Combinatorial Optimisation: Relaxation, Duality and Heuristics

Mohammad Hasan Mansoor

Department of Management Science

Lancaster University Management School

A thesis submitted for the degree of

Doctor of Philosophy

February 2022

bi-smi llāh

(with God's name)

Abstract

Relaxation and dual-based heuristics have been a part of research in combinatorial optimisation since the early 1970s. This thesis extends that strand of research into less popular forms of relaxations in particular surrogate relaxation, which is theoretically a tighter relaxation than the two most common relaxations (Linear Programming and Lagrangian relaxations). The aim is to show surrogate dual information can add to the performance of dual-based matheuristics. In chapter 2 we provide some theoretical results related to surrogate and group relaxation. We follow it up with an exact and a heuristic surrogate dual method along with computation results, in chapters 3 and 4 respectively.

Finally, in chapter 5, we take a step back and seek to make an introductory empirical investigation into the value of good and better dual solutions in guiding primal heuristics using LP relaxation as an example.

Acknowledgements

A big thank you to the Department of Management Science at Lancaster University for providing me with the funding for my PhD. To the administrators of the department, in particular Gay, who have always been forthwith any support and administration needs I required.

To my supervisor Prof. Adam Letchford, whose passionate teaching at Masters level inspired my research interest in Operational Research. Then his ever supportive and keen interest in my PhD and belief in me without which I would not have been able to complete this thesis. To my other supervisor Trivikram Dokka for his unflinching support and candid meetings which were a great source of motivation and insight through the years.

To my parents whose continual prayers have granted me strength and who have sacrificed for my educational and personal journey. To my mother-in-law, who has helped us in so many ways and has been there for us when we needed her.

To our friends in Lancaster, the Kazmis, Shahs, Talibs, Khans, Kheiris and the rest, for being available to us at all times.

To my wife, Maheen, who has been a rock by my side and provided me with comfort and counsel through the ups and downs of our journey. Lastly, to our children - Muhammad, for his invigorating company and trying his best to help us, and Maryam, for her refreshing smiles and kisses

Statement of Originality

This thesis has not been submitted in support of an application for another degree at this or any other university. It is the result of my own work and includes nothing that is the outcome of work done in collaboration except where specifically indicated. Many of the ideas in this thesis were the product of discussions with my doctoral supervisors Prof. Adam N. Letchford and Dr. Trivikram Dokka.

Chapter 2 of this thesis has been published as:
T. Dokka, A.N. Letchford & M.H. Mansoor (2021) On the complexity of surrogate and group relaxation for integer linear programs. *Oper. Res. Lett.*, 49(4), 530–534.

Chapter 3 is under preparation for submission as:
T. Dokka, A.N. Letchford & M.H. Mansoor (2022) Revisiting Surrogate Relaxation for the Multidimensional Knapsack Problem.

Chapter 4 of this thesis has been submitted to *J. Heur.* as:
T. Dokka, A.N. Letchford & M.H. Mansoor (2022) Using Surrogate Relaxation as a Matheuristic.

Chapter 5 is under preparation for submission as:
T. Dokka, A.N. Letchford & M.H. Mansoor (2022) Anomalous Behaviour of Dual-Based Heuristics.

M. Hasan Mansoor
Lancaster University, UK

Contents

Chapter 1

Introduction

One hears about *optimisation* quite often both within and outside academia. In layman's terms, optimisation can be thought of as improving the way one does a certain thing. For example, people might think about "optimising" their morning routine, to be as time efficient as possible. Or an online retailer could speak of "optimising" the shopping experience on their website, to make it as easy as possible to browse through their collection. As a third example, a cyclist might think about "optimising" their saddle position; low enough to stop safely, yet high enough for an efficient peddle stroke.

Mathematical optimisation, sometimes also called *mathematical programming*, provides a formal framework for modelling and solving optimisation problems (see, e.g., [53, 143]). Mathematical optimisation is an extremely important branch of Applied Mathematics, having a wealth of applications not only in Operational Research and Management Science, but also in Statistics, Computer Science, Engineering and the Physical Sciences.

The essential features of a mathematical optimisation problem are:

- One or more *decision variables*, which represent quantities over which one has some control.
- One or more goals or *objectives*, which are quantities that one wishes to maximise or minimise.

- One or more *constraints*, which restrict the possible values that the decision variables can take.

It must be possible to represent all objectives and constraints as functions of the decision variables. (The process of converting a real problem into its mathematical representation is called *modelling* or *formulation*.)

This thesis is concerned with *combinatorial* optimisation problems, in which each variable is restricted to take values from a finite set. Although these problems may seem a little obscure at first sight, they arise frequently in industry. Examples of such problems include facility location, vehicle routing, timetabling, scheduling, rostering, packing and cutting problems, along with various network design and production planning problems (see, e.g., the textbooks [40, 105, 122, 143]).

A summary of the contribution of the thesis will be given later on, in section 1.8. First, however, we provide some background, notation and terminology (in Sections 1.1 to 1.7).

1.1 Linear and Nonlinear Programming

An important kind of optimisation problem arises when there is a single objective, all variables are continuous, and all functions are linear. Such a problem is called a *Linear Program* or LP (see, e.g., the textbooks [45, 136, 139]). An LP with n variables and m constraints can be written as:

$$\min \left\{ c^T x : \, Ax \geq b, \, x \geq 0 \right\}, \tag{1.1}$$

where $c \in \mathbb{Q}^n_+$ is the vector of *cost coefficients*, $A \in \mathbb{Q}^{mn}_+$ is the matrix of *constraint coefficients*, and $b \in \mathbb{Q}^m_+$ is the vector of *right-hand sides*. The term $c^T x$ is called the *objective function*, and the set of all x vectors that satisfy the constraints is called the *feasible region*.

We now make some remarks. First, note that the feasible region is a convex polyhedron. Second, in this thesis, all vectors are assumed to be column vectors rather than row vectors. Third, the condition $x \geq 0$ is assumed to operate "component-wise". (That is, we require $x_j \geq 0$ to hold for $j = 1, \ldots, n$.) Finally, one may change the objective from "min" to "max", or change the sense of some or all constraints from "$\geq$" to "$\leq$" or "$=$". The resulting problem is still called an LP.

Although LPs may appear to be a very restricted family of problems, they are very useful in practice. Important examples of applications include blending problems, problems involving flows in networks, problems involving allocation of divisible resources, and various problems in statistics and data mining (e.g., [45, 139, 143]).

A variety of algorithms exist to solve LPs (e.g., [45, 122, 136]). The most well-known is the *simplex* method, proposed by Dantzig in 1947. It begins by finding a corner (or extreme point) of the feasible region. It then iteratively moves to an adjacent corner with lower cost, until no such corner exists. Also worth mentioning are *Interior Point Methods* or IPMs, which find a point in the interior of the feasible region and then iteratively move towards an optimal point.

Modern IPMs can be competitive with the simplex method, especially on very large LPs. On the other hand, the simplex method makes it easier to "re-optimise" after a small change has been made to the problem (such as the addition or deletion of a constraint or variable).

An obvious way to generalise LPs is to allow some of the functions involved to be nonlinear (e.g., [12, 20]). A *Nonlinear Program* or NLP with n variables and m constraints can be written as:

$$\begin{array}{rl} \min & f^0(x) \\ \text{s.t.} & f^i(x) \leq 0 \quad (i = 1, \ldots, m) \\ & x \in \mathbb{R}^n. \end{array}$$

Here, $f^0(\cdot), \ldots, f^m(\cdot)$ are arbitrary continuous functions mapping $\mathbb{R}^n$ onto $\mathbb{R}$. (Note that the convention in the NLP literature is to write the objective as "min" and the constraints as "$\leq$".)

NLPs have many applications, for example in Operational Research, Statistics, Finance and Engineering. Unfortunately, they can be much harder to solve than

LPs, due for example to the possible presence of local minima that are not global minima. If however all of the functions involved are *convex*, then NLPs typically become much easier to solve. For details on the available solution approaches for the convex case, along with a description of the main applications, we refer the reader to [27].

1.2 Integer Programming and Combinatorial Optimisation

Integer programming refers to optimisation problems in which some or all of the variables are constrained to take integer values (see, e.g., [36,38,122,136]). An important family of integer programs is that of the *Mixed-Integer Linear Programs* or MILPs. An MILP with n variables and m constraints can be written in the form:

$$\min \quad c^T x \tag{1.2}$$

$$\text{s.t.} \quad Ax \geq b \tag{1.3}$$

$$x \in \mathbb{R}^n_+ \tag{1.4}$$

$$x_j \in \mathbb{Z} \quad (j \in I), \tag{1.5}$$

where $c \in \mathbb{Q}^n_+$, $A \in \mathbb{Z}^{mn}_+$, $b \in \mathbb{Z}^m_+$, and the set $I \subseteq \{1, \ldots, n\}$ contains the indices of the integer-constrained variables. It is also common to hear of "ILPs" (in which all variables are integral), and "0-1 LPs" (in which all variables are binary).

An obvious application of integer programming is when one is dealing with indivisible items, such as jobs, people, vehicles or machines but the reach of integer programming goes far beyond that. Indeed, as far back as 1960, Dantzig [44] pointed out that *binary* variables can be used to model a variety of logical conditions, and also to approximate nonlinear functions with piecewise-linear functions. Since then, a huge variety of problems have been modelled successfully as 0-1 LPs, ILPs or MILPs (see, e.g., [36, 94, 122, 136, 143]). We remark that a 0-1 LP can be formulated as an NLP, by replacing the binary condition $x_i \in \{0, 1\}$ with the quadratic constraint

$x_i = x_i^2$. Nevertheless, integer programming and nonlinear programming are usually treated separately, since they use very different solution methods.

Integer programming is closely related to *combinatorial optimisation* (e.g., [40, 105, 122, 143]). A generic *Combinatorial Optimisation Problem* (or COP) can be defined as follows. We are given a finite ground set E and a family $\mathcal{S}$ of subsets of E. The members of $\mathcal{S}$ are called *feasible solutions* and all other subsets of E are called *infeasible*. We are also given an *objective function* $f(\cdot)$, which maps each subset of E to a real number. The task is to find a member $S \in \mathcal{S}$ that minimises $f(S)$. (One may of course wish to maximise instead of minimise.)

Many COPs have linear objective functions. That is, there exists a vector $c \in \mathbb{Q}^E$ such that

$$f(S) = \sum_{e \in S} c_e \qquad \big(\forall S \subseteq E\big).$$

In this case, it is often possible to formulate the COP as a 0-1 LP. For each $e \in E$, define a binary variable x_e that takes the value 1 if and only if $e \in S$. Then minimise $\sum_{e \in E} c_e x_e$ subject to a suitable collection of linear constraints. The constraints must be chosen in such a way that a vector $x \in \{0,1\}^E$ satisfies them if and only if it represents a feasible solution to the COP.

To illustrate the difference between a COP and its 0-1 LP formulation, we consider the example of the 0-1 Knapsack Problem and the Set Covering Problem.

Definition 1.1 (0-1 Knapsack Problem or 0-1 KP) *We have n "items", where n is a positive integer. For each item $j = 1, \ldots, n$, we are given a positive rational "profit" p_j and a positive integer "weight" w_j. We are also given a positive integer "capacity" c. The task is to find a set of items of maximum total profit, whose collective weight does not exceed the capacity.*

Here, $E = \{1, \ldots, n\}$, and a set $S \subseteq E$ is feasible if and only if $\sum_{e \in S} w_j \leq b$. The function $f(S)$ is simply $\sum_{j \in S} p_j$. The corresponding 0-1 LP formulation is:

$$\max \Big\{ p^T x : \; w^T x \leq c, x \in \{0,1\}^n \Big\},$$

where the binary variable x_j takes the value 1 if and only if the jth item has been selected.

Definition 1.2 (Set Covering Problem or SCP) *We are given positive integers m and n, a family of sets $T_1, \ldots, T_n \subset \{1, \ldots, m\}$, and a cost c_j for $j = 1, \ldots, n$. The task is to find a minimum-cost collection of sets such that each member of $\{1, \ldots, m\}$ is contained in at least one set in the collection.*

Here, $E = \{1, \ldots, n\}$ as before, and a set $S \subseteq E$ is feasible if and only if $\cup_{e \in S} T_e = \{1, \ldots, m\}$. The function $f(S)$ is simply $\sum_{j \in S} c_j$. The corresponding 0-1 LP formulation is:

$$\min \left\{ c^T x : \ Ax \geq e_m, \ x \in \{0,1\}^n \right\}. \tag{1.6}$$

Here, the binary variable x_j indicates whether the jth set has been selected; $A \in \{0,1\}^{mn}$ is a matrix whose columns encode the sets $T_1, \ldots, T_n$; and e_m denotes the all-ones vector of order m.

We now give an example of a COP that has a nonlinear objective function, yet can still be formulated as a 0-1 LP.

Definition 1.3 (Simple Plant Location Problem or SPLP) *We are given positive integers m and n, representing the number of 'facilities' and 'clients', respectively. For $i = 1, \ldots, m$, we are given a constant $f_i \in \mathbb{Q}_+$, representing the cost of opening facility i. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, we are given a constant $c_{ij} \in \mathbb{Q}_+$, representing the cost of serving client j from facility i. The task is to decide which facilities to open, and assign each client to an open facility, in order to minimise the total cost.*

To fit this within our framework, it suffices to let $E = \{1, \ldots, m\}$. A set $S \subseteq E$ is feasible if and only if it is non-empty. The objective function to minimise is:

$$f(S) = \sum_{i \in S} f_i + \sum_{j \in J} \min_{i \in S} \{c_{ij}\}.$$

To formulate this as a 0-1 LP, we use two sets of variables. For $i = 1, \ldots, m$, let y_i take the value 1 if and only if facility i is opened. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, let x_{ij} take the value 1 if and only if client j is served from facility i. We then have:

$$\begin{array}{lll} \min & \sum_{i=1}^{m} f_i y_i + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} & \\ \text{s.t.} & \sum_{i=1}^{m} x_{ij} = 1 & (\forall j) \\ & y_i - x_{ij} \geq 0 & (\forall i, j) \\ & x \in \{0,1\}^{mn}, y \in \{0,1\}^{m}. & \end{array}$$

1.3 Algorithms and Complexity

An *algorithm* is a step-by-step procedure that can be written as a computer program (see, e.g., Harel & Feldman [87]). It often happens that more than one algorithm is known for a given problem. (For example, there are several algorithms known for sorting a list of integers, such as 'bubblesort', 'quicksort' and 'heapsort'.) This leads naturally to the question of which algorithm is fastest.

In Computer Science, it is common to measure the number of elementary mathematical operations (addition, multiplication, comparison, etc.) that an algorithm will perform, for a given input size, in the *worst case*. This can easily be determined for an instance of size 'n' by counting the number of nested 'loops' (such as 'for', 'while' or 'repeat' loops) that run n times. For example, an algorithm that contains one loop of nesting depth three and two loops of nesting depth two will perform around $n^3 + 2n^2$ elementary operations in the worst case.

Note that, when n is large, n^3 will be much larger than $2n^2$. Computer scientists use the so-called "big O" notation to deal with this (see again [87]). An algorithm that performed around n^3+2n^2 elementary operations in the worst case would be said to run in $O\big(n^3\big)$ time. More generally, an algorithm is said to run in $O\big(f(n)\big)$ time, for some function $f(n)$, if there exists a positive constant c such that the number of elementary operations is never more than $cf(n)$.

An algorithm is said to run in 'polynomial time' if it runs in $O\big(n^c\big)$ time for some positive constant c. It is said to run in 'exponential time' if it runs in $O\big(c^n\big)$ time for some positive constant c. Intuitively, algorithms that runs in polynomial time are more likely to be useful than ones that runs in exponential time. There are however exceptions. For example, the Simplex algorithm is very fast for most LPs arising in practice, but it takes exponential time in the worst case [102].

It can sometimes be useful to express the running time of an algorithm in terms of more than one parameter. For example, an algorithm for solving an optimisation problem that involves graphs might run in $O(mn)$ time, where n is the number of nodes and m is the number of edges.

It can also happen that the running time of an algorithm depends on some numeric value that appears as part of the input. A well-known example is the classical dynamic programming algorithm for the 0-1 KP, discovered by Bellman [18]. It runs in $O(nc)$ time, where n is the number of items and c is the knapsack capacity.

Note that Bellman's algorithm does *not* run in polynomial time. This is because the knapsack capacity c will be represented in binary on a computer. If d is the number of bits needed to represent c, then the algorithm 'really' runs in $O\big(n\,2^d\big)$ time, which is exponential in d. On the other hand, provided that c is reasonably small, then Bellman's algorithm will be fast enough to be useful. The algorithm is said to run in 'pseudo-polynomial time', which means that it runs in polynomial time if c is bounded by a polynomial in n.

Instead of analysing specific algorithms, it is also possible to study the complexity of specific problems (see [67,87]). A problem is said to be 'solvable in polynomial time' if there exists an algorithm that solves the problem exactly in polynomial time. For example, the problem of multiplying two $n \times n$ matrices can be solved in polynomial time, since there is a simple algorithm that runs in $O\big(n^3\big)$ time.

When discussing complexity, computer scientists often prefer to work with *decision problems* rather than optimisation problems. A decision problem is a problem that has a 'yes-no' answer. A common example is the following:

Definition 1.4 (Partition Problem or PP) *We have a positive integer n and a set of positive integers $a_1, \ldots, a_n$. Does there exists a set $S \subset \{1, \ldots, n\}$ such that $\sum_{j \in S} a_j = \frac{1}{2} \sum_{j=1}^{n} a_j$?*

Note that the PP can be reduced to the 0-1 KP. This is done by setting the profits and weights of the items to $a_1, \ldots, a_n$, and setting the knapsack capacity to $\frac{1}{2}\sum_{j=1}^{n} a_j$. If the optimal solution to the resulting 0-1 KP instance has a profit equal to the knapsack capacity, then we know that the answer to the PP is 'yes'. Moreover, suppose that the answer to a given PP instance is 'yes', and someone has found a suitable set S. They can quickly convince someone that the answer to the given instance is 'yes', just by showing them the set S. In other words, the set S is a 'short certificate' that the answer is 'yes'.

This leads to the following definitions. The set of all decision problems that can be solved in polynomial time is called $\mathcal{P}$. The set of all decision problems for which there is a short certificate when the answer is 'yes' is called $\mathcal{NP}$. By definition, $\mathcal{P} \subseteq \mathcal{NP}$, and a famous open question in Computer Science is whether $\mathcal{P} = \mathcal{NP}$.

A problem is called '$\mathcal{NP}$-complete' if (a) it is in $\mathcal{NP}$, and (b) all other problems in $\mathcal{NP}$ can be reduced to it in polynomial time. The PP, mentioned above, is $\mathcal{NP}$-complete (Karp [96]). The $\mathcal{NP}$-complete problems can be thought of as the 'hardest' problems in $\mathcal{NP}$. If a polynomial-time algorithm were ever found for any $\mathcal{NP}$-complete problem, then $\mathcal{NP}$ would equal $\mathcal{P}$. Most computer scientists believe that this is unlikely.

Of course, most optimisation problems (such as MILPs and COPs) are not decision problems. We can however easily define a decision version of any given optimisation problem. Instead of requiring one to find the *optimal* solution, we can ask whether a solution *of cost less than* k exists, where k is an input parameter. In principle, one can then solve the optimisation problem by solving the decision version a polynomial number of times, provided that all variables are bounded. This can be done for example, by performing binary search on the optimal value.

An optimisation problem is called '$\mathcal{NP}$-hard' if its corresponding decision problem is $\mathcal{NP}$-complete. Problems that are $\mathcal{NP}$-hard are 'at least as hard as' the $\mathcal{NP}$-complete problems. The 0-1 KP is $\mathcal{NP}$-hard, since the decision version includes the PP. This implies that 0-1 LPs are $\mathcal{NP}$-hard as well. In fact, many other important COPs are $\mathcal{NP}$-hard, including the SCP and SPLP (see again [67]).

To end this section, we mention that an $\mathcal{NP}$-hard problem that can be solved in pseudo-polynomial time is called *weakly* $\mathcal{NP}$-hard. Otherwise it is called *strongly* $\mathcal{NP}$-hard. The 0-1 KP is weakly $\mathcal{NP}$-hard, but the SCP and SPLP are strongly $\mathcal{NP}$-hard [96, 106], and so is 0-1 LP in general [67].

1.4 Relaxation and Duality

In this section, we discuss two other key ideas in optimisation: *relaxation* and *duality.*

Relaxation means making the original problem easier to solve by expanding the set of feasible solutions. Typically, this is done by dropping some of the constraints. The solution of the relaxed problem gives a *lower bound* on the optimal cost for the original problem (assuming that the objective is to minimise cost).

The most common way to relax a MILP of the form (1.2)-(1.5) is simply to drop the integrality constraint (1.5). This is called *continuous relaxation.* Note that the relaxed problem is an LP, which is likely to be much easier to solve than the original MILP.

It may happen that the solution to the LP is already feasible for the original MILP. If that happens, the solution is optimal for the MILP and we are done. This happens, for example, when the constraint matrix A is totally unimodular (see [90, 136]). In practice, however, the LP solution is frequently infeasible for the MILP. In that case, one must do more work (see the next section for details).

