)        ((long) (((b)+1) - (a)))
...
/* section 'type declarations' - long long instead of long and
   long double instead of double */
typedef long long     itype;    /* item profits and weights */
typedef long long     stype;    /* sum of profit or weight  */
...
typedef long double   prod;     /* product of state, item   */
...
/* item record */
/* included the item number */
typedef struct {
  itype    p;                  /* profit                   */
  itype    w;                  /* weight                   */
  boolean  x;                  /* solution variable        */
  int number;                  /* number of item           */
} item;
```

Listing 7.1: Changes to the standard COMBO algorithm in file *combo.c*.

- `kcmstLD`: Consists of all algorithms presented in Chapter 4. Because all procedures and functions are either problem specific or directly use problem specific data to speed up the computation, and are thus not meant to be reused, they were not decomposed into more classes.

- `edgeListChrom`: The base class of the chromosome used in [24] (i.e., the edge-set representation), derived from the abstract class chromosome in EAlib and containing general purpose methods like ranking the parental edges used in recombination or selecting an edge for mutation, both according to a chosen method (i.e. with or without any profit-based heuristic).

- `kcEdgeListChrom`: The chromosome class for the KCMST problem, with problem specific methods for initialization, recombination, mutation, repair and local search, see Section 6.2.

- `kcmstSteadyStateEA`: The class of an EA for the KCMST problem derived from the EAlib class steadyStateEA. It consists of methods from steadyStateEA adapted to our needs, like extending the method for performing a single generation by a local search option, as well as methods to integrate solutions from and export to integer arrays.

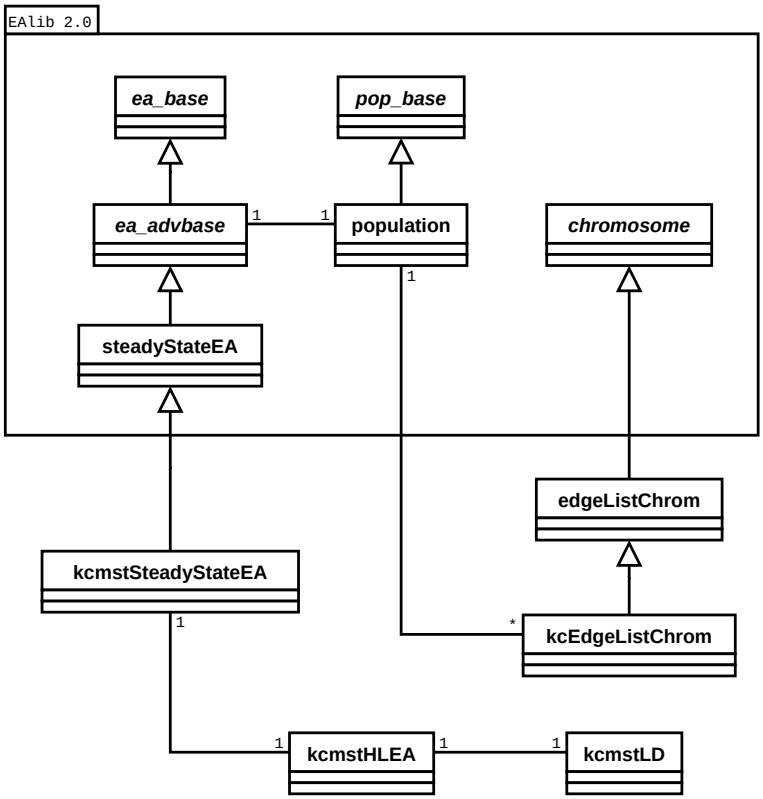


Figure 7.1: UML class diagram.

- `kcmstHLEA`: The class of the algorithm presented in Section 6.3. It combines `kcmstLD` and `kcmstSteadyStateEA` and is responsible for the appropriate information exchange in case of sequential or intertwined execution.

7.3 Program Parameters

Though all parameters were already mentioned at the appropriate place throughout the work, they will be summarized in this section. Those of the program `kcmstld`, which implements KCMST-LD, are given in Table 7.1. The additional parameters of the program `hlea`, the implementation of the Hybrid Lagrangian EA (HLEA) using KCMST-EA, are listed in Table 7.2 whereas the parameters of EAlib used for KCMST-EA are given in Table 7.3. KCMST-EA is only used in combination with KCMST-LD in HLEA, since it is not intended to be run solely. For other parameters, e.g. other possible termination conditions, that apply to every class derived of `steadyStateEA` (contained in EAlib) and thus to KCMST-EA implemented by `kcmstSteadyStateEA`, see [38].