Another popular approach is *Lagrangian relaxation* (LR). We partition the constraints $Ax \geq b$ in (1.3) into a set of 'complicating' constraints and a set of 'simple' constraints. Let $A^1 x \geq b^1$ and $A^2 x \geq b^2$ denote the two sets of constraints, and suppose that the number of complicating constraints is t. We pick a vector $\lambda \in \mathbb{R}^t_+$ of *Lagrangian multipliers*, and then solve the following simpler MILP:

$$\min \Big\{ c^T x + \lambda^T \big(b^1 - A^1 x\big) : \, A^2 x \geq b^2, \, (1.4), \, (1.5) \Big\}. \tag{1.7}$$

The idea here is that we penalise violation of the complicating constraints in the objective function. It can be shown that, for any choice of λ, LR yields a lower bound on the optimal cost for the original MILP [56, 70].

Surrogate relaxation (SR), proposed in [75, 79], is similar to LR. The difference is that, instead of modifying the objective function (1.2), we replace the complicating constraints with a single constraint. More precisely, we pick a vector $\mu \in \mathbb{R}^t_+$ of *surrogate multipliers* and replace the constraints $A^1 x \geq b^1$ with the constraint:

$$\big(\mu^T A^1\big) x \geq \mu^T b^1. \tag{1.8}$$

The constraint (1.8) is called a 'surrogate' constraint.

Yet another relaxation method, called *group relaxation* (GR), was introduced by Gomory [77, 78]. Consider again the MILP (1.2)-(1.5). If we define a vector $s \in \mathbb{R}^m_+$ of surplus variables, we can write the MILP as:

$$\min \Big\{ c^T x : \; Ax - s = b, \; (1.4), \; (1.5), \; s \in \mathbb{R}^m_+ \Big\}. \tag{1.9}$$

Now, suppose we solve the continuous relaxation of this modified MILP with the Simplex method. This yield a basic optimal solution, say (x^*, s^*). Exactly m variables will be basic and n variables will be non-basic at (x^*, s^*). The GR is then obtained from (1.9) by retaining integrality, but dropping the non-negativity restriction on the basic variables. Provided that (x^*, s^*) is non-degenerate, this is equivalent to dropping all non-binding constraints in the original MILP (including all non-binding non-negativity constraints, if any). Thus, GR yields a lower bound that is at least as good as the one obtained with continuous relaxation.

Closely related to relaxation is the concept of *duality.* Suppose we have two optimisation problems, called 'P' and 'D'. Suppose also that P is a minimisation problem, whereas D is a maximisation problem. We call D a *dual* of P if the optimal profit for D is no larger than the optimal cost for P. In that case, P is called the *primal* and D is called the *dual.* If the optimal profit for D is *equal* to the optimal cost for P, we call D a *strong* dual; otherwise it is a *weak* dual (see, e.g., [45,122,139]).

There is a well-developed duality theory for LPs (see, e.g., [45, 136, 139]). In particular, if the primal problem is an LP of the form (1.1), then the following LP is a strong dual:

$$\max \big\{ b^T y : \; A^T y \leq c, \; y \geq 0 \big\}, \tag{1.10}$$

where $y \in \mathbb{R}^m_+$ is the vector of dual variables.

We remark that the LP (1.10) is also a weak dual for any MILP of the form (1.2)-(1.5). Indeed, the optimal cost for the LP (1.1) is itself a lower bound for such an MILP. From this it follows that the optimal profit for (1.10) cannot exceed the optimal cost for the MILP.

There is also a duality theory for LR [70]. For a given multiplier vector $\lambda \in \mathbb{R}^t_+$, let $L(\lambda)$ denote the lower bound from the relaxed problem (1.7). The problem of finding the vector λ which maximises $L(\lambda)$ is a (typically weak) dual of the original MILP, called the *Lagrangian* dual. For details on Lagrangian duality and its applications, see, e.g., [56, 57, 84, 109].

In a similar way, one can define a *surrogate* dual. For a given multiplier vector $\mu \in \mathbb{R}^t_+$, let $S(\mu)$ denote the lower bound from the relaxed problem. The surrogate dual is then the problem of finding the vector μ that maximises $S(\mu)$. For details, see, e.g., [75, 79, 98].

1.5 Exact Methods for Integer Programming

Exact methods solve an optimisation problem to proven optimality, which involves not only providing a feasible solution, but also a proof that no better solution exists. We now review popular general purpose exact methods for MILPs.

Branch-and-bound, first proposed by Land and Doig in [108], implicitly enumerates the feasible set by a strategy of branching (i.e., partitioning) and bounding (i.e., pruning) the search space. In LP-based branch-and-bound, the LP relaxation of the original MILP is solved, yielding a solution x^* and a lower bound $c^T x^*$ (minimisation case). If x^*_j is fractional for some $j \in I$, the problem is divided into two sub-problems. In the first sub-problem, the constraint $x_j \leq \lfloor x^*_j \rfloor$ is added. In the second sub-problem, the constraint $x_j \geq \lceil x^*_j \rceil$ is added. The LP relaxation is solved again at each child node, and the process repeats. Along the way, the subproblems are stored in a 'tree' structure.

If at any stage a feasible solution to the original problem is encountered, it becomes a candidate solution, and no further branching is performed at that node. Any candidate solution encountered is compared with the best candidate solution so far, and the best upper bound is recorded. Furthermore, any node with a (fractional) lower bound larger than the current upper bound is 'pruned' from the search space.

In the absolute worst case, branching continues until all solutions are enumerated. In practice, however, we expect branch-and-bound to perform much better. In the whole process, there are four ingredients that will impact the performance of any branch-and-bound algorithm (see [92] for a detailed survey):

1. Choice of relaxation: This is the 'bounding' part of the algorithm. Instead of the LP relaxation, other relaxations like those mentioned in section 1.4 can also be used (e.g. [17] use LR and [135] use SR to compute lower bounds).

2. Branching rule: the standard way to branch is as described above, but other rules may be used if desired.

3. Variable selection: which of the variables to branch on from the selected node (e.g., one could branch on the most fractional variable).

4. Node selection: which of the nodes to explore next. A simple strategy could be to pick an arbitrary child node each time, this would be a type of 'depth first' search. A more refined strategy may involve picking the node with the smallest lower bound.

Cutting-plane methods, first introduced by Gomory in [76], are another method for solving MILPs. Whereas branch-and-bound used linear constraints to 'partition' the relaxed feasible space at each branching operation, cutting-plane methods use linear constraints to 'cut' off fractional LP solutions, without excluding any feasible solutions from the original MILP. Gomory [76] proposed a process of strengthening the LP relaxation by adding a cutting plane obtained from the simplex tableau, and resolving the LP relaxation. If the new LP solution is feasible for the MILP, the procedure ends. If not, the LP is further strengthening by adding another cutting plane from the resultant simplex tableau. This is repeated until an integer solution is obtained.

The strength of a cutting plane depends on its 'depth', i.e., the proportion of the LP feasible region that it cuts off. In this context, it is helpful to consider the convex hull of the feasible solutions to the MILP. This convex hull is always a polyhedron. The strongest cutting planes are then the ones that define the facets of the convex hull. Unfortunately, the general purpose cutting planes in [76] tend to be weak in practice, which means that a high number of iterations is usually needed to reach the integral optimal solution. Fortunately, various special-purpose 'deep' cutting planes have been found for a range of COPs and MILPs (see, e.g., [38, 122]).

Finally, we mention that Padberg & Rinaldi [124] proposed the *branch-and-cut* approach, which fully integrates cutting planes with branch-and-bound. The idea is that cutting planes may be used to tighten the LP (and thereby improve the lower bound) at any branch of the enumeration tree. While hard to implement, branch-and-cut algorithms have proven very useful, to the point that they are now incorporated into many modern specialised software packages for MILP (such as `CPLEX`, `Gurobi` and `Xpress`).

1.6 Heuristic Methods

In practice, exact methods are not always used to solve MILPs or COPs. Indeed, while a sufficiently small instance of any $\mathcal{NP}$-hard problem can be solved in reasonable time by an exact method, solving a large-scale real-world instance to proven optimality may take a prohibitively large amount of time and/or memory. In such cases, it becomes necessary to resort to *heuristics.*

Heuristics, first introduced in [125], are methods that are intended to provide a feasible solution of 'sufficiently' good quality in an 'acceptable' amount of time. Whereas exact methods draw on concepts from mathematics, heuristics tend to draw on concepts from artificial intelligence and machine learning.

A nice feature of many heuristics is that they enable one to choose the 'trade-off' between solution quality and computing time. (For some time-critical applications, one may need a solution in less than a second, whereas for some strategic problems, one may be able to run a heuristic for hours or even days.) On the other hand, most heuristics do not provide a mechanism to evaluate the quality of the solutions that they find (but see the end of the next section for an important exception). In practice, therefore, the quality of a heuristic solution is often evaluated using past experience, or by comparing it to the status quo.

It is popular in the heuristics literature to make a distinction between *constructive* heuristics, which build a solution from scratch, and *improvement* heuristics, which start with a given solution and try to improve it. If we take the example of the 0-1 KP (introduced in Section 1.2), a simple constructive heuristic would be to start with an empty knapsack, and then insert items into the knapsack in non-increasing order of profit, until no more items can be inserted.

Nowadays, constructive heuristics are seldom used on their own, since they usually provide non-competitive solutions. They are however often used to provide initial solutions for improvement heuristics to work on.

The simplest kind of improvement heuristics are the so-called ‘local search’ or ‘neighbourhood search’ heuristics (see, e.g., [1]). Given a feasible solution to the problem, a set of similar solutions, called the ‘neighbourhood’ of the current solution, is defined. The neighbourhood is then searched in a systematic manner. If a neighbouring solution is found with lower cost than the current solution, we ‘move’ to the neighbouring solution, which then becomes the current solution. This procedure is repeated, in an iterative manner, until no more improvement is possible.

The choice of neighbourhood is related to the structure of the given optimisation problem. In fact, even for a given optimisation problem, more than one type of neighbourhood may exist. Taking the example of the 0-1 KP again, one simple neighbourhood could be to remove one item from the knapsack and replace it with another item. A different neighbourhood, that allows for changes in the total number of items in the knapsack, could be removing one item while inserting one or more different items.

A problem with local search heuristics is that they may become trapped at sub-optimal solutions (so-called 'local optima'). Several methods have been developed to mitigate against this problem. One option is simply to restart the heuristic many times, from many different random initial solutions. Another option is to switch to a different neighbourhood as soon as a local optimum is found for the current neighbourhood. Another is to occasionally accept a solution that does not improve the objective function, in the hope of finding better local optima later on.

There are more sophisticated general-purpose heuristics, known as *metaheuristics*, that extend the idea of local search with mechanisms to help them escape from local optima. Some of the most popular metaheuristics take inspiration from natural phenomena. *Simulated Annealing*, proposed by Kirkpatrick *et al.* in [101], drew parallels between the physical process of *annealing* (slowly cooling molten metals) and searching for good solutions to optimisation problems. The idea is to accept worst neighbouring solutions with a small probability, and then slowly decrease this probability over time. *Genetic Algorithms*, developed by Holland and others in the 1970s (see [91]), take inspiration from natural selection. They work with a *population* of solutions instead of a single solution. The idea is that pairs of 'parent' solutions produce 'child' solutions with similar characteristics. The probability that a child solution 'survives' to the next generation is inversely proportional to its cost.

Another popular metaheuristic is *Tabu Search*, developed by Glover [74]. In each iteration, it explores the entire neighbourhood of the current solution, and moves to the best neighbour, regardless of whether or not that neighbour improves on the current solution. To prevent the algorithm from returning to an already-visited solution, recent moves are stored in a 'tabu list'. Moves in the tabu list are forbidden, unless they lead to a solution that is better than the best found so far.

For a more detailed discussion about local search and meta-heuristics, the reader is referred to the textbooks [1, 69].

1.7 Heuristics Based on Relaxation and Duality

In addition to the approaches mentioned in the last section, there are also heuristic approaches that are based on relaxation and/or duality. In one such approach, the LP relaxation of an 0-1 LP is solved first, yielding a lower bound. The LP solution is then *rounded* to integers, in an intelligent way, with the hope of obtaining a feasible solution to the 0-1 LP (and therefore also an upper bound).

A good example of the LP rounding approach is the heuristic of Hochbaum [89] for SCP instances in which no element is contained in more than k sets or, equivalently, no row of the matrix A in the 0-1 LP (1.6) has more than k ones. After solving the LP relaxation, she takes the LP solution and rounds up to 1 each variable whose LP value is at least $1/k$. (The other variables are rounded down to 0.) The resulting heuristic solution is guaranteed to be feasible and have a cost no more than k times the optimum.

Hochbaum's heuristic is also a good example of an 'approximation algorithm'. An approximation algorithm is a heuristic that runs in polynomial time, yet is guaranteed to produce a solution whose cost is within a known factor of the optimum. For more details on approximation algorithms, see [140, 144]. For more on heuristics based on LP rounding, see Chapter 4 of [144].

A variant of the LP rounding approach, called *randomised rounding*, was developed by Raghavan and Tompson [129]. The idea is to set each variable to 1 independently at random, with a *probability* proportional to its value in the LP solution. Although the resulting solutions are not guaranteed to be feasible for the 0-1 LP, they can often be easily 'repaired' to make them feasible. For more on this approach, see Chapter 5 of [144].

Observe that, to obtain a valid lower bound on the optimal cost, it is not necessary to solve the LP relaxation to proven optimality. Indeed, any *feasible* solution to the *dual* of the LP yields a valid lower bound. This leads naturally to methods that use heuristics to construct 'reasonably good' primal and dual solutions simultaneously.

An early example of a 'primal-dual' heuristic is the 'dual ascent' heuristic of [21,51] for the Simple Plant Location Problem. The problem is formulated as a 0-1 LP of minimisation type. A greedy constructive heuristic is used to find a feasible solution to the dual of the LP relaxation. The dual solution is then used to compute an 'approximate reduced cost' for each facility. To obtain a heuristic solution to the 0-1 LP, the facilities with zero reduced cost are opened, and each client is assigned to the nearest open facility. This yields an upper bound.

Since the publication of [21, 51], dual ascent has been applied to many other problems, such as the SCP [5, 13, 59], the *Generalised Assignment Problem* [58], the *Uncapacitated Network Design Problem* [4], and the *Directed Steiner Tree Problem* [146]. There is also a more sophisticated version, called the *primal-dual schema*, which has been used to create approximation algorithms for several important combinatorial optimisation problems. For details, see [140, 144].

There are also heuristics based on *Lagrangian* relaxation. For example, Beasley [16] developed heuristics for several different facility location problems as follows. The problem is formulated as an 0-1 LP that contains constraints stating that each client must be served by exactly one facility. These constraints are relaxed in a Lagrangian way. The relaxed problem, which is a very simple 0-1 LP, is trivial to solve. Next, facilities that are open in the solution to the relaxed problem are opened. To complete the heuristic solution, each client is simply assigned to the nearest open facility. This procedure can be repeated for different choices of Lagrangian multipliers, if desired.

Another good example of a Lagrangian heuristic, this time for the SCP, is that of Caprara *et al.* [32]. The subgradient method is used to derive several near-optimal sets of Lagrangian multipliers. Each set of multipliers is used to generate an approximate reduced cost for each variable. To create the heuristic solution, all variables are initially set to 0, and variables are then iteratively set to 1, in non-decreasing order of approximate reduced cost, until a feasible solution is obtained.

We remark that all of the heuristics mentioned so far in this section have a desirable property that the heuristics in the previous section do not have: they produce both lower and upper bounds. They are also a special case of so-called 'matheuristics', which are heuristics that incorporate concepts from exact algorithms, such as relaxation, duality, branching, bounding or decomposition. For surveys on matheuristics, see [9, 25, 114, 115, 130].

1.8 Overview of Thesis

We are now in a position to explain the topics addressed in this thesis.

In Chapter 2, we present some theoretical results related to *surrogate* and *group* relaxation (see Section 1.4). We prove that (a) when only inequalities are surrogated, the surrogate dual is $\mathcal{NP}$-hard, but solvable in pseudo-polynomial time under certain conditions; (b) when equations are surrogated, the surrogate dual exhibits unusual complexity behaviour; (c) the group relaxation is $\mathcal{NP}$-hard for the 0-1 KP and its general-integer version; and (d) the group relaxation is *strongly* $\mathcal{NP}$-hard for another COP, known as the *Set Packing Problem.*

In Chapter 3, we examine in detail the surrogate dual for another COP, the *Multidimensional Knapsack Problem* or MKP. Although several authors have used surrogate relaxation to compute upper bounds for the MKP (e.g., [63, 68, 98, 134]), those authors solved the surrogate dual heuristically. We present an *exact* algorithm for the surrogate dual, along with a primal heuristic. Computational results show that our algorithms are fast, and that our primal heuristic yields good lower bounds. On the other hand, the upper bound from the dual tends to be only slightly stronger than the one from the LP relaxation.

In Chapter 4, we turn our attention to *matheuristics*, which we defined in the previous section. It turns out that, while there is already a substantial literature on matheuristics that draw on dual ascent, Lagrangian relaxation, Dantzig-Wolfe decomposition or Benders decomposition, very little has been published on matheuristics based on surrogate relaxation. To address this, we present surrogate-based matheuristics for three specific COPs: the MKP, the SPLP, and the so-called *Three-Dimensional Assignment Problem.* The computational results obtained are rather encouraging.

Chapter 5 is concerned with matheuristics that begin by constructing a feasible solution to the dual of the 0-1 LP. We show that such ‘dual-based’ matheuristics can exhibit highly counter-intuitive behaviour. In particular, for some problem classes, solving the dual exactly invariably leads to much worse primal solutions than solving the dual with a simple greedy heuristic. We provide a tentative explanation for this phenomenon, based on the concept of *primal degeneracy.* We use the SPLP and SCP as examples.

Finally, in Chapter 6, we summarise the contributions of this thesis, and make some concluding remarks.

Chapter 2

On the Complexity of Surrogate and Group Relaxation for Integer Linear Programs

2.1 Introduction

Many important $\mathcal{NP}$-hard problems have a natural formulation as an *integer linear program* or ILP (see, e.g., Conforti *et al.* [38]). To obtain a bound on the optimal value, one can solve the *continuous relaxation* of the ILP, which is obtained by permitting variables to take fractional values. The resulting bound can however be very weak in some cases. Popular ways to obtain stronger bounds include *cutting planes* (e.g., [39,119]), *Lagrangian relaxation* (e.g., [70,84]) and *Dantzig-Wolfe decomposition* (e.g., [11,46]).

Two further methods for obtaining strong bounds, which are less well known, are *surrogate* and *group* relaxation (see [75, 79] and [77, 78], respectively). The existing literature on these techniques leaves unanswered several natural questions concerned with computational complexity. In an attempt to address this gap, we show the following:

- For the case in which only inequalities may be surrogated, the surrogate dual is $\mathcal{NP}$-hard, but solvable in pseudo-polynomial time under certain conditions.

- For the case in which only equations may be surrogated, the surrogate dual exhibits unusual complexity behaviour: computing the bound is $\mathcal{NP}$-hard in the strong sense, but optimal multipliers can be found in polynomial time.
- The group relaxation is $\mathcal{NP}$-hard for the integer knapsack and 0-1 knapsack problems, and strongly $\mathcal{NP}$-hard for the set packing problem.

The paper has a simple structure. The literature is reviewed in Section 4.2. The subsequent three sections present the three theoretical results mentioned above. Some concluding remarks are made in Section 2.6.

Throughout the paper, we let n denote the number of variables, and let N denote $\{1, \ldots, n\}$. We call surrogate and group relaxation "SR" and "GR", respectively. Given a vector $v \in \mathbb{Q}^p_+$, we let $||v||_1$ denote $\sum_{i=1}^p v_i$. Given a rational scalar s, vector v or matrix M, we let "size(s)", "size(v)" and "size(M)" denote the number of bits needed to represent s, v or M, respectively. We assume that the reader is familiar with the basics of computational complexity theory, including ordinary and strong $\mathcal{NP}$-hardness, and pseudo-polynomial time (see [67]). Finally, we remind the reader that a function $f : S \mapsto \mathbb{R}$ with convex domain C is called *quasi-convex* if

$$f\big(\lambda x + (1-\lambda)y\big) \leq \max\big\{f(x), f(y)\big\} \qquad \big(\forall x, y, \in C,\ \lambda \in (0,1)\big).$$

2.2 Literature Review

In this section, we recall the key papers on SR and GR.

2.2.1 Surrogate relaxation

Consider an ILP of the form

$$\max\big\{c^T x :\ Ax \leq b,\ x \in \mathcal{X}\big\}, \tag{2.1}$$

where $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$, and

$$\mathcal{X} = \big\{x \in \mathbb{Z}^n_+ :\ Dx \leq e\big\}$$

for some integral matrix D and integral vector e. In SR, we pick a vector $\mu \in \mathbb{R}^m_+$ of *surrogate multipliers*, and solve the following simpler ILP [75, 79]:

$$\max\Big\{c^T x : \big(\mu^T A\big)x \le \mu^T b,\, x \in \mathcal{X}\Big\}. \tag{2.2}$$

This gives an upper bound, that we call $U(\mu)$.

Note that computing $U(\mu)$ is itself an ILP, and may even be $\mathcal{NP}$-hard. On the other hand, when $\mathcal{X}$ has a sufficiently simple structure, the ILP in question may be solvable reasonably quickly in practice. For example, if $\mathcal{X}$ is $\mathbb{Z}^n_+$ or $\{0,1\}^n$, then (2.2) is a knapsack problem, and can be solved in pseudo-polynomial time by dynamic programming [18].

The problem of finding the vector μ that gives the best upper bound is called the *surrogate dual.* Greenberg & Pierskalla [79] were the first to present theory related to the surrogate dual. They showed that the surrogate dual upper bound is at least as good as the one from LP relaxation. It is also shown that $U(\mu)$ is a quasi-convex function of μ. Glover [75] extended the previous work and formalised Surrogate Duality theory to make it directly comparable to Lagrangian relaxation. Glover generalised the concepts of perturbation functions and subgradients from Lagrangian duality to be applicable to surrogate duality and provided necessary and sufficient conditions for weak and strong duality in the surrogate case.

For heuristic methods for solving the surrogate dual see, e.g., [97,98,134]. Karwan & Rardin [97,98] showed some negative results regarding the possibility of determining optimal surrogate multipliers when 'direct search methods' are used. They proposed a heuristic approach to compute surrogate multipliers, by iteratively solving linear relaxations obtained from the solution of sub-optimal surrogate relaxations. Sarin *et al.* [134] proposed a modified sub-gradient approach for obtaining surrogate multipliers by solving a Lagrangian sub-problem. They do not however show that this procedure is capable of solving the surrogate dual to proven optimality.

Exact algorithms for solving the surrogate dual can be found in, e.g., [22, 23, 100]. Boros [23] was the first to propose an exact algorithm for finding optimal surrogate multipliers based on the ellipsoid method. Kim & Kim [100] extended the previous work of Sarin *et al.* [134], and proved convergence in their stopping rule for solving the Lagrangian subproblem and hence their overall problem. More recently. Boland *et al.* [22] incorporated the use of the so-called 'bundle trust region' method to improve search efficiency in the surrogate multiplier space.

Note that $U(s\mu) = U(\mu)$ for any positive scalar s. Accordingly, most authors impose the condition that $||\mu||_1 = 1$. We will *not* do this, however, for reasons which will become clear in Section 2.3.

We will also consider the case in which equations, rather than inequalities, are surrogated. For this situation, we will need a result of Glover & Woolsey [73]. It states that, if $Cx = d$ is a set of m *equations* in n binary variables, one can compute in polynomial time a non-negative integral vector μ, with coefficients bounded by $2^m \prod_{i=1}^{m}(|d_i| + 1)$, such that the single equation $(\mu^T C)x = \mu^T d$ has the same set of binary solutions.

2.2.2 Group relaxation

Now consider an ILP written in the slightly different form

$$\max\left\{c^T x : Ax \leq b,\, x \in \mathbb{Z}_+^n\right\}, \tag{2.3}$$

where $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$. Adding slack variables we obtain

$$\max\left\{c^T x : Ax + s = b,\, (x, s) \in \mathbb{Z}_+^{n+m}\right\}. \tag{2.4}$$

Suppose we solve the continuous relaxation of (2.4) by the simplex method, yielding a basic optimal solution (x^*, s^*). Exactly m variables will be basic and n variables will be non-basic at (x^*, s^*).

Gomory's GR is obtained from (2.4) by retaining integrality, but dropping the non-negativity restriction on the basic variables [77,78]. Provided that (x^*, s^*) is non-degenerate, this is equivalent to dropping all non-binding constraints in the original ILP (2.3), including all non-binding non-negativity constraints, if any. Thus, the upper bound obtained with GR is at least as good as the one obtained with LP relaxation.

The reason for the name *group relaxation* is that Gomory showed how to express it in group-theoretic terms. He also showed that the GR can be reduced to a shortest-path problem in a graph with D nodes, where D is the determinant of the basis matrix. This has led to several algorithms for solving the GR (see, e.g., [35, 132]). Unfortunately, none of them run in polynomial time.

It is stated in [110] that, for general ILPs, the GR is $\mathcal{NP}$-hard in the strong sense. An explicit proof is given in [54]. Interestingly, however, there are some $\mathcal{NP}$-hard ILPs whose GR can be solved in polynomial time. Indeed, Cornuéjols *et al.* [41] show that this is true for ILPs of the form

$$\max\Big\{p^T x :\ x_u + x_v \le 1 \,\big(\{u,v\} \in E\big),\ x \in \mathbb{Z}^n_+\Big\}. \tag{2.5}$$

where $p \in \mathbb{Z}^n_+$ and E is the edge set of an arbitrary (simple, loopless, undirected) graph on n nodes. Such ILPs are used to model the *independent set problem* (also known as the node packing problem or stable set problem), which is well-known to be strongly $\mathcal{NP}$-hard [96].

2.3 Surrogating Inequalities

In this section, we consider the computational complexity of the surrogate dual when only inequalities are surrogated. To begin, for any $\theta \in \mathbb{Z}$, let

$$\Lambda(\theta) = \left\{\mu \in \mathbb{R}^m_+ :\ U(\mu) \le \theta\right\}.$$

That is, $\Lambda(\theta)$ is the *lower level set* of $U(\mu)$ for the given θ. Observe that $\Lambda(\theta)$ is the set of points $\mu \in \mathbb{R}^m_+$ satisfying the following linear inequalities:

$$(A\bar{x} - b)^T \mu > 0 \qquad \big(\forall \bar{x} \in \mathcal{X} :\ c^T \bar{x} \ge \theta + 1\big). \tag{2.6}$$

Thus, $\Lambda(\theta)$ is a convex cone. On the other hand, it is not closed (except in the trivial cases when it is either empty or equal to $\mathbb{R}^m_+$). This makes it rather difficult to work with.

Now, consider the following modified version of the inequalities (2.6):

$$(A\bar{x} - b)^T \mu \ge 1 \qquad \big(\forall \bar{x} \in \mathcal{X} :\ c^T \bar{x} \ge \theta + 1\big). \tag{2.7}$$

Observe that, if a vector lies in $\Lambda(\theta)$, we can multiply it by a suitable positive scalar to make it satisfy (2.7). This leads us to define the following (unbounded) convex set:

$$\tilde{\Lambda}(\theta) = \left\{\mu \in \mathbb{R}^m_+ : \text{ (2.7) holds}\right\}.$$

From the above argument, $\Lambda(\theta)$ is empty if and only if $\tilde{\Lambda}(\theta)$ is empty. We also have the following result.

Proposition 2.1 *$\tilde{\Lambda}(\theta)$ is a polyhedron.*

Proof. Define the polyhedron

$$P^\theta = \left\{x \in \mathbb{R}^n_+ : Dx \le e, \, c^T\bar{x} \ge \theta + 1\right\},$$

and let P^θ_I denote the convex hull of the integral points in P. By Meyer's theorem [118], P^θ_I is a polyhedron. Thus, it can be described by a finite set of extreme points and extreme rays. Let $p^1, \dots, p^s$ be the points and $r^1, \dots, r^t$ be the rays. By definition, $\tilde{\Lambda}(\theta)$ is the set of all points in $\mathbb{R}^m_+$ that satisfy the following inequalities:

$$(Ap^k - b)^T\mu \ge 1 \qquad (k = 1, \dots, p) \tag{2.8}$$

$$(Ar^k)^T\mu \ge 0 \qquad (k = 1, \dots, r). \tag{2.9}$$

It is therefore the intersection of a finite number of half-spaces. $\square$

The above concepts are made clear by the following example:

Example 1: Consider the following trivial ILP:

$$\max\left\{x_1 + x_2 : \; 6x_1 + 3x_2 \le 4, \; 3x_1 + 6x_2 \le 4, \; x \in \{0,1\}^2\right\}.$$

The optimal solution is $(0,0)$, with profit 0. The optimal solution to the LP relaxation is $(4/9, 4/9)$, giving the upper bound $8/9$.

Suppose we set $\mathcal{X}$ to $\{0,1\}^2$, and surrogate both of the linear constraints, with multipliers μ_1 and μ_2. Suppose also that we set θ to 0. There are three points in $\mathcal{X}$ with profit greater than 0; namely, $(0,1)$, $(1,0)$ and $(1,1)$. The corresponding constraints (2.6) are $-\mu_1 + 2\mu_2 > 0$, $2\mu_1 - \mu_2 > 0$ and $5\mu_1 + 5\mu_2 > 0$, respectively. The last of these is redundant. Therefore:

$$\begin{aligned}
\Lambda(0) &= \big\{\mu \in \mathbb{R}^2_+ : -\mu_1 + 2\mu_2 > 0,\, 2\mu_1 - \mu_2 > 0\big\} \\
\tilde{\Lambda}(0) &= \big\{\mu \in \mathbb{R}^2_+ : -\mu_1 + 2\mu_2 \geq 1,\, 2\mu_1 - \mu_2 \geq 1\big\}.
\end{aligned}$$

One can check that $\tilde{\Lambda}(0)$ has the unique extreme point $(1, 1)$. Thus, SR with this choice of μ yields the upper bound 0. $\square$

We will also need the following lemma.

Lemma 2.1 *The number of bits needed to represent any inequality of the form (2.8) or (2.9) is polynomial in n, size(c), size(D), size(e) and size(θ).*

Proof. This follows from [136], Corollary 17.1c, and the definition of the p^k and r^k. $\square$

Next, consider the following decision problem associated with the surrogate dual:

The Level Set Problem (LSP): *Given some $\theta \in \mathbb{Z}$, is $\Lambda(\theta)$ empty?*

It follows from the above observations that the answer to the LSP is "yes" if and only if the following LP is infeasible:

$$\min \big\{ ||\mu||_1 : \text{ (2.7) hold, } \mu \in \mathbb{R}^m_+ \big\}. \tag{2.10}$$

We remark that, if $\mathcal{X}$ has infinite cardinality, then (2.10) is a *semi-infinite* LP.

Clearly, solving the LSP cannot be harder than solving the surrogate dual. On the other hand, we have the following negative result.

Proposition 2.2 *Even when only inequalities are surrogated, the LSP is $\mathcal{NP}$-complete.*

Proof. Consider the special case of the ILP (2.1) in which $m = 1$, $A \geq 0$, $b > 0$ and $\mathcal{X} = \{0, 1\}^n$. In this case, the ILP reduces to the 0-1 knapsack problem. Moreover, by setting μ_1 to any positive quantity, we can make the upper bound $U(\mu)$ exact (i.e., equal to the profit of the optimal solution of the knapsack problem). So, the answer to the LSP is “yes” if and only if there exists a solution to the knapsack problem with profit larger than θ. This is $\mathcal{NP}$-hard to check.

To complete the proof, we just have to show that the LSP is in $\mathcal{NP}$. For a given θ, let S be the system of linear inequalities formed by the union of (2.8) and (2.9). By definition, the answer to the LSP is “yes” if and only if S is inconsistent. Moreover, if S is inconsistent, then, by Helly’s theorem [88], there exists a subset of S of cardinality at most $m + 1$ that is also inconsistent. Moreover, by Lemma 2.1, each of the $m+1$ inequalities involves a polynomially bounded number of bits. Thus, whenever the answer to the LSP is ‘yes’, there exists a short certificate of that fact. This shows that the LSP is in $\mathcal{NP}$. $\square$

We now return to the surrogate dual itself.

Theorem 2.1 *Consider once more an ILP of the form (2.1). Suppose that the following three assumptions hold:*

1. *For any $\theta \in \mathbb{Z}$ and $v \in \mathbb{Q}^n$, one can solve the following ILP in time that is polynomial in n, size(D), size(e), size(θ), size(v) and $||c||_1$:*
$$\min\Big\{v^T x : x \in \mathcal{X},\ c^T x \geq \theta\Big\}.$$
2. *The continuous relaxation of the ILP (2.1) is feasible and bounded.*
3. *One can compute in polynomial time a lower bound L on the optimal profit, whose encoding length is polynomial in n, size(A), size(b), size(c), size(D) and size(e).*

Then one can solve the surrogate dual in time that is polynomial in n, size(A), size(b), size(D), size(e) and $||c||_1$.

Proof. First, suppose that $\theta \in \mathbb{Z}$ is fixed. Consider the *separation problem* for the constraints (2.7) (i.e., the problem of detecting when a given $\bar{\mu} \in \mathbb{R}^m_+$ violates one of the constraints). The separation problem is equivalent to the ILP

$$\min\Big\{(\bar{\mu}^T A)x :\ x \in \mathcal{X},\ c^T x \geq \theta + 1\Big\}.$$

(Indeed, one of the linear constraints is violated if and only if the optimal solution of this ILP, say $\bar{x}$, satisfies $\big(\bar{\mu}^T A\big)\bar{x} < 1 + \bar{\mu}^T b$.) Now, under the first assumption, the separation problem can be solved in time that is polynomial in the parameters listed. Then, by Lemma 2.1 and the polynomial equivalence of separation and optimisation [80], the LP (2.10) can be solved in time that is polynomial in n, size(A), size(b), size(D), size(e), size(θ) and $||c||_1$. This implies in turn that the LSP can be solved in time that is polynomial in the same parameters.

Now suppose that assumption 2 holds. Using the ellipsoid method [80, 136], we can solve the LP relaxation of the ILP (2.1) in time that is polynomial in n, size(A), size(b), size(c), size(D) and size(e). This yields an upper bound on the optimal profit, which we denote by U. Moreover, from [136], Theorem 10.3, size(U) is bounded by a polynomial in the same parameters.

Now suppose that assumption 3 also holds. We now have both lower and upper bounds on the optimal value of θ, each of which has polynomial encoding length. Thus, by applying binary search on the value of θ, one can reduce the surrogate dual to a number of LSP instances that is polynomial in the stated parameters. $\square$

This result generalises a result of Boros [23], who proved it for the case in which $\mathcal{X}$ is the hypercube. It means that, if we can solve the surrogate relaxed problem in pseudo-polynomial time, then the surrogate dual cannot be $\mathcal{NP}$-hard in the *strong* sense (unless of course $\mathcal{P} = \mathcal{NP}$).

We remark that computing L is often easy in practice. Indeed, if it is known that all points $\bar{x} \in \mathcal{X}$ satisfy $\ell \leq \bar{x} \leq u$, for some vectors $\ell, u \in \mathbb{Z}^n_+$, then an easily-computable value for L is

$$\sum_{j=1}^{n} \ell_j \max\{0, c_j\} + \sum_{j=1}^{n} u_j \min\{0, c_j\}.$$

2.4 Surrogating Equations

Now we turn our attention to the case in which equations, rather than inequalities, are surrogated. More precisely, we now assume that the ILP takes the form:

$$\max \left\{ c^T x : \; Ax = b, \; x \in \mathcal{X} \right\}.$$

Similarly, we now assume that

$$U(\mu) = \max \left\{ c^T x : \; \left(\mu^T A\right) x = \mu^T b, \; x \in \mathcal{X} \right\}.$$

Note also that, here, it may make sense to allow some of the surrogate multipliers to take negative values.

As in the previous section, we can define lower level sets:

$$\Lambda(\theta) = \left\{ \mu \in \mathbb{R}^m : \; U(\mu) \leq \theta \right\}.$$

Note however that, in this case, we have:

$$\Lambda(\theta) = \left\{ \mu \in \mathbb{R}^m : \; \left(\mu^T A\right) \bar{x} \neq \mu^T b \; \left(\forall \bar{x} \in \mathcal{X} : \; c^T x > \theta\right) \right\}.$$

The presence of the symbol "$\neq$" suggests that $\Lambda(\theta)$ is not convex in general. This is indeed the case. In fact, $\Lambda(\theta)$ can even be disconnected, as shown by the following example.

Example 2: Consider the following equality-constrained 0-1 LP:

$$\max \left\{ x_1 + 5x_2 : \; x_1 = 0, \; x_1 + x_2 = 0, \; x \in \{0,1\}^2 \right\}.$$

Trivially, the optimal solution has profit 0. Now, as before, we set $\mathcal{X}$ to $\{0,1\}^2$ and surrogate both of the linear constraints. Note that the domain of μ is now $\mathbb{R}^2$. One can check that

$$\begin{aligned} U(\mu) &= 6 \quad (\text{if } \mu_1 + 2\mu_2 = 0) \\ &= 5 \quad (\text{if } \mu_2 = 0 \text{ and } \mu_1 \neq 0) \\ &= 1 \quad (\text{if } \mu_1 + \mu_2 = 0 \text{ and } \mu_2 \neq 0) \\ &= 0 \quad (\text{otherwise}). \end{aligned}$$

So the lower level sets $\Lambda(1), \ldots, \Lambda(6)$ are all disconnected. $\square$

Now recall that, in the inequality-constrained case, we reduced the Level Set Problem to the LP (2.10). The equivalent problem in the equality-constrained case is:

$$\min \left\{ ||\mu||_1 : \ \left(A\bar{x}\right)^T \mu \neq b^T \mu \ \left(\forall \bar{x} \in \mathcal{X} : \ c^T \bar{x} > \theta\right), \ \mu \in \mathbb{R}^m \right\}. \tag{2.11}$$

Note that (2.11) looks much harder to solve than (2.10). Indeed, we have the following negative result.

Theorem 2.2 *If only equations are surrogated, then the surrogate dual is $\mathcal{NP}$-hard in the strong sense, and the LSP is $\mathcal{NP}$-complete in the strong sense.*

Proof. Suppose that $\mathcal{X} = \{0,1\}^n$, i.e., we are dealing with a 0-1 LP. By using a slack variable s_j, we can convert the condition $x_j \in \{0,1\}$ to $x_j + s_j = 1$, $x_j, s_j \in \mathbb{Z}_+$. In this way, we can transform a 0-1 LP into an ILP in which all constraints (apart from the non-negativity and integrality constraints) are equations. The result of Glover & Woolsey [73] mentioned in Subsection 5.2.1 then implies that there exists an integral vector μ, whose encoding length is polynomial in that of the input, such that $U(\mu)$ is equal to the profit of the optimal solution of the original 0-1 LP. Thus, solving the surrogate dual problem is as hard as solving a 0-1 LP, and is therefore strongly $\mathcal{NP}$-complete. Moreover, the answer to the LSP is "yes" if and only if there is a feasible solution to the 0-1 LP with profit greater than θ. For 0-1 LPs in general, checking whether such a feasible solution exists is strongly $\mathcal{NP}$-complete. $\square$

Thus, in the equality-constrained case, one cannot determine the best possible value of $U(\mu)$ in pseudo-polynomial time (unless $\mathcal{P} = \mathcal{NP}$). Bizarrely, however, one can find an optimal vector μ in polynomial time (by applying the procedure in [73]). The explanation of this apparent paradox is that the components of μ may be exponentially large, which makes solving the surrogate relaxed problem as hard as solving the ILP itself.

2.5 Group Relaxation

In this section, we prove some results concerned with the complexity of GR. We start with two results concerned with knapsack problems (KPs). We remind the reader that the integer KP takes the form

$$\max \left\{ p^T x : \, a^T x \leq b, \, x \in \mathbb{Z}_+^n \right\}, \tag{2.12}$$

where $p, a \in \mathbb{Z}_+^n$ and b is a positive integer. The 0-1 KP is similar, except that all variables are constrained to be binary. Both problems are known to be weakly $\mathcal{NP}$-hard [96, 113]. Our results are as follows.

Proposition 2.3 *The GR is $\mathcal{NP}$-hard for the integer KP.*

Proof. Consider an integer KP instance of the form (2.12). Let M be a "large" positive integer. (In fact, it suffices to set M to any integer that is larger than $(b+1)\max_{j \in N}\{p_j/a_j\}$.) We construct a different integer KP instance, which we call the *augmented* KP:

$$\begin{array}{rc} \max & p^T x + My \\ \text{s.t.} & a^T x + (b+1)y \leq 2b+1 \\ & (x, y) \in \mathbb{Z}_+^{n+1}. \end{array} \tag{2.13}$$

Note that the LP relaxation of the augmented KP has a unique optimal solution with $x_j^* = 0$ for all j, and $y^* = (2b+1)/(b+1) < 2$. This solution is non-degenerate, since the only binding constraints are the non-negativity constraints on the x variables and the constraint (2.13). Thus, applying GR to the augmented KP just means dropping non-negativity on y. Since y has a very large profit, it will be set to 1 in the optimal GR solution. So the optimal x for the GR of the augmented KP is identical to the optimal x of the original integer KP. □

Proposition 2.4 *The GR is $\mathcal{NP}$-hard for the 0-1 KP.*

Proof. Consider again an integer KP instance of the form (2.12), and let M be as in the previous proof. We construct the following 0-1 KP with $n+2$ variables:

$$\begin{aligned} \max \quad & p^T x + 2My + Mz \\ \text{s.t.} \quad & a^T x + (b+1)(y+z) \le 2b+1 \\ & (x, y, z) \in \{0,1\}^{n+2}. \end{aligned} \tag{2.14}$$

The LP relaxation of the 0-1 KP has an optimal solution with $x_j^* = 0$ for all j, $y^* = 1$ and $z^* = b/(b+1) < 1$. This solution is non-degenerate, since the only binding constraints are the non-negativity constraints on the x variables, the constraint (2.14), and the upper bound of 1 on y. Thus, applying GR to the 0-1 KP means dropping the upper bounds of 1 on the x_j, the lower bound of 0 on y, and the lower and upper bounds on z. Since y has a very large profit, it will be set to 1 in the optimal GR solution. This forces z to take the value 0. So the optimal x for the GR of the 0-1 KP is identical to the optimal x of the original integer KP. $\square$

Next, we consider the *set packing problem* (SPP). In the SPP, we are given a positive integer n, a positive integer profit p_j for each $j \in N$, and a collection of m non-empty subsets of N, say $S_1, \ldots, S_m$. The task is to select a subset of N of maximum profit, subject to the constraint that, for $i = 1, \ldots, m$, at most one element of S_i is selected. The SPP is strongly $\mathcal{NP}$-hard [96]. The standard ILP formulation is as follows [123]:

$$\max \Big\{ p^T x : \sum_{j \in S_i} x_j \le 1 \; (i = 1, \ldots, m), \; x \in \mathbb{Z}_+^n \Big\}. \tag{2.15}$$

Theorem 2.3 *The GR is $\mathcal{NP}$-hard in the strong sense, even for set packing problems that are formulated as ILPs of the form (2.15).*

Proof. Consider an SPP of the form (2.15), and let M be a "large" positive integer. (Here, it suffices to set M to $2\sum_{j \in N} p_j$.) We construct an augmented SPP with

$n + 3m$ variables. It takes the form:

$$
\begin{aligned}
\max \quad & p^T x + M \textstyle\sum_{i=1}^m \big(y_i + y'_i\big) + (M+1) \textstyle\sum_{i=1}^m z_i \\
\text{s.t.} \quad & \textstyle\sum_{j \in S_i} x_j + y_i + y'_i \le 1 && (i = 1, \dots, m) && (2.16) \\
& y_i + z_i \le 1 && (i = 1, \dots, m) && (2.17) \\
& y'_i + z_i \le 1 && (i = 1, \dots, m) && (2.18) \\
& x \in \mathbb{Z}_+^n \\
& y, y', z \in \mathbb{Z}_+^m.
\end{aligned}
$$

The LP relaxation of the augmented SPP has a feasible solution with $x_j = 0$ for all j, and $y_i = y'_i = z_i = 1/2$ for all i. This LP solution is optimal, as one can verify by setting the dual variables for (2.16) to $(M-1)/2$, and the dual variables for (2.17) and (2.18) to $(M+1)/2$. In fact, it is the unique optimal solution, as one can verify by noting that every x variable has a positive reduced cost.