Parameter	Description
$ifile$	Name of instance file (with and w/o directory).
$idir$	Base directory of instance file (not necessary if absolute position of file is given in $ifile$).
$isPlane$	Whether the graph is plane or not.
$steps$	Number of consecutive non-improving steps before changing the coefficient f.
$multiply$	The coefficient f is multiplied by this value when altered.
$maxSteps$	Number of maximal consecutive non-improving steps (i.e. while $LB \geq \lfloor UB \rfloor$) before terminating.
$volAlg$	If Volume Algorithm is used instead of Subgradient Optimization.
$optimalAlpha$	Determine α_{opt} in case $volAlg = 1$.
$alphaStart$	The start value of α.
$mstAlg$	Selects the algorithm to compute the MST, see Section 2.6.
$klsType$	The edge selection scheme in KLS, see *getNextEdge()* in Section 4.4.
$klsRetries$	The number of retries of KLS.
$klsMinGap$	The minimum gap when KLS starts, i.e. as soon as $LB/UB \geq klsMinGap$.
$useLBfromKLS$	If lower bounds derived by KLS are used for updating the target value T.

Table 7.1: Program parameters of `kcmstld` (KCMST-LD).

Parameter	Description
heuinik	Used factor for random initialization, see α in (6.1).
crotype	The type of crossover (0:RX, 1:GX, 2:TX).
heucrok	Tournament size in case $crotype = 2$.
prefib	If edges occurring in both parents are preferred.
heumut	Used mutation-heuristic (0:UNIF, 1:NORM).
heumutk	Used factor for rank-based selection of an edge, see β in (6.2).
klsTypeEA	The edge selection scheme of KLS in EA (0: random, 1: greedy).
klsRetriesEA	The number of retries of KLS used in EA.
execOrder	The type of execution of LD and EA (0: sequential, 1: intertwined).
reducedEL	If reduced edge list should be used for the EA in case $execOrder = 0$.
iterLD	The number of consecutive iterations of LD in case $execOrder = 1$.
iterEA	The number of consecutive iterations of the EA in case $execOrder = 1$.

Table 7.2: Program parameters of `hlea` (HLEA using KCMST-EA) in addition to those in Table 7.1.

Parameter	Description
maxi	Must be set to 1 since we are maximizing.
pmut	Probability/rate of mutating a new chromosome.
plocim	Probability with which locallyImprove (hence KLS) is called for a new chromosome.
tgen	The number of generations until termination.

Table 7.3: Additional program parameters of `hlea` from EAlib, used in KCMST-EA.

Parameter	Description
seed	The seed value for the random number generator.
graphType	The type of graph (0=plane, 1=complete).
corr	The type of correlation (u, w or s).
nodes	The number of nodes of the graph.

Table 7.4: Parameters of instance generator *gengraph*.

7.4 Instance Generators

The instance generator *mstkp* of Yamada et al. for plane and complete graphs was already used in [33], thereby slightly modified to provide a more convenient interface and named *mstkp2*. We will give the usage message (shown when run without a parameter):

```
Usage: ./mstkp2 seed graph-type graph correlation

        seed: integer value
        graph-type: 0=plane,1=complete
        graph: p___x___ for plane graph or number of nodes in case of
               complete graph
        correlation: 0=uncorrelated, 1=weakly correlated and
                     2=strongly correlated

examples:
        ./mstkp2 2 0 p400x1120 1
        ./mstkp2 7 1 100 2
```

The parameter sequence is fixed, whereas the knapsack constraint is hardcoded into the program for both graph types, using the formula presented in Section 5.1. The program uses the random number generator of the C++ standard library. The generator *mstkp2* was used to create the instances of test set 1. The 10 random instances per graph and correlation type were generated by running *mstkp2* with a seed value ranging from 1 to 10.