Now note that, at this LP solution, the binding constraints are (2.16)–(2.18), together with the non-negativity constraints on the x variables. Thus, exactly $n+3m$ constraints are binding, and the LP solution is non-degenerate. Applying GR to the augmented SPP therefore means just dropping non-negativity on the y, y' and z variables. Now, note that the remaining constraints imply that $y_i + y'_i + z_i \le 1$ for all i. Since z_i has a larger profit than y_i and y'_i, it will be set to 1 in the optimal GR solution. This in turn will force y_i and y'_i to take the value 0. Then, the optimal x of the GR of the augmented SPP is identical to the optimal x of the original SPP. $\square$

We close this section by noting that the complexity of GR for a particular combinatorial optimisation problem depends on the way it is formulated as an ILP. Indeed, Padberg [123] showed that the SPP and the independent set problem are equivalent, i.e., any ILP of the form (2.15) can be converted into one of the form (2.5) and vice-versa. Yet, as mentioned in Subsection 5.2.2, the GR of (2.5) can be solved in polynomial time.

2.6 Final Remark

We close the paper by mentioning some interesting open problems concerned with *composite* relaxation (CR), which is a hybrid of Lagrangian and surrogate relaxation [75, 79]. In CR, we choose vectors $\lambda, \mu \in \mathbb{R}^m_+$, and solve the relaxed problem

$$\max \Big\{ c^T x + \lambda^T \big(b - Ax\big) : \ \big(\mu^T A\big)x \le \mu^T b, \ x \in \mathcal{X} \Big\}.$$

In theory, CR can produce better upper bounds than both Lagrangian and surrogate relaxation. Unfortunately, the dual function is not even quasi-convex in λ and μ. In fact, it may have local minima that are not global minima [97]. Thus, the composite dual is even harder to solve than the surrogate dual. We conjecture that, even in the inequality case, the composite dual is strongly $\mathcal{NP}$-hard. As for the Level Set Problem in CR, we do not even know whether it lies in $\mathcal{NP}$ or co-$\mathcal{NP}$.

Acknowledgement: We are grateful to an anonymous reviewer, whose comments enabled us to improve the paper significantly.

Chapter 3

Revisiting Surrogate Relaxation for the Multidimensional Knapsack Problem

3.1 Introduction

The *multidimensional knapsack problem* (MKP) is a classic problem in combinatorial optimisation. It is defined as follows. We have n items and m resources. The profit of item j is p_j. The amount of resource i available is b_i. Item j uses a_{ij} units of resource i. The goal is to select a set of items of maximum total profit, while respecting the availabilities of the resources.

There is a vast literature on the MKP. For reviews of the literature up to 2004 or so, see the surveys [61, 62, 99]. For recent examples of exact algorithms, heuristics and upper bounds, see [116, 142], [3, 47, 128] and [82, 95], respectively.

In the 1980s, several authors proposed to compute upper bounds for the MKP using a method called *surrogate relaxation* [63, 68, 98, 134]. To obtain a strong bound with surrogate relaxation, one must solve an auxiliary optimisation problem, called the *surrogate dual*. It is shown in [23] that the dual can be solved exactly in pseudo-polynomial time. Nevertheless, in the four above-mentioned papers, the dual was solved only approximately, using variations of the subgradient method.

In this paper, we present an algorithm for solving the surrogate dual exactly. The algorithm exploits the fact that excellent simplex-based linear programming solvers are now available. We then show how to use the information generated during the solution of the dual to drive a "primal heuristic", and thereby obtain lower bounds for the MKP. We also report extensive computational results, obtained by applying our algorithms to benchmark instances.

The paper has the following structure. The literature is reviewed in Section 3.2. The algorithm for the dual is described in Section 3.3, and the primal heuristic is given in Section 3.4. The computational results are in Section 3.5. Finally, some concluding remarks are made in Section 3.6.

Throughout the paper, we assume that the reader is familiar with the basics of linear, integer and dynamic programming (see [36, 38]). We write "LP", "DP" for *linear program* and *dynamic program*, respectively. We write "SR" and "SD" for *surrogate relaxation* and *surrogate dual*, respectively. We let N denote $\{1, \ldots, n\}$. We assume that the p_j, b_i and a_{ij} are positive integers. Finally, given a vector $v \in \mathbb{R}^p_+$, we let $|v|$ denote $\sum_{i=1}^p v_i$.

3.2 Literature Review

We now review the literature. For brevity, we mention only papers of direct relevance.

3.2.1 The MKP

The MKP is $\mathcal{NP}$-hard in the strong sense [67], but solvable in pseudo-polynomial time for fixed m [99]. It has a natural formulation as a 0-1 LP. For all $j \in N$, define a binary variable x_j, taking the value 1 if and only if item j is selected. We then have:

$$\max \left\{ p^T x : \, Ax \leq b, \, x \in \{0,1\}^n \right\}. \tag{3.1}$$

The *LP relaxation* of the 0-1 LP is obtained by replacing the binary condition with the weaker condition $x \in [0,1]^n$. This LP can be solved extremely quickly in practice. We will let $x^* \in [0,1]^n$ denote the LP solution, U_{LP} the corresponding upper bound, $\pi \in \mathbb{Q}_+^m$ the vector of dual prices, and $\rho \in \mathbb{Q}_+^n$ the vector of reduced costs.

Several exact and heuristic algorithms for the MKP attempt to exploit information in the LP solution, by giving "priority" to variables with large x_j^* and/or small ρ_j (e.g., [3, 47, 142]). Angelelli *et al.* [3] use such a criterion to construct an initial "kernel" of promising items in their adaptation of the kernel search heuristic to the MKP. Della Croce & Grosso [47] use the reduced costs to calculate "penalties" for each item. They fix the variables with the largest penalties to either 0 or 1, and solve the reduced problem exactly with an ILP solver. Vimont *et al.* [142] branch on the variables with the largest absolute reduced costs in their "implicit enumeration scheme". They also generate useful additional linear constraints using "reduced-cost propagation".

3.2.2 Surrogate relaxation and duality

Now consider an arbitrary 0-1 LP of the form (3.1), where negative coefficients are permitted. In SR, we pick a vector $\mu \in \mathbb{Q}_+^m$ of *surrogate multipliers*, and solve the following simpler 0-1 LP [75, 79]:

$$\max\left\{p^T x : \left(\mu^T A\right)x \le \mu^T b,\ x \in \{0,1\}^n\right\}. \tag{3.2}$$

We will let $U(\mu)$ denote the resulting upper bound.

The SD is the problem of finding the vector μ that minimises $U(\mu)$. It is shown in [79] that $U(\mu)$ is a quasi-convex function of μ. This fact is used in various algorithms for the SD (e.g., [22, 97, 98, 100, 134]), most of which are variants of the subgradient method. Karwan & Rardin [97, 98] showed some negative results regarding the possibility of determining optimal surrogate multipliers when "direct search" methods are used. They proposed a heuristic approach to compute surrogate multipliers, by iteratively solving linear relaxations obtained from the solution of suboptimal surrogate relaxations. Sarin *et al.* [134] proposed a modified subgradient approach for obtaining surrogate multipliers, which involves the solution of Lagrangian subproblems. They did not however show that this procedure is capable of solving the surrogate dual to proven optimality. Kim & Kim [100] extended the previous work of Sarin *et al.* [134], and proved convergence in their stopping rule for solving the Lagrangian subproblem and hence their overall problem. More recently, Boland *et al.* [22] incorporated the use of the so-called “bundle trust region” method to improve search efficiency in the surrogate multiplier space.

It has been proved that, under certain conditions, the SD can be solved in pseudo-polynomial time [23, 48]. However, the proofs in [23, 48] rely on the equivalence of separation and optimisation, which in turn relies on the ellipsoid method [80]. Given that the ellipsoid method is very slow in practice, the results in [23, 48] are of theoretical interest only.

Now recall the definition of π from the previous subsection. It was shown in [79] that $U(\pi) \le U_{LP}$. Thus, the upper bound from the SD is at least as strong as the one from the LP.

3.2.3 Surrogate relaxation for the MKP

Observe that, if we apply SR to the MKP, all coefficients are non-negative in (3.2). Thus, in this case, (3.2) is a *0-1 knapsack problem* (0-1 KP). Although the 0-1 KP is itself $\mathcal{NP}$-hard [96], it is often easy in practice [99]. Moreover, if $\mu \in \mathbb{Z}^m_+$, then (3.2) can be solved in $O\big(n\,(\mu^T b)\big)$ time, by DP [18]. For these reasons, SR may be an attractive option for the MKP.

In the 1980s, several papers applied SR to the MKP (e.g., [63, 68, 98, 134]). Gavish & Pirkul [68] provided an ϵ-approximate algorithm to determine the surrogate multipliers for the special case in which there are only two constraints. They observed that, in this case, the search for the surrogate multipliers can be reduced to an optimisation problem involving a single variable. They then used their algorithm for the bi-dimensional case within a heuristic method for the multi-dimensional case. They avoided solving 0-1 KP by solving the LP relaxation of the surrogate dual instead. Fréville & Plateau [63] provided an exact method for determining the surrogate multipliers in the bi-dimensional case, based on the solution of a sequence of real-weighted 0-1 KP instances. Finally, the authors of [98, 134] tested their general-purpose SD algorithms on the MKP.

In all these early papers, the SD was either solved for only the two-constraint version, or heuristically, via iterative approaches. The resulting upper bounds were good, but the test instances were rather small by today's standards (see again [47, 116]).

Crama & Mazzola [42] proved two negative results concerning the application of SR to the MKP. The first is that, for any $\mu \in \mathbb{Q}_+^m$, we have $U(\mu) \geq U_{LP}/2$. The second is that, again for any $\mu \in \mathbb{Q}_+^m$, we have $U(\mu) \geq U_{LP} - p^+$, where p^+ is the maximum of p_j over all $j \in N$. Intuitively, these results show that the upper bound from the SD cannot be *much* stronger than one from the LP.

3.3 Solving the Surrogate Dual

As mentioned above, the exact algorithms in [23, 48] for solving the SD are of theoretical interest only, since the ellipsoid method is very slow in practice. Given the fact that excellent simplex-based LP solvers are now available, we consider using a simplex-based method instead.

Our starting point is the following observation, made in [48]. For any given $\theta \in \mathbb{Z}_+$, we can check whether there exists a vector $\mu \in \mathbb{Q}_+^m$ such that $U(\mu) \leq \theta$ by solving the following LP:

$$\min \quad |\mu|_1 \tag{3.3}$$

$$\text{s.t.} \quad (A\bar{x} - b)^T \mu \geq 1 \quad \big(\forall \bar{x} \in \{0,1\}^n : p^T \bar{x} > \theta\big) \tag{3.4}$$

$$\mu \in \mathbb{R}_+^m. \tag{3.5}$$

Indeed, if this LP is feasible, the solution gives us the desired μ vector. If the LP is infeasible, it means that no such μ vector exists.

We call the LP (3.3)-(3.5) the "master" LP. Note that the number of constraints (3.4) is typically exponential in n. Thus, to solve the master LP (for a given θ), one must use a cutting-plane algorithm. As mentioned above, our cutting-plane algorithm uses the simplex method.

In each iteration of our cutting-plane algorithm, the solution to the master LP yields a "tentative" multiplier vector, say $\bar{\mu}$. The problem of checking whether $\bar{\mu}$ satisfies all of the constraints (3.4) is called the *separation problem*. (See [80] for a thorough treatment of separation problems in convex and combinatorial optimisation.)

Intuitively, the separation problem attempts to find a binary vector $\bar{x}$ which has large profit, but satisfies the current "tentative" surrogate constraint. The separation problem can itself be formulated as the following 0-1 LP:

$$\min \Big\{ \big(\bar{\mu}^T A\big) x : \; x \in \{0,1\}^n, \; p^T x > \theta \Big\}. \tag{3.6}$$

Let $\bar{x}$ denote the optimal solution of this 0-1 LP (for a given $\bar{\mu}$). There are three cases:

- Case 1: $\big(\bar{\mu}^T A\big)\bar{x} \geq 1 + \bar{\mu}^T b$ and $\bar{x}$ is feasible for the MKP instance. In this case, we know that the profit of the optimal MKP solution is at least $p^T \bar{x} > \theta$. Thus, our trial value θ was too small.

- Case 2: $\big(\bar{\mu}^T A\big)\bar{x} \geq 1 + \bar{\mu}^T b$ and $\bar{x}$ is not feasible for the MKP instance. In this case, the multiplier vector $\bar{\mu}$ satisfies $U(\bar{\mu}) \leq \theta$.

- Case 3: $\big(\bar{\mu}^T A\big)\bar{x} < 1 + \bar{\mu}^T b$. In this case, the corresponding constraint (3.4) is violated by $\bar{\mu}$.

The problem (3.6) is a 0-1 KP (of minimisation type). It is not hard to show that it can be solved in $O(n\theta)$ time, by DP. Note the profit of the optimal MKP solution cannot exceed $|p|$. Thus, we can assume without loss of generality that $\theta \leq |p|$. As a result, we can solve (3.6) in $O\big(n\,|p|\big)$ time. Note that this running time is pseudo-polynomial.

The above-mentioned cutting-plane algorithm takes a given value of $\theta \in \mathbb{Z}_+$ as input, but we do not know the optimal value of θ in advance. Thus, we perform a binary search on θ, solving the LP (3.3)-(3.5) in each major iteration.

Let μ^* denote the (as yet unknown) optimal solution to the SD, and let $\theta^* = U(\mu^*)$ denote the corresponding upper bound (also unknown) for the MKP. For the binary search, we need initial lower and upper bounds on θ^*. We will call these ℓ and u. To obtain ℓ, we simply run a greedy heuristic for the MKP (inserting items in non-increasing order of profit), and then set ℓ to the profit of the resulting MKP solution. As for u, we simply use $\lfloor U_{LP} \rfloor$.

The overall approach is described in Algorithm 1. Note that, towards the end of the outer "while" loop, we set θ to $\lfloor 0.9u + 0.1\ell \rfloor$ rather than to $\lfloor (u + \ell)/2 \rfloor$. This is because, in our experience, θ^* tends to be closer to u than to ℓ.

We remark that the algorithm as stated returns only the upper bound θ^*. One can easily modify the algorithm so that it also returns the optimal multiplier vector μ^*, and/or the associated x vector. We omit details for brevity.

3.4 Primal Heuristic

Observe that, each time we solve the separation problem (3.6), we obtain a vector $\bar{x} \in \{0,1\}^n$. Typically, $\bar{x}$ is not a feasible solution to the MKP. Our primal heuristic attempts to "repair" $\bar{x}$, to make it feasible for the MKP.

Before running the heuristic for the first time, we sort the items in order of "attractiveness" and create a sorted list. In more detail, when we solve the LP relaxation at the start of Algorithm 1, we store the vectors x^* and ρ. We then sort the items in N according to two criteria. First, we sort them in non-increasing order of x^*_j value. Then, if there are any items with $x^*_j = 0$, we sort those in non-decreasing order of ρ_j. We call the sorted list "`SL`".

Our primal heuristic is described in Algorithm 2. The idea is that, instead of relying entirely on the list `SL`, we give priority to items that have $\bar{x}_j = 1$. Since the separation problem is solved many times during Algorithm 1, this gives our primal heuristic several chances to find a good solution. Of course, the best primal solution found so far is stored in memory.

We remark that, if Algorithm 2 obtains a solution whose profit is larger than ℓ, we can update ℓ in Algorithm 1. This can be accomplished by adding the following line to Algorithm 1, immediately after the cutting-plane phase: "if the primal heuristic

Algorithm 1: Solving the Surrogate Dual of the MKP

```
input: positive integers $m$, $n$; non-negative integer matrix $A$;
       positive integer vectors $p$ and $b$.
// initialisation
Solve the LP relaxation of the MKP and set $u$ to $\lfloor U_{LP} \rfloor$;
Run a greedy heuristic for the MKP to get initial lower bound $\ell$;
Set up initial "master" LP min $\{|\mu| : |\mu| \geq 1, \ \mu \in \mathbb{R}^m_+\}$;
Solve the master LP via primal simplex;
Set $\theta$ to $u$;
// binary search
while $u \neq \ell$ do
    // cutting-plane algorithm
    while the master LP is feasible do
        Let $\bar{\mu}$ be the current solution to the master LP;
        Solve the 0-1 LP (3.6), yielding a solution $\bar{x}$;
        if $(\bar{\mu}^T A)\bar{x} < 1 + \bar{\mu}^T b$ then
            // we have found a violated constraint
            Add the constraint (3.4) to the master LP;
            Re-optimise the master LP via dual simplex;
        else
            Break;
        end
    end
    // check whether $\theta^* \leq \theta$
    if the master LP remains feasible then
        Set $u$ to $\theta$;
        Clean the master LP by deleting all cuts with positive slack;
    end
    else
        Set $\ell$ to $\theta + 1$;
        Let $S$ denote the set of newly-added cutting-planes;
        Clean the master LP by deleting the cutting planes in $S$;
    end
    Set $\theta$ to $\lfloor 0.9u + 0.1\ell \rfloor$;
end
Output $\theta$ and stop.
```

Algorithm 2: Primal Heuristic

```
input  : positive integers $m$, $n$; positive integer vectors $p$ and $b$,
         non-negative integer matrix $A$, sorted list SL,
         current solution $\bar{x}$ of the subproblem (3.6).
output: feasible MKP solution $S \subset N$.
Let $N^1$ be the set of items for which $\bar{x}_j = 1$;
Let $N^2 = N \setminus N^1$;
Set $S = \emptyset$;
for $k = 1, 2$ do
    for $t = 1$ to $|N^k|$ do
        Let $j$ be the $t$th item in SL that belongs to $N^k$;
        if item $j$ fits into the knapsack then
            Place item $j$ into $S$;
        end
    end
end
```

has found a new best MKP solution, then set ℓ to the profit of that solution".

3.5 Computational Results

In this section, we present our computational results. Our algorithm was implemented in `C++` and compiled with Microsoft Visual Studio (v. 15.0) in Windows 10. To solve the master LPs, we used the `CPLEX` (v. 12.7.1) dual simplex solver, under default settings. All experiments were conducted on a 2.80GHz, 4-core Intel i7-7700HQ laptop, with 16GB RAM.

3.5.1 Test instances

The standard benchmark MKP instances, created by Chu & Beasley [37], are available at the `OR-Library` [15]. The weights a_{ij} are random integers between 1 and 1000. Each profit p_j is set to:

$$\sum_{i=1}^{m} a_{ij}/m + 500u_j,$$

where u_j is a random number between 0 and 1. The b_i are set to $\Delta \sum_{j=1}^{n} a_{ij}$, where $0 < \Delta < 1$ is a parameter.

For each combination of $n \in \{100, 250, 500\}$, $m \in \{5, 10, 30\}$ and $\Delta \in \{0.25, 0.5, 0.75\}$, there are ten instances. This makes $3 \times 3 \times 3 \times 10 = 270$ instances in total. For the instances with $n = 100$ and/or $m \leq 10$, the optimal values are known [116]. The remaining 60 instances, with $n \in \{250, 500\}$ and $m = 30$, remain unsolved. The best known lower bounds for those instances can be found in [116].

We remark that the above-mentioned instances have rather large p_j values. Since our separation algorithm for the constraints (3.6) runs in $O\big(n\,|p|\big)$ time, these are unfavourable instances for our approach. Nevertheless, we will see that the running times were acceptable for the instances in question.

3.5.2 Results

Table 3.1 presents the results obtained for the 210 solved MKP instances. Each entry in the table is the average over 10 instances. The columns headed "UB %gap" show the average gap between two upper bounds and the optimum, expressed as a percentage of the optimum. The upper bounds are obtained by LP and SD. The columns headed "LB %gap" show the average gap between three lower bounds and the optimum, again expressed as a percentage of the optimum. These three lower bounds are explained below. Finally, the last column shows the average time taken by our SD algorithm, in seconds. (The time taken to solve the LP, and time taken by our primal heuristic, were negligible, being less than 0.1 seconds in every case.)

We now explain the lower bounds. The column "Gr"" corresponds to the greedy heuristic, in which items are inserted in non-increasing order of profit. The column "LP" corresponds to the case in which items are first inserted in non-increasing order of x_j^*, and then in non-decreasing order of ρ_j. (This is equivalent to running Algorithm 2 with N^1 set to N.) The column "SD" corresponds to our primal heuristic.

A comparison of the 4th and 5th columns shows that the upper bound from the SD is often only slightly stronger than the one obtained with linear programming. The only exception is when $n = 100$ and $m = 5$, where SD closes a significant proportion of the integrality gap. We found this rather surprising, given the fact, mentioned in the introduction, that several authors recommended applying SR to the MKP.

Another observation is that both integrality gaps tend to decrease as n increases, but increase as m increases. The effort required to solve the SD, however, seems to be an increasing function of both n and m.

The news on the lower bounds is much more positive. The LP-based heuristic performs much better than the greedy heuristic, and our primal heuristic performs much better still. As for the computing times, the final column shows that we could solve the SD exactly within a few seconds for most of the instances. (No instance took more than 40 seconds.)

Table 3.2 presents the results obtained for the 60 unsolved MKP instances. For

			UB %gap		LB %gap			Time
n	m	Δ	LP	SD	Gr	LP	SD	(sec)
		.25	0.997	0.881	35.04	1.554	0.672	3.48
100	5	.50	0.452	0.418	24.56	0.883	0.272	4.00
		.75	0.318	0.271	13.43	0.561	0.259	2.47
		.25	0.220	0.204	35.09	0.582	0.416	4.32
250	5	.50	0.108	0.105	24.36	0.383	0.149	4.76
		.75	0.076	0.070	12.27	0.226	0.117	4.30
		.25	0.071	0.070	35.53	0.405	0.268	9.24
500	5	.50	0.038	0.037	22.92	0.166	0.104	8.87
		.75	0.023	0.023	12.26	0.118	0.067	7.66
		.25	1.584	1.565	41.65	2.029	1.103	7.23
100	10	.50	0.796	0.789	28.99	1.218	0.461	6.65
		.75	0.478	0.476	16.22	0.667	0.243	4.95
		.25	0.454	0.452	40.94	0.965	0.720	10.15
250	10	.50	0.231	0.230	27.48	0.528	0.322	14.02
		.75	0.141	0.141	15.13	0.242	0.197	12.95
		.25	0.166	0.166	39.96	0.503	0.427	23.71
500	10	.50	0.078	0.076	25.99	0.260	0.234	22.12
		.75	0.056	0.056	14.20	0.138	0.123	15.80
		.25	2.971	2.971	42.97	3.306	2.680	12.50
100	30	.50	1.336	1.336	32.05	0.873	0.761	11.47
		.75	0.830	0.830	18.49	0.986	0.581	8.04

Table 3.1: Results for the solved MKP instances.

			UB %gap		LB %gap			Time
n	m	Δ	LP	SD	Gr	LP	SD	(sec)
		.25	1.065	1.065	43.20	1.595	1.517	35.75
250	30	.50	0.477	0.477	31.32	0.616	0.600	32.71
		.75	0.278	0.278	16.98	0.417	0.387	26.11
		.25	0.480	0.480	42.29	0.807	0.807	108.64
500	30	.50	0.209	0.209	29.91	0.451	0.451	89.76
		.75	0.134	0.134	16.20	0.180	0.180	106.78

Table 3.2: Results for unsolved benchmark MKP instances.

these instances, the average percentage gaps are computed with respect to the best known lower bound, rather than the optimal solution value. The results here are similar to those obtained for the solved instances, but the similarity between the LP and SD bounds is even more apparent. In fact, closer inspection of the output revealed that, for almost all of these instances, the SD upper bound was equal to the LP upper bound rounded down to the nearest integer. Thus, for large-scale random instances, one may as well just solve the continuous relaxation, without bothering to use SR.

We now move on to look at the distribution of results for individual instances. Figure 3.1 plots the difference between the LP and SD lower bound gaps and the difference between SD and LP upper bound gaps for each of the 210 instance for which the optimum is known. The dotted line shows the improvement in lower bound gaps when the SD is used while the solid line shows the same thing for the upper bound. The differences in upper bounds have been magnified by a factor of 10 to make them more discernible. The first thing to note is each line lies completely on one side of the horizontal axis. In the case of the upper bounds, this is simply because the SD bound is at least as good as the LP bound. In the case of the lower bounds, this means the SR-based ones are always better than the LP-based ones. Secondly, noting that the 90 instances in each group have the same number of constraints (with

Figure 3.1: LP vs Surrogate bounds - 210 instances with known optima

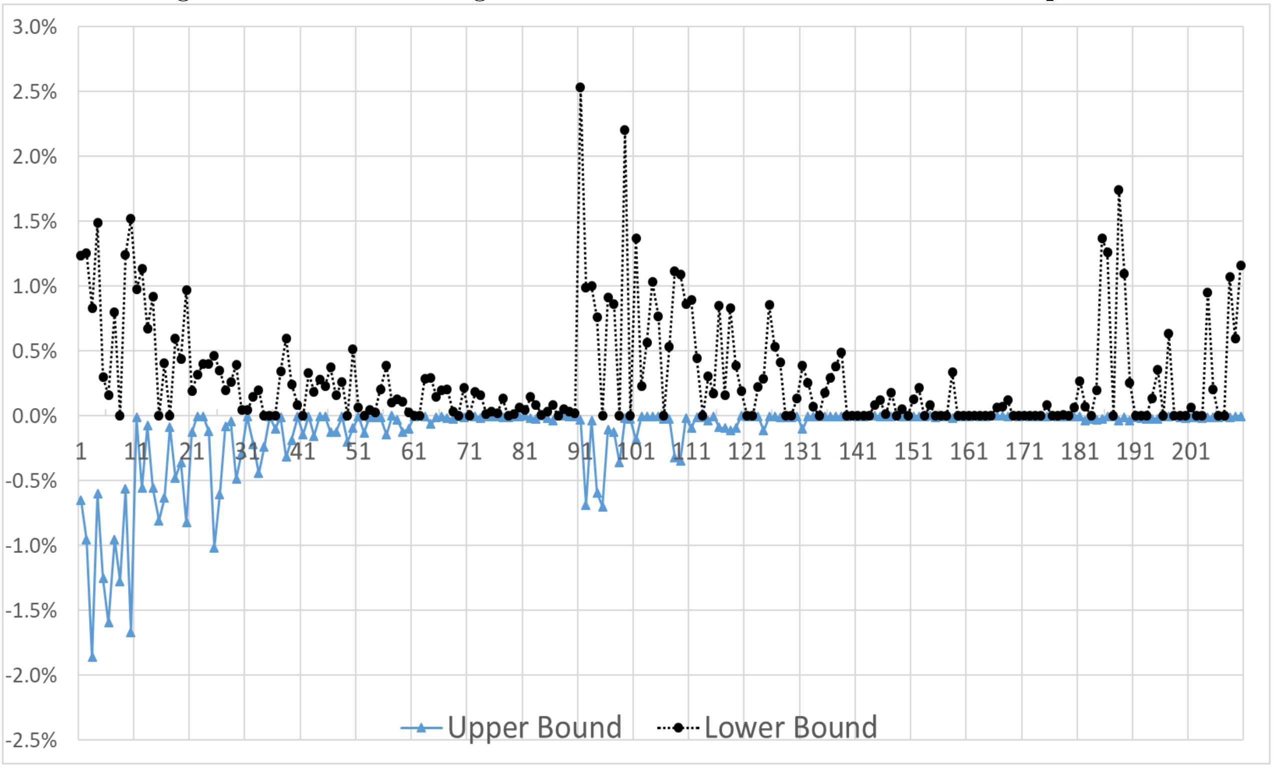
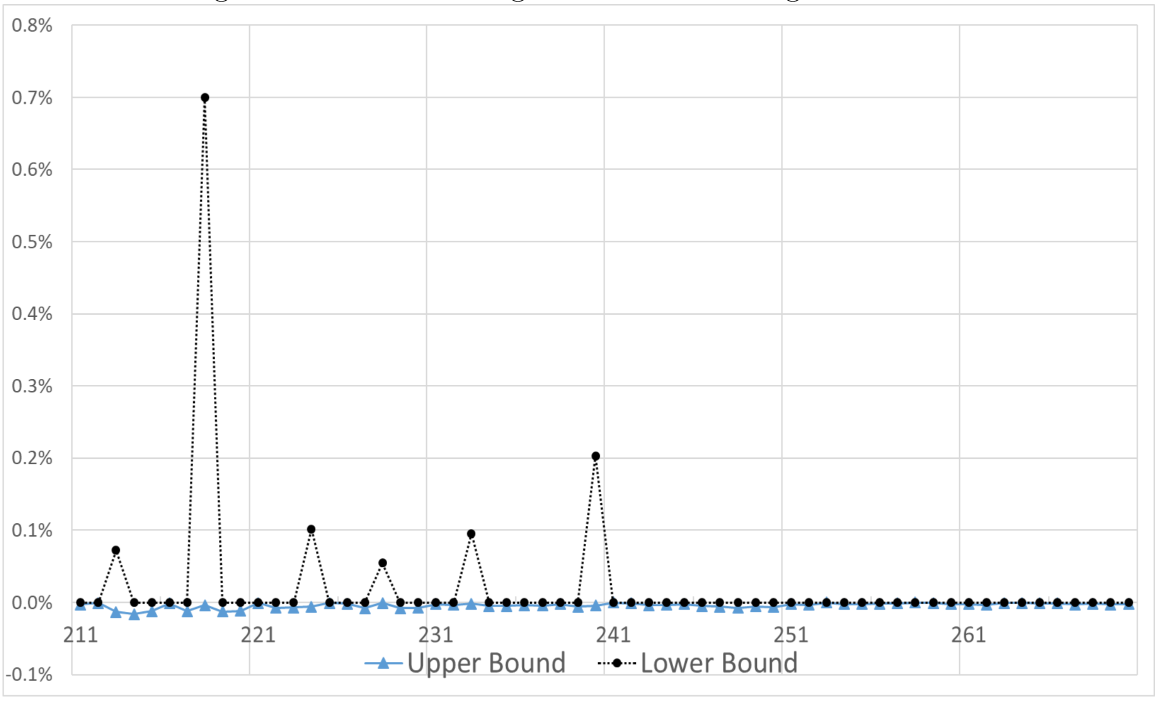


Figure 3.2: LP vs Surrogate bounds - 60 largest instances



100 items in the first 30 instances, 250 in the next 30, and 500 in the last 30), we see a clear pattern that, regardless of the number of constraints, the improvement provided by the surrogate approach to the lower bounds is most pronounced for fewer items. Finally, there appears to be a decreasing trend in each group of 30 instances as well. This suggests that the surrogate approach works best when the capacity of each knapsack is around 25 percent of the weight of all items, and performs less well as the capacity increases to 50 percent and 75 percent, respectively.

Figure 3.2 plots the same for the largest 60 instances, whose optimum values are not known. It reconfirms that for each individual instance the LP and SD upper bounds are identical. Furthermore, the SR-based lower bounds are only sporadically better than the LP-based ones (although, again, never worse).

We remark that some additional implementation "tricks" could potentially improve our algorithm. For example, one could begin by running a local-search heuristic for the MKP, in order to obtain a better initial lower bound ℓ. One could use a heuristic to solve the separation problem (3.6), and only use an exact separation algorithm when the heuristic fails. Finally, one could also use local search to improve the solutions found by the primal heuristic.

3.6 Conclusion

Although surrogate relaxation has been applied many times to the MKP, we are the first to solve the surrogate dual exactly for instances of meaningful size. To do this, we used an approach based on cutting planes and binary search, rather than one of the classical heuristics, which are all essentially variations of the subgradient method.

The results indicate that one can solve the dual quite quickly in practice, using a modern implementation of the simplex method to solve the intermediate linear programs. They also indicate that our primal heuristic produces solutions of good quality. On the other hand, the upper bounds from the surrogate dual were rarely much better than the upper bounds from linear programming relaxation.

An interesting topic for future research would be to improve our bounding procedures, and integrate them within an exact algorithm for the MKP. It would also be interesting to adapt the approach to some other combinatorial optimisation problems.

Chapter 4

Using Surrogate Relaxation as a Matheuristic

4.1 Introduction

In the field of combinatorial optimisation, it is important to distinguish between *exact* algorithms, which are guaranteed to yield a provably optimal solution, and *heuristics*, which are not (see, e.g., [117, 131]). In recent years, there has been increasing interest in heuristics which draw on concepts and algorithms from the literature on exact methods (e.g., [9, 25, 130]). These hybrid solution methods have come to be known as *matheuristics* [114, 115].

There is a long history of matheuristics that are based on *dual ascent* (e.g., [4, 10, 146]) or *Lagrangian relaxation* (LR for short) (e.g., [15, 16, 34]). More recently, matheuristics have been proposed based on *Benders decomposition* and *Dantzig-Wolfe decomposition* (e.g., [9, 25, 115, 130]). The goal of this paper is to show that one can also easily design matheuristics based on a different technique, known as *surrogate relaxation* [75, 79].

Our approach exploits the fact that, in SR, the relaxed problem is typically solved using *dynamic programming* (DP). This enables one to easily extract a large number of "promising partial solutions", each of which can be "repaired" to obtain a feasible solution to the original problem.

To show the generality of our approach, we conduct computational experiments on three specific combinatorial optimisation problems: the *multidimensional knapsack problem* (MKP), the *simple plant location problem* (SPLP) and the *three-dimensional assignment problem* (AP3).

The paper has a simple structure. Section 4.2 is a literature review. Sections 4.3 to 4.5 concern the MKP, SPLP and AP3, respectively. Concluding remarks are made in Section 4.6.

4.2 Literature Review

In this section, we briefly review the relevant literature. We cover surrogate relaxation in Subsection 4.2.1. The other three subsections are concerned with the MKP, SPLP and AP3.

4.2.1 Surrogate relaxation

Consider an integer linear program (ILP) of the form

$$\max\Big\{p^T x : \ Ax \le b, \ Cx \le d, \ x \in \mathbb{Z}^n\Big\}, \tag{4.1}$$

where $p \in \mathbb{Q}^n$, $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $C \in \mathbb{Q}^{r\times n}$ and $d \in \mathbb{Q}^r$. In *surrogate relaxation* (SR for short), we pick a vector $\mu \in \mathbb{Q}^m_+$ of *surrogate multipliers*, and solve the following simpler ILP [75, 79]:

$$\max\Big\{p^T x : \ \big(\mu^T A\big)x \le \mu^T b, \ \ Cx \le d, \ x \in \mathbb{Z}^n\Big\}. \tag{4.2}$$

This gives an upper bound, that we call $U(\mu)$.

The following results are proved in [75, 79]:

- If suitable multipliers are chosen, then the upper bound from SR is at least as strong as the one obtained by relaxing the integrality condition.
- If an optimal solution to (4.2) is feasible for (4.1), then it is also optimal for (4.1).

- $U(\cdot)$ is a piecewise-constant quasi-convex function of μ.

The problem of finding a vector μ that maximises $U(\mu)$ is called the *surrogate dual*. Heuristics for solving the surrogate dual can be found in [50, 98, 134]. Dyer [50] provides two methods, one based on interior-point methods for non-convex non-linear programs and another based on "quasi-subgradients". The latter are a generalisation of the well-known subgradients used in convex non-differentiable minimisation. Dyer was able to show finite termination only for the first method, and optimality was not guaranteed for either method. Karwan & Rardin [98] proposed a different heuristic approach, based on the iterative solution of linear programs. Sarin *et al.* [134] proposed yet another heuristic, a modified subgradient approach which involves the solution of Lagrangian subproblems.

Exact algorithms for the surrogate dual can be found in [22,23,48,100]. Boros [23] showed that, when the relaxed problem is a 0-1 knapsack problem (KP), the dual can be reduced to the solution of a polynomial number of KPs. This result relies however on the ellipsoid method, which is notoriously slow in practice. Kim & Kim [100] extended the previous work of Sarin *et al.* [134], and proved convergence in their stopping rule for solving the Lagrangian subproblem and hence their overall problem. More recently, Boland *et al.* [22] incorporated the use of the so-called 'bundle trust region' method to improve search efficiency in the surrogate multiplier space. Finally, Dokka *et al.* [48] extended the result of Boros to a more general family of problems and relaxations (see the previous chapter).

SR has been applied to several combinatorial optimisation problems (e.g., [60, 62, 65, 99, 112, 121]). Fisher *et al.* [60] applied SR to the *job shop scheduling* problem. They tried relaxing two different constraints sets: (i) the capacity constraints, which resulted in a 0-1 KP that they solved using dynamic programming, and (ii) the precedence constraints, which left them with a set of single-machine problems. Despite the pseudo-polynomial and polynomial time complexity for solving each of these relaxations, respectively, they reported prohibitive running times when the lower bounds from the relaxation were incorporated in the state-of-the-art implicit enumeration scheme of the era.

Lorena & Lopes [112] applied a combination of SR and LP relaxation to the *set covering* problem. They updated their multipliers with a sub-gradient scheme. They also obtained feasible primal solutions, using a simple greedy heuristic, that was initialised using a 'critical' variable identified by the solution of the SR. The idea of using SR and LP relaxation in combination was also used by Galvão *et al.* [65], in the context of the *maximal covering location* problem. They used a subgradient-based heuristic to calculate multipliers, and they construct a primal solution by opening facilities present in the relaxed solution. Narciso & Lorena [121] applied SR to the *generalised assignment* problem. They too used a sub-gradient scheme, but they solved the relaxed problems via specialist algorithms, rather than resorting to LP relaxation as in [65, 112]. They also converted the solutions to the relaxation into primal solutions, using an efficient constructive heuristic.

Finally, the book 'Knapsack Problems' [99] provides a couple of heuristics for the MKP that use SR. Each time the surrogate multipliers are updated, two primal heuristics are called. The first heuristic uses the multipliers to construct a kind of 'artificial' weight for each item. It then applies the classical 'profit-per-weight' greedy heuristic for the 0-1 KP, but with the artificial weights in place of actual weights. The second heuristic is similar, but based on a combination of SR and LP relaxation. We say more about the MKP in the next subsection.

As one can see, although SR has been applied to several combinatorial optimisation problems, only a few papers use SR within a heuristic scheme [60, 65, 112, 121]. Moreover, our approach, which exploits the nature of DP algorithms, is different to the approaches in those papers.

4.2.2 The MKP

The MKP takes the form:

$$\max \quad \textstyle\sum_{j=1}^{n} p_j x_j \tag{4.3}$$

$$\text{s.t.} \quad \textstyle\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad (i = 1, \dots, m) \tag{4.4}$$

$$x_j \in \{0, 1\} \qquad (j = 1, \dots, n), \tag{4.5}$$

where the p_j are positive rationals and the a_{ij} and b_i are positive integers. The MKP is $\mathcal{NP}$-hard in the strong sense [67]. For surveys of solution approaches, including applications of SR to the MKP, see [62] or Chapter 9 of [99].

4.2.3 The SPLP

In the SPLP (also known as the *uncapacitated facility location problem*), we have m facilities and n clients. The cost of opening facility i is f_i. The cost of assigning client j to facility i is c_{ij}. The goal is to decide which facilities to open, and to assign each client to an open facility at minimum cost. The standard ILP formulation of the SPLP is:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=1}^m f_i y_i + \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} && && (4.6)\\
\text{s.t.} \quad & \textstyle\sum_{i=1}^m x_{ij} = 1 && (\forall j) && (4.7)\\
& y_i - x_{ij} \geq 0 && (\forall i,j) && (4.8)\\
& x \in \{0,1\}^{mn}, y \in \{0,1\}^m. && && (4.9)
\end{aligned}$$

The SPLP is $\mathcal{NP}$-hard in the strong sense [106]. For surveys, see, e.g., [106, 141].

4.2.4 The AP3

In the (axial) *three-index assignment problem* (AP3), we have n tasks, n workers and n machines, and the cost incurred if task i is performed by worker j on machine k is c_{ijk}. The problem calls for an assignment of tasks to workers and machines of minimum total cost. The standard 0-1 LP formulation of the AP3 is:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n c_{ijk} x_{ijk} && && (4.10)\\
\text{s.t.} \quad & \textstyle\sum_{j=1}^n \sum_{k=1}^n x_{ijk} = 1 && (i = 1, \ldots, n) && (4.11)\\
& \textstyle\sum_{i=1}^n \sum_{k=1}^n x_{ijk} = 1 && (j = 1, \ldots, n) && (4.12)\\
& \textstyle\sum_{i=1}^n \sum_{j=1}^n x_{ijk} = 1 && (k = 1, \ldots, n) && (4.13)\\
& x_{ijk} \in \{0,1\} && (i,j,k = 1, \ldots, n). && (4.14)
\end{aligned}$$

The AP3 is $\mathcal{NP}$-hard in the strong sense [67]. For surveys, see, e.g., [72] or Chapter 10 of [28].

4.3 The Multidimensional Knapsack Problem

In this section, we present and test an SR-based matheuristic for the MKP. The four subsections in this section cover the relaxation, the primal heuristic, the test instances and the computational results.