Our new instance generator *gengraph*, for maximal plane and complete graphs, uses LEDA to create random maximal plane graphs, whereas the parameter handling routine and the random number generator (being more reliable than that of the C++ standard library) of EAlib is applied. The generator is configured by the parameters in Table 7.4. It was used to generate the instances of test set 2, thereby proceeding as described before.

8 Conclusions

This thesis proposed a Lagrangian decomposition (LD) approach, an Evolutionary Algorithm (EA) and a Hybrid Lagrangian EA (HLEA) which combines both, for solving the Knapsack Constrained Maximum Spanning Tree (KCMST) problem.

The LD divides the problem into the Maximum Spanning Tree (MST) problem and the Knapsack Problem (KP). Suitable MST algorithms for the different graph types were implemented and a slightly adapted version of the publicly available COMBO algorithm, an enhanced Dynamic Programming algorithm, was used for the KP. To solve the Lagrangian dual problem the Subgradient optimization method as well as the more extended Volume Algorithm were applied. The solution of the MST subproblem, produced in every iteration, turned out to be quite often feasible, with increasing profit towards the end of the process. This can already be seen as a simple Lagrangian heuristic. To strengthen it, a problem specific local search was devised, which can be applied in a random or greedy variant.

Instances with plane graphs up to 1000 nodes, maximal plane graphs up to 12000 nodes and complete graphs up to 500 nodes were generated, thereby using three different correlations between profit and weight of an edge. Experimental results showed that the LD algorithm produces in almost all cases optimal upper bounds, only failed to do so for some instances of the smaller graphs, whereas only the Volume Algorithm was suited for all graph types. Although the generated lower bounds were generally good, the application of the local search towards the end of the process led for some graphs to impressive improvements. A comparison to previous results of exact methods showed that the LD algorithm is superior on all strongly correlated graphs and mostly to prefer for the other graphs as well, especially when taking the small running time into account. Due to the latter the algorithm was also successfully applied on much larger graphs. The only problem remained the few non-optimal upper bounds, making it impossible to prove the optimality of actually optimal solutions. It was tried to alleviate this by strengthening the LD with using the exact k-item KP (E-kKP) instead of the KP, but even then not all optimal upper bounds could be derived, besides resulting in a much worse running time.

Further a suitable EA was developed using the edge-set coding scheme, offering strong locality, and operators, including the same local search as the LD algorithm, working on this representation. These operators were designed to produce only feasible solutions and to possibly utilize heuristics based on edge-profits. It was also ensured that the mutation operator exhibits strong heritability. Tests using this EA suggested that applying the heuristics based on straightforward edge-profits, derived from the data given by an instance, are not of benefit. Though the best produced solutions are quite good, the EA is inferior to the LD algorithm, also in terms of run-time.

In the end both methods were appropriately combined, resulting in the HLEA. Thereby two possible collaborative combinations were considered, namely to run them either in sequential order or intertwined. In both cases the algorithms exchange as much information as possible. The Sequential-HLEA was applied to maximal plane graphs, where the optimal lower bound was sometimes missing. As it turned out, the algorithm was able to find it for almost all instances, due to the EA utilizing the heuristics based on the Lagrangian multiplier values. The effect of the Intertwined-HLEA was tested on complete graphs. Its application resulted in better solutions at the beginning and throughout the solution process, when compared to the LD algorithm. So if an additional time limit would be set, the Intertwined-HLEA is to prefer.

It would be of interest to create instances with other correlation types, since those that were generated were almost all solved to optimality, regardless of the size. Even better would be real-world instances, allowing to investigate the practical applicability of the devised algorithms.

Since the obtained bounds (both lower and upper) are very tight, in fact mostly optimal, a B&B algorithm could be devised utilizing these bounds by applying the LD algorithm instead of a Linear Programming relaxation. This would finally result in provably optimal solutions for all instances. Though this would particularly be useful for those few rather small graphs where it was not possible to derive the optimal upper bound, they were already solved to optimality in short time by our former B&C algorithm [33]. Additionally the running time of this B&B algorithm on large graphs might be worse than applying the presented hybrid algorithm twice or more and thereby probably finding the optimal solution.

Another potentially promising attempt would be to use a construction heuristic by utilizing the Lagrangian multiplier values as was successfully done for the local search and the EA. A suitable metaheuristic for this could be the Ant Colony Optimization (ACO) framework [8].