From now on, given some $v \in \mathbb{Z}^r$, we let $|v|$ denote $\sum_{i=1}^r |v_i|$. Also, $\lfloor v \rfloor$ denotes the vector obtained by rounding down each component of v to the nearest integer. We also call the constraint $\big(\mu^T A\big)x \leq \mu^T b$ the "surrogate constraint".

4.3.1 Relaxation

Suppose that we relax constraints (4.4) with multiplier vector $\mu \in \mathbb{Q}_+^m$. The relaxed problem is:

$$\begin{aligned} \max \quad & p^T x \\ \text{s.t.} \quad & \big(\mu^T A\big)x \leq \mu^T b \\ & x \in \{0,1\}^n. \end{aligned} \tag{4.15}$$

This is a *0-1 knapsack problem* (0-1 KP). Although the 0-1 KP is $\mathcal{NP}$-hard [96], it can be solved in pseudo-polynomial time using DP [18]. More precisely, if $\mu^T A \in \mathbb{Z}_+^m$ and $\mu^T b \in \mathbb{Z}_+$, then the 0-1 KP can be solved in $O\Big(n\,\big(\mu^T b\big)\Big)$ time.

Unfortunately, if we place no restrictions on μ, there is no guarantee that $\mu^T A$ and $\mu^T b$ will be integral. Moreover, even if we are able to make them integral, there is no guarantee that $\mu^T b$ will be small. As a result, the naive DP approach may not be viable. This led us to devise a heuristic to obtain a surrogate constraint with "small enough" integer coefficients:

1. Let t be a positive integer *target* value for the right-hand side of the surrogate constraint.

2. Solve the continuous relaxation of the ILP (4.3)-(4.5), and let $\pi \in \mathbb{Q}^m_+$ be the optimal dual vector.

3. Compute the "scaling factor" $s = (t + 1 - \epsilon)/\big(\pi^T b\big)$, where ϵ is a small positive quantity.

4. Set μ to $s\pi$. The resulting "tentative" surrogate constraint is:
$$\Big((s\pi)^T A\Big)x \le t + 1 - \epsilon. \tag{4.16}$$

5. Round the coefficients of (4.16) down to the nearest integer. The resulting surrogate constraint is
$$\Big\lfloor (s\pi)^T A \Big\rfloor x \le t.$$

In our experience, setting t to somewhere between $n^{1.5}$ and $n^{2.5}$ gives a good compromise between upper bound strength and running time. We provide details with the computational results.

We now make an important observation. Let T be any positive integer not less than t. For $q = 0, \dots, T$, let $f(q)$ denote the optimal solution to a modified version of the relaxed problem, in which the right-hand side of (4.15) is changed to q. Using DP, one can compute $f(q)$ for $q = 0, \dots, T$ in $O(nT)$ time. Moreover, for any given q, one can extract the corresponding optimal (binary) x vector in $O(n)$ time. We will let $\bar{x}^q$ denote the optimal x vector for a given q.

4.3.2 Primal heuristics

A very simple greedy heuristic for the MKP is as follows. Sort the items in non-increasing order of profit. Starting with an empty knapsack, go through the sorted list. Each time one encounters an item that fits into the knapsack, insert that item and proceed to the next item.

A slightly more sophisticated version of this heuristic is as follows. Solve the continuous relaxation of the ILP (4.3)-(4.5). Let $x^* \in [0,1]^n$ be the optimal solution, and let $\rho \in \mathbb{Q}^n_+$ be the vector of reduced costs. Sort the items in non-increasing order of x^*, and then sort the items with $x^*_j = 0$ in non-decreasing order of ρ_j. Then start with an empty knapsack, and go through the sorted list, as before.

We now modify the second heuristic so that it exploits information obtained during the DP. Let q be an integer that is "close" to t. We say that x_j is *promising* if $\bar{x}^q_j = 1$. We start with an empty knapsack, and then proceed through the sorted list *twice*. The first time, we insert only promising items into the knapsack. The second time, we insert non-promising items (if there is space).

Assuming that the sorted list has been constructed in a pre-processing step, our heuristic takes $O(nm)$ time for a given q. In practice, it is extremely fast. We therefore run it for $O(n)$ values of q. More precisely, we run it for $q = t - n, \ldots, t + n$. This gives us $2n + 1$ primal solutions, of which we pick the best. The entire matheuristic runs in $O\big(n^2 m\big)$ time (not including the time taken to solve the LP). In practice, it is very fast.

Intuitively, the vectors $\bar{x}^q$ can be viewed as "partial" MKP solutions, and scanning the sorted list can be viewed as a way to "repair" those partial solutions. The attractive feature of DP is that it enables us to extract many different partial solutions quickly. The use of many different partial solutions can be viewed as a "diversification" mechanism, which gives the matheuristic many chances to find a good MKP solution.

4.3.3 Test instances

The standard benchmark MKP instances, created by Chu & Beasley [37], are available at the `OR-Lib` [15]. The weights a_{ij} are random integers between 1 and 1000, and each profit p_j is set to:

$$\sum_{i=1}^{m} a_{ij}/m + 500u_j,$$

where u_j is a random number between 0 and 1. The b_i are set to $\Delta \sum_{j=1}^n a_{ij}$, where $0 < \Delta < 1$ is a parameter.

For each combination of $n \in \{100, 250, 500\}$, $m \in \{5, 10, 30\}$ and $\Delta \in \{0.25, 0.5, 0.75\}$, there are ten random instances. This makes 270 instances in total. As mentioned in the previous chapter, the instances with $n \in \{250, 500\}$ and $m = 30$, sixty instances in total, remain unsolved. The optimal solution values for the solved instances, and the best known lower bounds for the others, can be found in [26, 116]. We verified the optimal values for a sample of the solved instances with `CPLEX` (v. 12).

4.3.4 Computational results

We implemented the lower- and upper-bounding procedures in C, and we used the dual simplex solver of the `CPLEX` callable library (v. 12.0) to solve the LP relaxation. We set the "target" t to n^2.

Table 4.1 shows the results obtained for the 210 solved MKP instances. Each row corresponds to a particular combination of m, n and Δ. The columns headed "UB gaps (%)" show the average gaps between two upper bounds and the optimum, expressed as a percentage of the optimum. Each figure is the average over ten random instances. Similarly, the columns headed "LB gaps (%)" show the average percentage gaps between three lower bounds and the optimum. The column labelled 'Gr' corresponds to a simple greedy heuristic, in which items are inserted in non-increasing order of profit. The column labelled "LP" was obtained using the LP-based heuristic mentioned above. finally, the column labelled 'SR' was obtained using our matheuristic.

We see that all upper bounds are pretty tight for these instances, but the SR upper bounds are consistently a little worse than the LP upper bounds. The situation is very different for the lower bounds. Indeed, the LP lower bounds are much better than the greedy ones, and the SR lower bounds are much better still. Note also that the SR-based primal heuristic described in this chapter performs slightly better than the one described in the previous chapter (see Table 3.1 in Section 3.5).

Table 4.2 presents the results obtained for the 60 unsolved MKP instances. For these instances, the average percentage gaps are computed with respect to the best known lower bound, rather than the optimal solution value. The results here are similar to those obtained for the solved instances, with the SR upper bounds slightly

			UB gaps (%)		LB gaps (%)		
m	n	Δ	LP	SR	Gr	LP	SR
		0.25	1.00	1.05	35.04	1.554	0.595
5	100	0.50	0.45	0.65	24.56	0.883	0.245
		0.75	0.32	0.58	13.43	0.561	0.157
		0.25	0.22	0.26	35.09	0.582	0.344
5	250	0.50	0.11	0.19	24.36	0.383	0.103
		0.75	0.08	0.19	12.27	0.226	0.088
		0.25	0.07	0.09	35.53	0.405	0.147
5	500	0.50	0.04	0.08	22.92	0.166	0.046
		0.75	0.02	0.08	12.26	0.118	0.035
		0.25	1.59	1.70	41.65	2.029	1.132
10	100	0.50	0.80	1.00	28.99	1.218	0.451
		0.75	0.48	0.77	16.22	0.667	0.202
		0.25	0.46	0.50	40.94	0.965	0.569
10	250	0.50	0.23	0.32	27.48	0.528	0.176
		0.75	0.14	0.26	15.13	0.242	0.113
		0.25	0.17	0.19	39.96	0.503	0.242
10	500	0.50	0.08	0.12	25.99	0.260	0.110
		0.75	0.06	0.11	14.20	0.138	0.061
		0.25	2.97	3.11	42.97	3.306	2.066
30	100	0.50	1.34	1.57	32.05	0.873	0.763
		0.75	0.83	1.14	18.49	0.986	0.477

Table 4.1: Results for the 210 solved MKP instances

			UB %gap		LB %gap		
m	n	Δ	LP	SR	Gr	LP	SR
		.25	1.065	1.011	43.20	1.595	1.027
30	250	.50	0.477	0.567	31.32	0.616	0.404
		.75	0.278	0.401	16.98	0.417	0.259
		.25	0.480	0.504	42.29	0.807	0.598
30	500	.50	0.209	0.253	29.91	0.451	0.269
		.75	0.134	0.193	16.20	0.180	0.136

Table 4.2: Results for the 60 unsolved MKP instances.

worse than the LP ones. As before, the LP-based primal heuristic performs much better than the greedy one, and the SR-based primal heuristic performs better still.

Note that most of the bounds deteriorate as m increases, but get better as n increases. This is consistent with other works on the MKP [62, 99].

We now move on to look at the distribution of results for individual instances. Figure 4.1 plots the difference between the LP and SR lower bound gaps (dotted line) and the difference between SR and LP upper bound gaps (solid line) for each of the 180 instances that have either 5 or 10 constraints. Note that a positive number for the dotted line means that the SR lower bound is better than the LP one, whereas a negative number for the solid line means that the SR upper bound is worse than the LP one. The upper bound differences are also magnified by a factor of 10 so that the trend is easier to observe. We remark that the instances are sorted by m first, then by n, and finally by Δ, just as in the above tables.

The first thing to note is that each line lies completely on one side of the horizontal axis. In the case of the upper bounds, this means that the LP bound is always better than the SR bound. We remark however that 115 out of 180 instances have an SR upper bound that is within 0.01 percent of the LP bound. In the case of lower bounds, the SR lower bound is always better than the LP one. On the other hand, there are 52 instances out of the 180 for which the improvement is less than 0.1 percent.

Figure 4.1: LP vs SR - 180 instances with m = 5 or 10

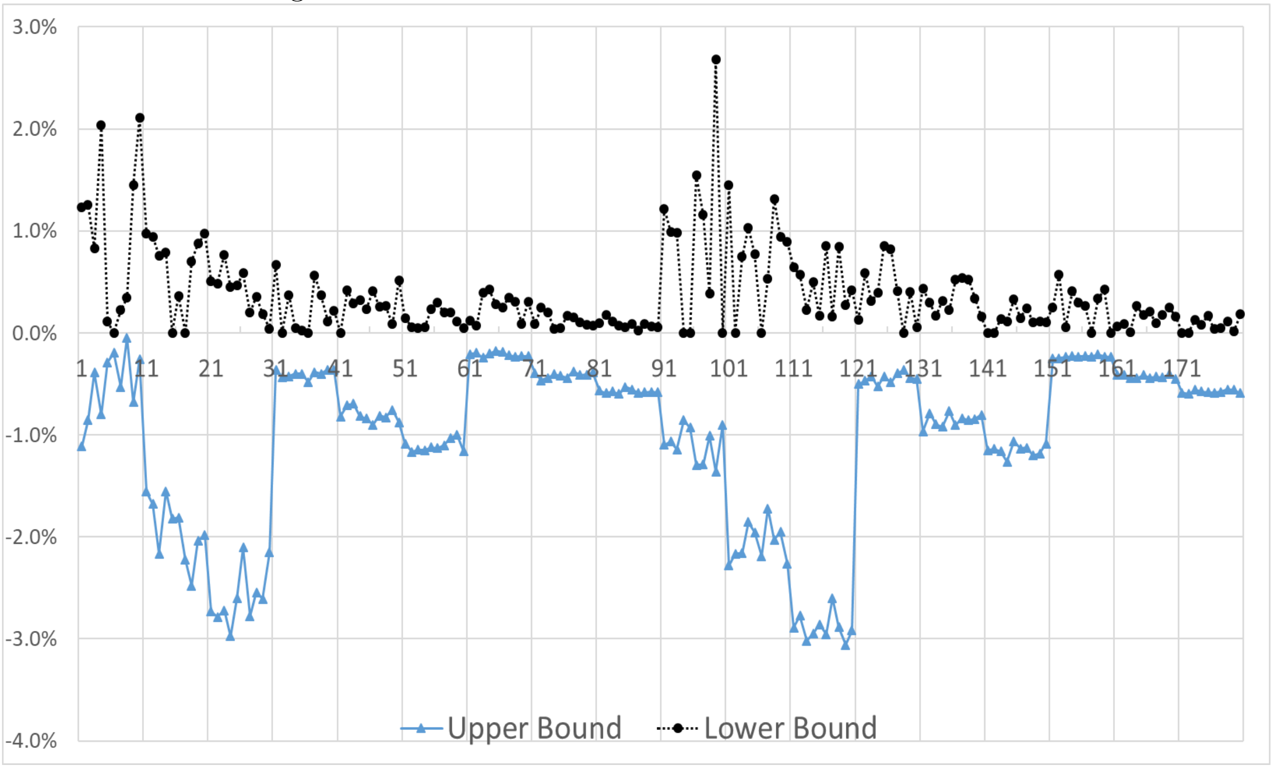


Figure 4.2: LP vs SR - 90 instances with m = 30

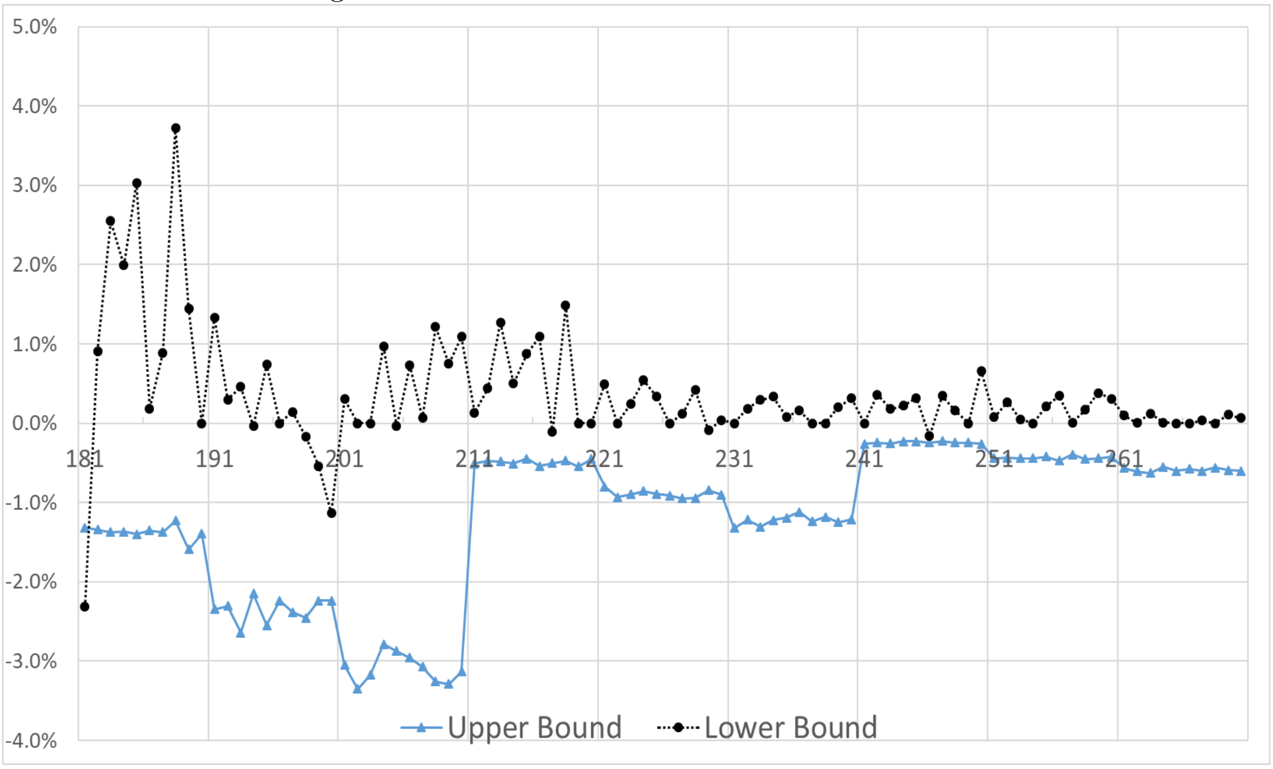
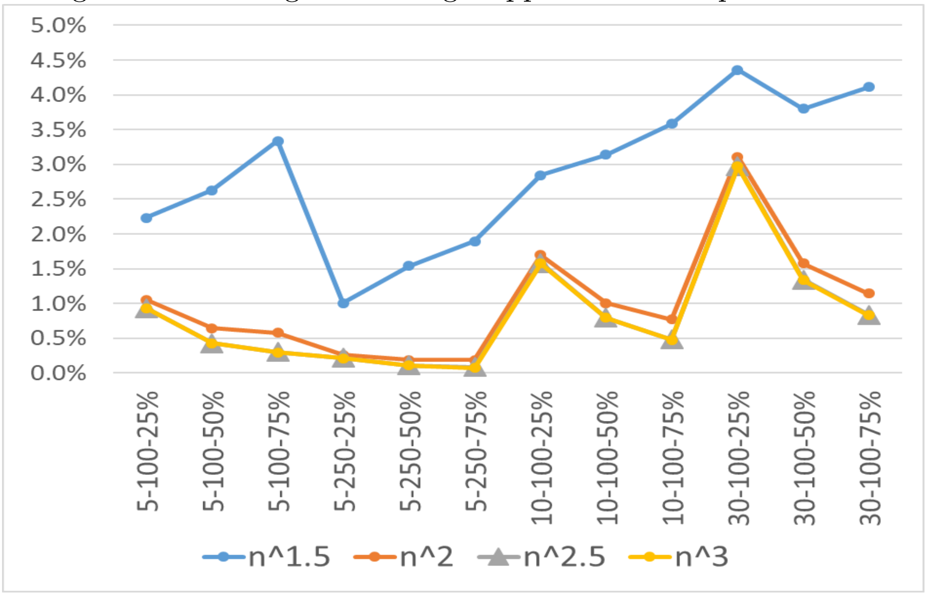


Figure 4.3: Average Percentage Upper Bound Gap as t varies



Looking at the upper bounds in more detail, it is apparent that the performance of the SR bound deteriorates as Δ increases (i.e., as the knapsack capacity gets larger relative to the total weight of the items). As for the lower bounds, it seems that, regardless of the number of constraints, the improvement in bound provided by the surrogate approach is most pronounced for the case of fewer items. Moreover, the SR lower bound seems to be particularly good when Δ is 25 percent. Finally, there seems to be some kind of correlation present, in the sense that, when the SR upper bound is weak, the SR lower bound tends to be weak as well.

Figure 4.2 plots the same for the most difficult 90 instances that have 30 constraints. (We remind the reader that the optimal values are not known for the last 60 of these instances.) It reconfirms the behaviour of upper bounds as discussed above. Also, the SR-based lower bounds are nearly always better than the LP-based ones, although there are a few exceptions. The improvement seems to decrease as the number of items n or knapsack capacity Δ increases.

For the sake of brevity, we do not report detailed running times for the MKP instances. The DP algorithm took a few seconds for the instances with $n = 100$, and a few minutes for those with $n = 500$. The time taken by the matheuristic was negligible in all cases.

Finally, we mention that we also experimented with setting the target t to other values, such as $\lceil n^{1.5} \rceil$ and $\lceil n^{2.5} \rceil$. This had a clear effect on running times but, oddly, did not have a consistent effect on bound quality. Figure 4.3 shows the SR upper bound gap versus the target t for a subset of the instances. The upper bound improves significantly as the the target t is increased from $\lceil n^{1.5} \rceil$ to $\lceil n^2 \rceil$. Thereafter as t increases there is a significant increase in computing time with little appreciable improvement in the upper bound. All things considered, setting t to n^2 seems like a good compromise.

4.4 The Simple Plant Location Problem

In this section, we apply our approach to the SPLP. The structure of this section is the same as the previous one.

4.4.1 Relaxation

When applying SR to the SPLP, one can relax either (4.7) or (4.8). If we relax (4.7), using a multiplier vector $\mu \in \mathbb{Z}_+^n$, the relaxed problem is to minimise

$$\sum_{i=1}^{m} f_i y_i + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{4.17}$$

subject to (4.8), (4.9) and

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \mu_j x_{ij} = |\mu|. \tag{4.18}$$

If, on the other hand, we relax (4.8), using a multiplier vector $\mu \in \mathbb{Z}_+^{mn}$, the relaxed problem is to minimise (4.17) subject to (4.7), (4.9) and

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \mu_{ij} x_{ij} \leq \sum_{i} \left(\sum_{j} \mu_{ij} \right) y_i. \tag{4.19}$$

In the Appendix, we show that both of the relaxed problems can be solved in $O\big(mn\,|\mu|\big)$ time via DP.

In our preliminary computational tests, we found that a good compromise between running time and bound quality was obtained by using the first relaxation and setting $|\mu|$ to around mn. To generate a suitable μ, we used the following heuristic:

1. Solve the continuous relaxation of the ILP (4.6)-(4.9), and let $\pi \in \mathbb{Q}_+^n$ be the optimal dual vector for the constraints (4.7).

2. Compute the scaling factor $s = mn/|\pi|$.

3. Set μ to $s\pi$, and round each component of μ to the nearest integer.

One peculiarity of the SPLP is that the constraints (4.8) make the LP relaxation massively primal degenerate. This slows down the simplex method, and it also means that there are typically many basic optimal dual solutions to the LP. We decided to solve the LP with an interior-point method rather than the simplex method, to speed things up. This also had the side-effect of making the dual solution π "central" (in the interior of the optimal face in the dual) rather than "extreme".

As in the previous section, we will find it helpful to have several SR solutions rather than only one. So, let T be any positive integer not less than $|\mu|$. For $q = 0, \ldots, T$, let $f(q)$ denote the optimal solution to a modified version of the relaxed problem, in which the right-hand side of (4.18) is changed to q. Using DP, one can compute $f(q)$ for $q = 0, \ldots, T$ in $O(nmT)$ time. Moreover, for any given q, one can extract the corresponding (binary) vectors x and y in $O(mn)$ time. Actually, for our purposes, we will need only to extract the y vector, which can be done in $O(m)$ time for a given q. We will let $\bar{y}^q$ denote the y vector for a given q.

4.4.2 Primal heuristics

One of the best performing constructive heuristics for the SPLP is the "drop" heuristic (see [52]). Sort the facilities in non-increasing order of fixed cost f_i. Temporarily open all facilities, and assign each client to its nearest facility. Then proceed through the sorted list, checking whether the cost can be reduced by closing the current facility and reassigning clients to their nearest open facility. The drop heuristic can be implemented to run in $O(mn + m \log m)$ time [111].

One can improve this heuristic as follows. Solve the continuous relaxation of the ILP (4.6)-(4.9). Let $y^* \in [0,1]^m$ be the optimal y vector, and let $\rho \in \mathbb{Q}_+^m$ be the vector of reduced costs for the y variables. Sort the facilities in non-decreasing order of y_i^*, and then sort the facilities with $y_i^* = 0$ in non-increasing order of ρ_i. Then apply the drop heuristic, but using the new sorted list in place of the original one.

As in the case of the MKP, we now modify the second heuristic so that it exploits information obtained during the DP. Let q be an integer that is "close" to mn. We say that y_i is *promising* if $\bar{y}_i^q = 1$. We start with all facilities open, and then proceed through the sorted list twice. The first time, we only consider non-promising facilities for closure. The second time, we consider the promising facilities.

We run this heuristic for $q = 0.900\,|\mu|, 0.901\,|\mu|, \ldots, 1.100\,|\mu|$. This gives us 201 primal solutions, of which we pick the best. The whole matheuristic runs in $O\big(m^2 n\big)$ time (not including the time taken to solve the LP).

4.4.3 Test instances

Many different benchmark SPLP instances have been created, most of which are available at `UflLib` [138]. We selected the following instances:

1. `BK` instances, constructed as in Bilde & Krarup [21]. These instances have random integer costs. They are small by modern standards, but have quite large integrality gaps. There are 10 instances each of type "B" and "C", and 100 each of types "D" and "E". We were able to compute the optimal solution values quite quickly using `CPLEX`.

2. `K-med` instances, constructed as in Anh *et al.* [2]. In these instances, the facility and client locations are points in the unit square. For $m = n = 250$, we created five instances with large facility costs (`Kmed-L`), five with medium facility costs (`Kmed-M`) and five with small facility costs (`Kmed-S`). Again, we computed the optimal values with `CPLEX`.

3. The `M*` instances of Kratica *et al.* [107]. These instances have a negative correlation between facility costs and assignment costs. We selected the fifteen smallest instances (five for each value of $m = n \in \{100, 200, 300\}$). Optimal values are given in [127].

4. The KG instances. These were created by Ghosh [71] using a scheme proposed by Koerkel [104]. They come in two types, symmetric (KG-S) and asymmetric (KG-A). We selected the fifteen smallest instances of each type, which have $m = n = 250$. Optimal values for these are given in [55,127].

5. Instances from Kochetov & Ivanenko [103], which we call KI. They only have $m = n = 100$. The ones of type "A", "B" and "C" are designed to have large integrality gaps. In the ones of type A, each client has 10 cheap connections. In the ones of type B, each facility has 10 cheap connections. The ones of type C satisfy both properties. Finally, the ones of type "U" have costs taken from a uniform distribution. There are 30 instances of each type, making 120 in total. Optimal values are given in [103].

4.4.4 Computational results

As before, we implemented the lower- and upper-bounding procedures in C. To solve the LP relaxation, we used the "barrier" solver of CPLEX 12.0. Table 4.3 shows the results. The table has a similar format to Table 4.1.

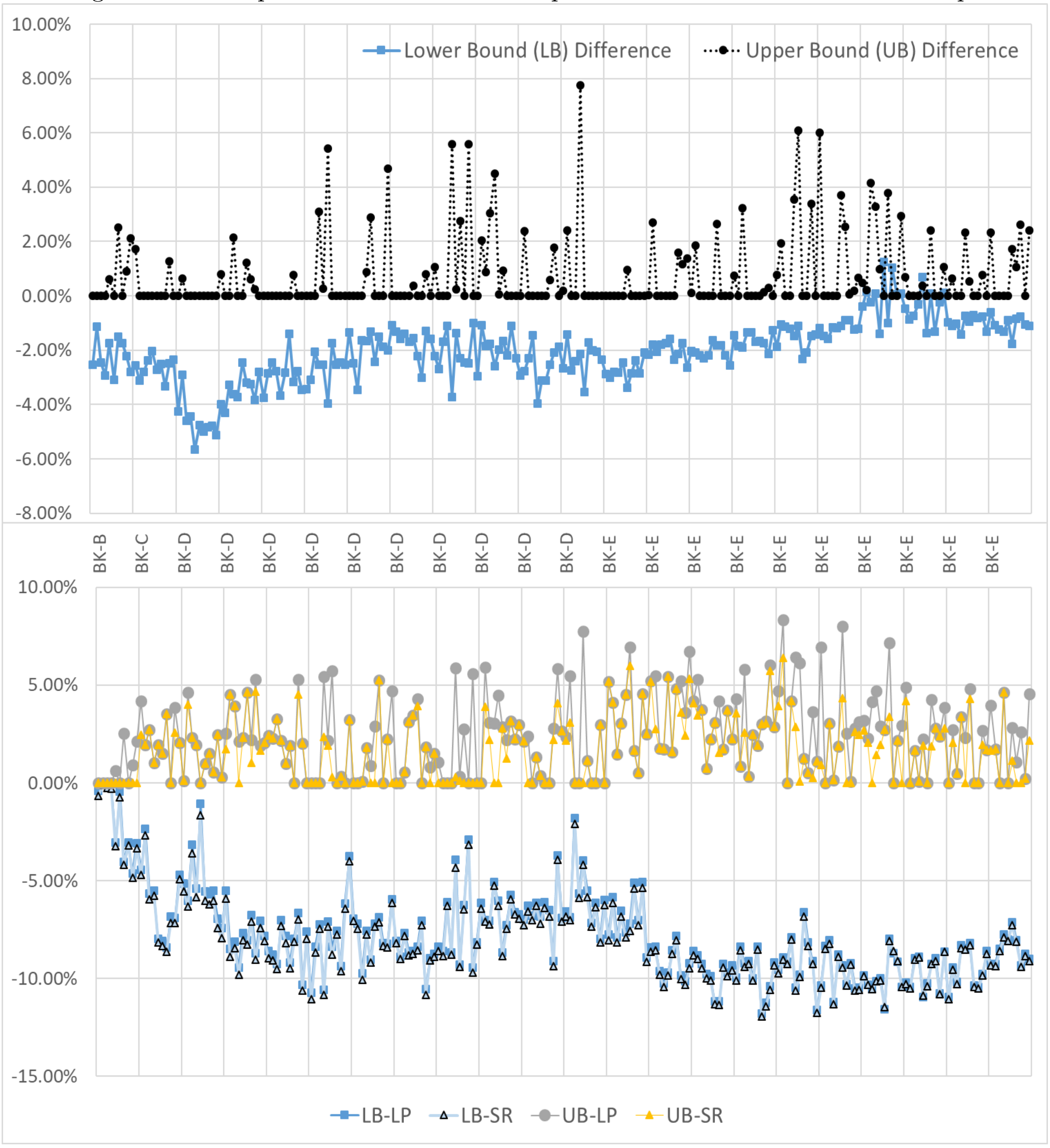
For most of the instances, the SR lower bound is not quite as strong as the LP lower bound. This is consistent with what we saw for the MKP. For the KI-A, KI-B and KI-C instances, however, the SR lower bound is a bit stronger than the LP lower bound.

As for the upper bounds, the LP and SR heuristics do much better than the classical drop heuristic, with the single exception of the MO instances. On the other hand, the superiority of the SR heuristic over the LP-based one is not as clear as it was in the case of the MKP: the SR heuristic does noticeably better on the BK and KI instances, but gives no significant advantage on the other instances.

			LB gaps (%)		UB gaps (%)		
Set	m	n	LP	SR	Drop	LP	SR
BK-B	100	50	1.87	2.09	4.35	0.62	0.00
BK-C	100	50	6.08	6.35	8.02	2.27	1.97
BK-D	80	30	7.16	7.42	14.51	2.00	1.33
BK-E	100	50	9.19	9.33	16.63	2.96	2.16
Kmed-L			0.00	0.01	11.67	0.00	0.00
Kmed-M	250	250	0.00	0.01	7.17	0.00	0.00
Kmed-S			0.00	0.01	0.03	0.00	0.00
MO	100	100	2.85	2.96	0.83	0.86	0.86
MP	200	100	4.20	4.25	1.31	0.21	0.21
MQ	300	100	4.04	4.07	1.07	0.28	0.28
KG-S-a			0.13	0.18	0.48	0.11	0.11
KG-S-b	250	250	0.96	1.00	1.35	0.39	0.39
KG-S-c			3.29	3.30	2.27	0.49	0.33
KG-A-a			0.14	0.19	0.48	0.11	0.11
KG-A-b	250	250	0.98	1.01	1.34	0.28	0.28
KG-A-c			3.13	3.15	2.32	0.67	0.37
KI-A			25.63	22.48	56.83	19.30	14.60
KI-B	100	100	21.10	17.65	41.43	21.66	15.53
KI-C			28.15	28.24	40.92	25.42	18.53
KI-U	100	100	4.64	4.81	28.96	3.71	3.12

Table 4.3: Average percentage gaps for the SPLP

Figure 4.4: Group 1 - LP vs SR Bound Gap Differences & Absolute Bound Gaps



We now analyse the results in more detail. We will find it helpful to divide the sets of instances into three groups. Group 1 contains the 220 `BK` instances; Group 2 contains the 60 instances from the `K-med`, `M*` and `KG` sets; and Group 3 contains the 120 `KI` instances. For each group, we present a figure that has a top half and a bottom half. The top half shows the differences between the upper bounds (dotted line) and lower bounds (solid line). As in the previous section, a positive number for the dotted line means that the SR upper bound improves upon the LP upper bound, and a negative number for the solid line indicates that the LP lower bound is better than the SR lower bound (this is reversed for `KI` instances, since on average the SR lower bound is better in those instances). The lower bound differences are also magnified by a factor of 10 to make the trend discernible (except for KI instances, since the lower bound differences are already big for those). The bottom half of each figure is self-explanatory: it just plots the percentage gaps.

Figure 4.4 shows the detailed results for the `BK` instances. A slight trend is observed in the lower bound differences for the `BK-D` and `BK-E` instances, suggesting that the SR bound, though nearly always worse than the LP bound, improves as the constant facility opening costs get larger. As for upper bounds, the SR bound improves on the LP bound in about 40 percent of the instances. The frequency and magnitude of the improvements does not significantly vary between the various categories of instances. The lower half of the chart, showing the actual percentage gaps, does not show any glaring trends apart from an appreciable deterioration of LP and SR lower bounds after the first 10 instances. These instances have the largest facility opening costs. Finally, looking at the upper bounds, we see that the SR-based ones are optimal for about a third of the `BK` instances.

We now move on to the second group, containing the `K-med`, `M*` and `KG` instances. The `K-med` instances were uninteresting, since nearly all LP and SR bounds were zero. Figure 4.5 shows the detailed results for the remaining `M*` and `KG` instances. Looking at the top half, we see two things. Firstly, the SR lower bounds get closer to the LP lower bound as (a) the size of the instances get bigger (from `MO` to `MQ`), and (b) as the facility opening costs get larger (from `KG-*-A` to `KG-*-C`). This is consistent with

Figure 4.5: Group 3 - LP vs SR Bound Gap Differences & Absolute Bound Gaps

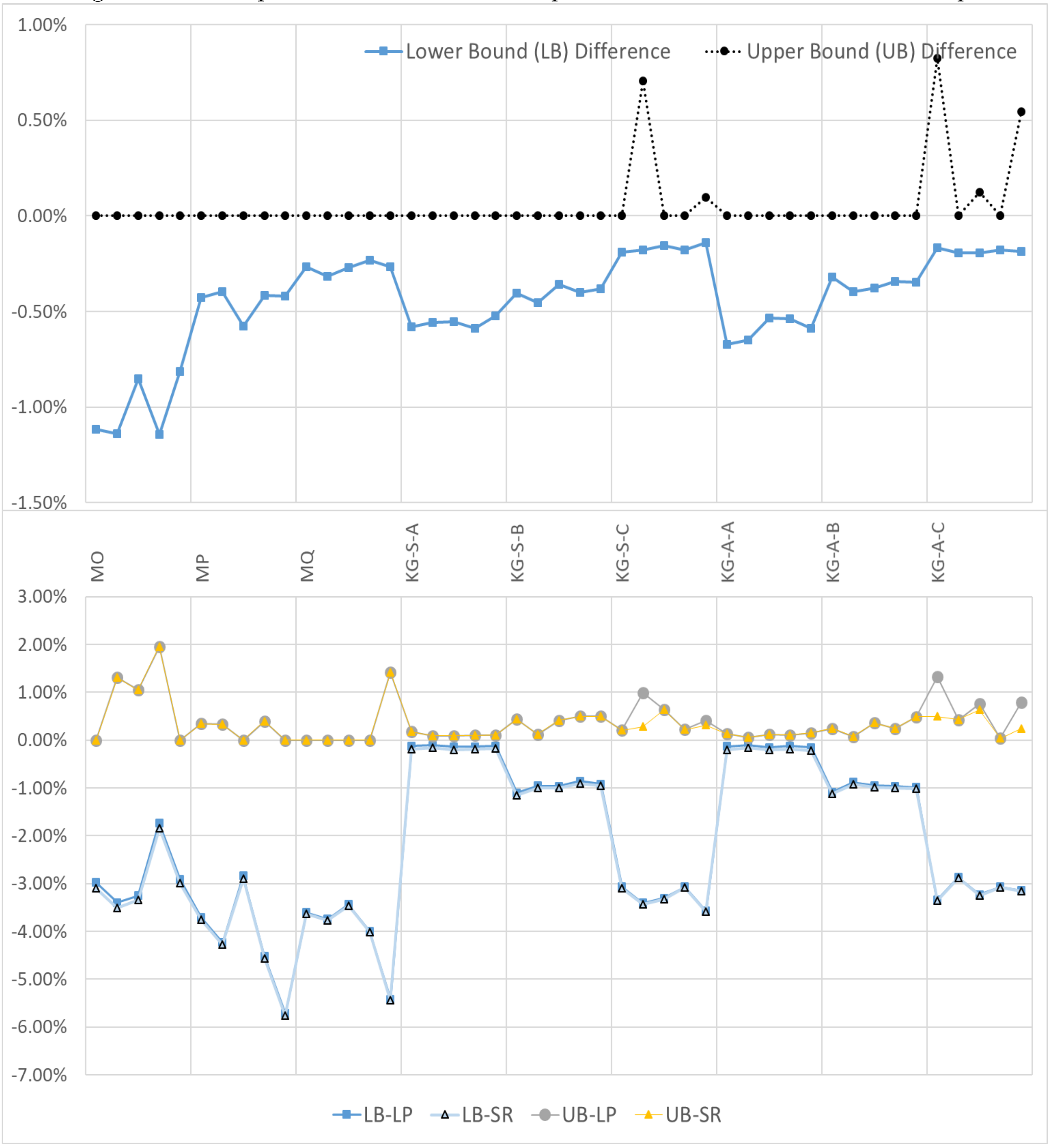
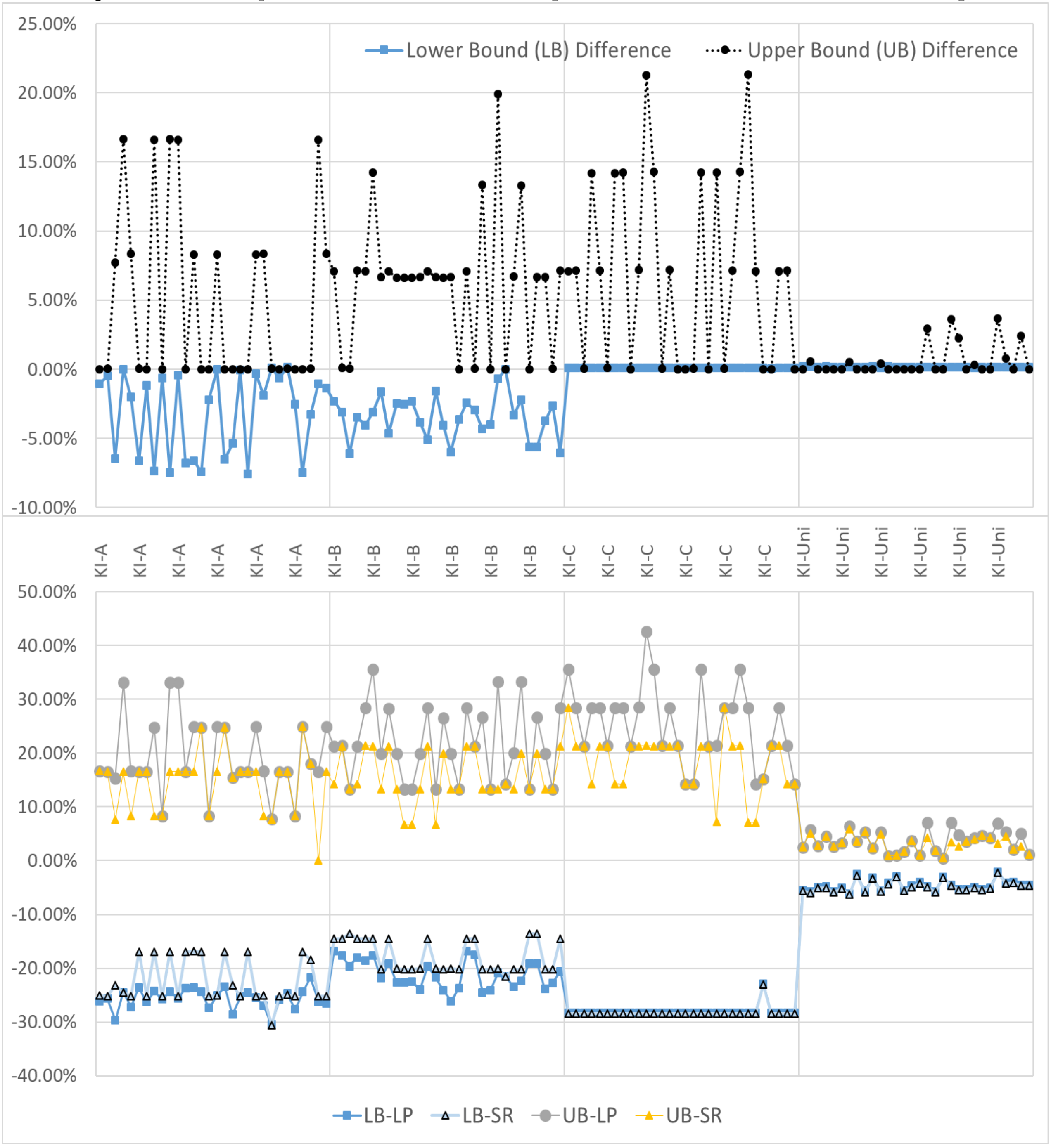


Figure 4.6: Group 3 - LP vs SR Bound Gap Differences & Absolute Bound Gaps



what we saw for the BK instances. Secondly, the SR-based approach rarely improves the upper bounds compared to the LP-based approach.

Coming to the lower half of Figure 4.5, we see a clear pattern in the lower bounds of the KG instances: the LP and SR lower bounds decrease in crisp steps as the facility opening costs get larger. The SR and LP upper bounds are strong, and are only occasionally more than 0.5 percent away from the optimum.

Finally, Figure 4.6 shows the detailed results for the KI instances. Starting with the differences, and reminding the reader that a negative number for the difference in lower bounds means that the SR bound is better, we see some peculiar behaviour. The SR lower bound improves the most compared to the LP lower bound for the KI-A and KI-B instances. The KI-A instances show some extreme differences, while the KI-B instances shows a more standard distribution. The LP and SR lower bounds are nearly identical for the KI-C and KI-U instances. The SR upper bound improvement also displays some "step-wise" behaviour, consistent with the nature of the first three sets of instances. The lower half of the figure suggests a similar step-wise behaviour in the SR lower bound for the KI-A and KI-B instances.

As before, we do not report detailed running times. We just mention that solving the LP and running the primal heuristics took just a few seconds, whereas the DP algorithm took up to a couple of minutes.

4.5 The 3-Dimensional Assignment Problem

Finally, we apply our approach to the AP3. This section has the same structure as the previous two.