Bibliography

[1] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[2] L. Bahiense, F. Barahona, and O. Porto. Solving steiner tree problems in graphs with lagrangian relaxation. *Journal of Combinatorial Optimization*, 7(3):259–282, 2003.

[3] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[4] J. E. Beasley. Lagrangian relaxation. In C. R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[5] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[6] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.

[7] R. Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 2005.

[8] M. Dorigo and T.Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[9] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[10] M. L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1):1–18, 1981.

[11] M. L. Fisher. An application oriented guide to Lagrangean Relaxation. *Interfaces*, 15:10–21, 1985.

[12] D.B. Fogel and Z. Michalewicz. *How to Solve It: Modern Heuristics*. Springer, 2000.

[13] A. Frangioni. About Lagrangian Methods in Integer Optimization. *Annals of Operations Research*, 139(1):163–193, 2005.

[14] M. L. Fredman. On the efficiency of pairing heaps and related data structures. *J. ACM*, 46(4):473–501, 1999.

[15] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. The pairing heap: a new form of self-adjusting heap. *Algorithmica*, 1(1):111–129, 1986.

[16] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

[18] M. Gendreau and J.-Y. Potvin. Metaheuristics in Combinatorial Optimization. *Annals of Operations Research*, 140(1):189–213, 2005.

[19] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[20] M. Guignard. Lagrangean Relaxation. *Top*, 11(2):151–228, 2003.

[21] M. Haouaria and J. C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.

[22] M. Held and R. M. Karp. The travelling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[23] M. Held and R. M. Karp. The travelling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 1971.

[24] B. A. Julstrom and G. R. Raidl. Edge sets: an effective evolutionary coding of spanning trees. *IEEE Transactions on Evolutionary Computation*, 7(3):225–239, 2003.

[25] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer Verlag, 2004.

[26] J. B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48–50, 1956.

[27] C. Lemaréchal. Lagrangian Relaxation. In *Computational Combinatorial Optimization*, pages 112–156, 2001.

[28] D. Lichtenberger. An extended local branching framework and its application to the multidimensional knapsack problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, March 2005.

[29] S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45:414–424, 1999.

[30] B. M. E. Moret and H. D. Shapiro. An empirical analysis of algorithms for constructing a minimum spanning tree. In Frank Dehne, Joerg-Ruediger Sack, and Nicola Santoro, editors, *Proceedings of Algoritms and Data Structures (WADS '91)*, volume 519 of *LNCS*, pages 400–411, Berlin, Germany, 1991. Springer.

[31] B. M. E. Moret and H. D. Shapiro. How to find a minimum spanning tree in practice. In Hermann Maurer, editor, *Proceedings of New Results and New Trends in Computer Science*, volume 555 of *LNCS*, pages 192–203, Berlin, Germany, 1991. Springer.

[32] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA, 1989.

[33] S. Pirkwieser. Solving the Knapsack Constrained Maximum Spanning Tree Problem within an Extended Local Branching Framework. Praktikum aus Intelligente Systeme, Vienna University of Technology, Institute of Computer Graphics and Algorithms, January 2006.

[34] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[35] W. H. Press et al. Numerical recipes in C (second edition). *Cambridge University Press*, 1992.

[36] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technology Journal*, 36:1389–1401, 1957.

[37] J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2005.

[38] G.R. Raidl and D. Wagner. EAlib 2.0 - A Generic Library for Metaheuristics. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2005.

[39] Algorithmic Solutions. *The LEDA User Manual*, 2005. http://www.algorithmic-solutions.info/leda_manual/MANUAL.html (last checked on October 10, 2006).

[40] J. T. Stasko and J. S. Vitter. Pairing heaps: experiments and analysis. *Commun. ACM*, 30(3):234–249, 1987.

[41] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Boston, 1996. Second Edition: 2001.

[42] T. Yamada, K. Watanabe, and S. Katakoa. Algorithms to solve the knapsack constrained maximum spanning tree problem. *Int. Journal of Computer Mathematics*, 82(1):23–34, 2005.

[43] Y. Yamamato and M. Kubo. *Invitation to the Traveling Salesman's Problem (in Japanese)*. Asakura, Tokyo, 1997.