4.5.1 Relaxation

In the AP3, there are three "blocks" of linear constraints, i.e., (4.11)-(4.13). We decided to relax two of the three blocks, and leave the other block intact. Suppose, for example, that we relax constraints (4.11) and (4.12) using multiplier vectors $\mu^1, \mu^2 \in$

$\mathbb{Z}^n$. The relaxed problem is to minimise

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk} x_{ijk},$$

subject to (4.13), (4.14) and

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \left(\mu_i^1 + \mu_j^2\right) \sum_{k=1}^{n} x_{ijk} = \left|\mu^1\right| + \left|\mu^2\right|.$$

This is an equality-constrained version of the well-known *multiple-choice knapsack problem* or MCKP (see Chapter 4 of [99]). Provided that $\mu_i^1 + \mu_j^2 \geq 0$ for all i and j, the standard DP algorithm for the MCKP can be applied. It can be shown that, in our context, the DP algorithm runs in $O\Big(n^3\left(\left|\mu^1\right| + \left|\mu^2\right|\right)\Big)$ time (details omitted for brevity).

To generate suitable vectors μ^1 and μ^2, we used the following heuristic:

1. Take the ILP (4.10)-(4.14).
2. Change the sense of the equations (4.11) and (4.12) to "$\geq$" and change their right-hand sides to 0.999.
3. Solve the LP relaxation of the resulting ILP with an interior-point method.
4. Let π^1 and π^2 be the optimal dual vectors for the first two blocks of constraints.
5. Compute the scaling factor $s = 2n^2/(|\pi^1| + |\pi^2|)$.
6. Set μ^1 to $s\pi^1$ and μ^2 to $s\pi^2$.
7. Round each component of μ^1 and μ^2 to the nearest integer.

We now make a few comments about this heuristic. First, the reason for modifying the constraints in step 2 is to ensure that π^1 and π^2 (and therefore also μ^1 and μ^2) are non-negative. Second, we use an interior-point method in step 3 to obtain a "central" dual solution, just as we did for the SPLP. Third, due to the choice of scaling factor in step 5, $|\mu^1| + |\mu^2|$ is approximately $2n^2$. As a result, the DP runs in $O\big(n^5\big)$ time.

Although this is rather slow, we found that it was acceptable for the benchmark AP3 instances described later.

As usual, we store several SR solutions rather than one. This is done using a similar method as in the previous two sections.

4.5.2 Primal heuristics

A very simple greedy heuristic for the AP3 is as follows. Sort the x variables in non-decreasing order of cost. Start with all variables set to zero. Go through the sorted list, setting each variable to one if it is feasible to do so. Note that the sorting step itself takes $O\big(n^3 \log n\big)$ time, which is fairly time-consuming.

As in the previous sections, we consider a modified greedy heuristic in which we use information from the continuous relaxation of the ILP. Specifically, we sort the x variables in non-increasing order of x^* value, and then sort the ones with $x^*_{ijk} = 0$ in non-decreasing order of reduced cost.

Next, as usual, we modify the heuristic so that it exploits the information from the DP. Let q be an integer that is "close" to $|\mu^1| + |\mu^2|$. We say that the variable x_{ijk} is "promising" if $\bar{x}^q_{ijk} = 1$. We then start with all variables set to zero, and proceed through the sorted list twice, first for the promising variables and then for the non-promising ones.

Now let $\alpha = |\mu^1| + |\mu^2|$. We run the above heuristic for $q = 0.9\,\alpha$ to $1.1\,\alpha$ in steps of 0.01α. This gives us 201 primal solutions, of which we pick the best. The whole matheuristic runs in $O\big(n^3\big)$ time (not including the time taken for the LP and DP).

4.5.3 Test instances

For our experiments we consider six sets of test instances, taken from [49]:

1. Instances generated using the method described in Burkard *et al.* [29]. In these instances, the cost coefficients c_{ijk} are of the form $a_i \cdot b_j \cdot c_k$, where each of a_i, b_j and c_k is uniformly distributed in the interval $[1, 10]$. We call these instances `BRW`. There are five instances for each value of $n \in \{25, 54, 66, 80\}$.

2. Instances generated using the method in Crama & Spieksma [43]. For these instances, n points are placed at random in the plane and the c_{ijk} are proportional to the perimeter of the corresponding triangle. There are 9 instances of size 33 and 9 of size 66. We call these instances `CS`.

3. Instances generated using the method in Grundel & Pardalos [81]. These instances are constructed in such a way that the optimal solution value is known *a priori*. We call these instances `GP`. There are five instances for each $n \in \{25, 50, 66, 80\}$.

4. Instances in which the c_{ijk} are chosen with equal probability from one of three intervals $[0, 49]$,$[450, 499]$,$[950, 999]$. Within each interval, the c_{ijk} are randomly picked integers with uniform probability. We call these instances `CLUSTER`. There are five instances for each $n \in \{25, 54, 66, 80\}$. In [49], these instances are attributed to Armin Fügenschuh.

5. Instances generated using the method in Fügenschuh & Höfler [64]. The coefficients c_{ijk} in these instances take the value $\lfloor 10000 \cdot z^2 \rfloor$, where z is uniformly distributed in the interval [0,1]. We call these instances `QUAD`. There are five instances for each $n \in \{25, 54, 66, 80\}$.

6. Instances generated using the method in Balas & Saltzman [6]. In these instance, the c_{ijk} are random integers generated uniformly in the interval $[0, 99]$. We call these instances `UNIFORM`. There are four instances for each $n \in \{25, 54, 66, 80\}$.

4.5.4 Computational results

As before, we implemented the lower- and upper-bounding procedures in C. Since the LP relaxation contains n^3 variables and is primal degenerate, we used the `CPLEX` barrier solver once more.

Table 4.4 shows the results for the `BRW`, `CS` and `GP` instances. The table has a similar format to Tables 4.1 and 4.3, the only difference being that the column labelled "#" shows the number of instances of each type. We see that the LP lower

			LB gaps (%)		UB gaps (%)		
Set	n	#	LP	SR	Gr	LP	SR
	25	5	0.05	2.19	212.76	0.96	0.14
BRW	54	5	0.05	0.93	212.22	0.43	0.12
	66	5	0.05	0.75	233.04	1.26	0.41
	80	5	0.05	0.68	210.27	1.00	0.40
CS	33	9	0.02	0.35	10.80	0.91	0.46
	66	9	0.02	0.33	13.32	0.17	0.09
	25	5	0.72	1.54	8.91	0.27	0.27
GP	50	5	0.43	0.85	5.53	0.23	0.23
	66	5	0.79	1.22	4.59	0.00	0.00
	80	5	0.02	0.35	3.70	0.77	0.75

Table 4.4: Average percentage gaps for some benchmark AP3 instances.

bounds are quite tight on all instances, whereas the SP lower bounds are a little worse. As for the upper bounds, the LP-based heuristic performs much better than the greedy heuristic in all cases, and the SR-based matheuristic performs much better still.

We had to treat the `CLUSTER`, `QUAD` and `UNIFORM` instances a little differently. For most of those instances, the optimal solution has a cost of zero. This makes it impossible to compute percentage integrality gaps. Moreover, the LP and SR lower bounds are almost always zero as well. Thus, in Table 4.5, we show only the average of the “raw” optimal values and upper bounds.

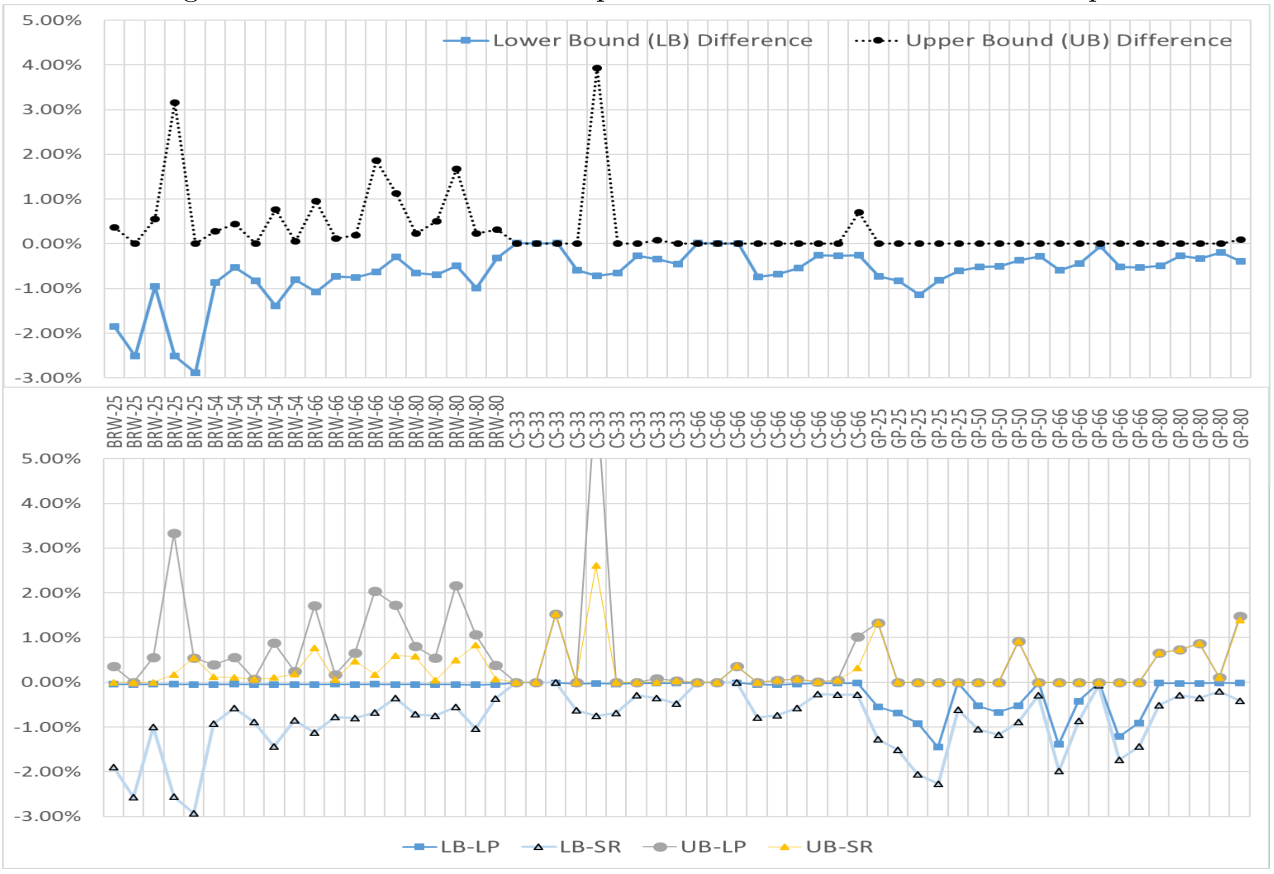
Table 4.5 reveals that, for these instances, the LP-based heuristic performs only a little better than the greedy heuristic. The SR-based matheuristic does much better, giving solutions whose cost is an order of magnitude smaller.

We now report the detailed results by instance, in Figures 4.7 and 4.8. Figure 4.7 is related to the instances from Table 4.4 and is constructed exactly like the Figures 4.4 and 4.5 in the previous section. The lower bound differences show that as the `BRW`

				UBs		
Set	n	#	Opt	Gr	LP	SR
CLUSTER	25	5	5.6	724.8	709.6	29.2
	54	5	0.0	719.0	630.2	33.4
	66	5	0.0	722.6	704.6	29.8
	80	5	0.0	460.8	612.4	28.2
QUAD	25	5	1.0	4481.8	3012.8	136.0
	54	5	0.0	5029.2	6013.0	210.0
	66	5	0.0	5082.0	3985.0	161.6
	80	5	0.0	3092.8	4832.8	160.8
UNIFORM	25	4	1.0	109.5	56.8	24.5
	54	4	0.0	94.5	76.5	16.8
	66	4	0.0	96.0	67.5	16.3
	80	4	0.0	103.8	106.3	18.8

Table 4.5: Average optima and upper bounds for remaining AP3 instances.

Figure 4.7: LP vs SR Bound Gap Differences & Absolute Bound Gaps

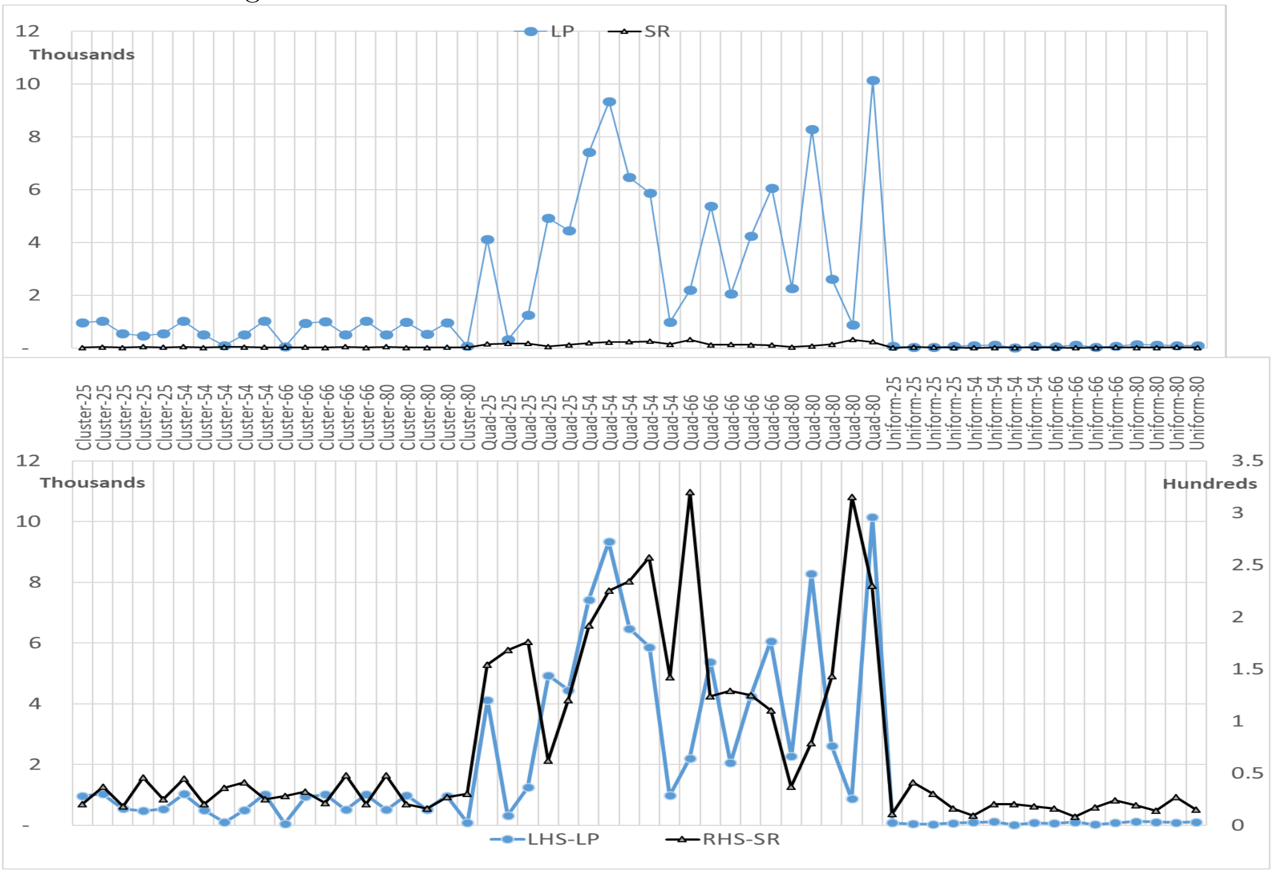


instances get larger the SR lower bound gets closer to the LP bound. Coming to the upper bounds, the SR-based one consistently improves upon the LP-based one in the `BRW` instances, but we do not see any trend related to instance size. In the case of the `CR` instances, we see a couple of outliers biasing the average results reported in Table 4.4. The lower part of the figure shows how the LP and SR based upper bounds come within 0.1 percent of optimal with high frequency (26 out of 38 instances).

For the "problematic" instances that mostly have an optimal value (and lower bounds) of zero, we show in Figure 4.8 the "raw" values of the LP and SR upper bounds. The top part of the figure is reproduced in the lower half, but with the SR upper bounds scaled as shown in the axis on the right-hand-side. This has been done to make any patterns more discernible. Overall, however, we do not see any extra information in the distribution of bounds compared to the average case.

As before, we do not report detailed running times. We just mention that solving

Figure 4.8: LP vs SR Bounds Differences & Absolute Bounds



the LP and running the primal heuristics took just a few seconds, whereas the DP algorithm took up to five minutes for the larger instances.

4.6 Concluding Remarks

We have shown that surrogate relaxation, a well-known technique for finding dual bounds for combinatorial optimisation problems, can also be used within a matheuristic scheme. The results that we obtained for the MKP, SPLP and AP3 show that this approach has promise.

There are several possible avenues for further research. In the first place, it would be worthwhile applying the surrogate approach to other combinatorial optimisation problems. Secondly, one could attempt to improve the primal solutions from our method, for example by using some kind of local search procedure. Lastly, one could experiment with alternative methods for computing the surrogate multipliers. (In principle, one could use one of the methods for the surrogate dual mentioned in Subsection 5.2.1. One would however have to be careful to get a good trade-off between bound quality and running time.)

Acknowledgement: We like to thank two former masters students at Lancaster University, Dongqi Liu and Karn Verochana, for performing some preliminary experiments with surrogate relaxations of the SPLP.

Appendix

In this appendix, we show how to solve two surrogate relaxations of the SPLP in pseudo-polynomial time using DP.

Option 1: relaxing the assignment equations

First, we consider the case in which the equations (4.7) are relaxed. Let $\mu \in \mathbb{Z}^n_+$ be the multiplier vector. The relaxed problem is to minimise (4.17) subject to (4.8), (4.9) and (4.18).

For $k = 1, \dots, m$ and $q = 0, \dots, |\mu|$, let $\phi(k, q)$ denote the optimal solution to the following 0-1 LP:

$$\begin{array}{lll} \min & \sum_{i=1}^k f_i y_i + \sum_{i=1}^k \sum_{j=1}^n c_{ij} x_{ij} & \\ \text{s.t.} & \sum_{i=1}^k \sum_{j=1}^n \mu_j x_{ij} = q & \\ & y_i - x_{ij} \geq 0 & (i = 1, \dots, k;\ j = 1, \dots, n) \\ & x \in \{0,1\}^{kn}, y \in \{0,1\}^k. & \end{array}$$

In other words, $\phi(k, q)$ represents the smallest contribution that the first k facilities can make to the lower bound, under the condition that they service a set of clients that contributes q units to the left-hand side of (4.18). By convention, we set $\phi(0, q) = 0$ for all q. We also set $\phi(k, q) = \infty$ if the above 0-1 LP is infeasible.

For $k = 1, \ldots, m$, $\ell = 1, \ldots, n$ and $q = 0, \ldots, |\mu|$, let $\psi(k, \ell, q)$ denote the optimal solution to the same 0-1 LP as before, but with the added constraints $y_k = 1$ and $x_{kj} = 0$ for $j > \ell$.

We have the following DP recursions for $k = 1, \ldots, m$ and $q = 0, \ldots, |\mu|$:

$$\begin{aligned} \phi(k,q) = &\quad \min\left\{\phi(k-1,q), \psi(k,n,q)\right\} \\ \psi(k,0,q) = &\qquad\quad \phi(k-1,q) + f_k. \end{aligned}$$

We also have the following DP recursion for $k = 1, \ldots, m$, $\ell = 1, \ldots, n$ and $q = 0, \ldots, |\mu|$:

$$\psi(k,\ell,q) = \min\Big\{\psi(k,\ell-1,q),\ \psi(k,\ell-1,q-\mu_\ell) + c_{k\ell}\Big\}.$$

From this it is possible to compute all ϕ and ψ values in $O\big(mn\,|\mu|\big)$ time. Once this is done, the lower bound is $\phi\big(m, |\mu|\big)$.

To save memory, one can use the following trick: re-use the same array to compute the ψ values when you go from one value of k to the next. This means that you need an array of size $m\,|\mu|$ to store the ϕ values and an array of size $n\,|\mu|$ to store the ψ values. Once the algorithm is finished, you only need the ϕ array to find out which facilities should be opened.

Option 2: relaxing the variable upper bounds

Now suppose that we relax the inequalities (4.8) instead, using multiplier vector $\mu \in \mathbb{Z}_+^{mn}$. The relaxed problem is to minimise (4.17) subject to (4.7), (4.9) and (4.19). We complement the y variables, by defining $\bar{y}_i = 1 - y_i$. The objective function (4.17) can then be written as:

$$\sum_i f_i(1 - \bar{y}_i) + \sum_{i,j} c_{ij} x_{ij}.$$

Constraint (4.19) becomes

$$\sum_i \left(\sum_j \mu_{ij}\right) \bar{y}_i + \sum_{i,j} \mu_{ij} x_{ij} \le |\mu|. \tag{4.20}$$

For $k = 1, \dots, n$ and $q = 0, \dots, |\mu|$, let $\phi(k, q)$ denote the optimal solution to the following 0-1 LP:

$$\begin{array}{lll}
\min & \sum_{i=1}^{m} \sum_{j=1}^{k} c_{ij} x_{ij} & \\
& \sum_{i=1}^{m} \sum_{j=1}^{k} \mu_{ij} x_{ij} = q & \\
\text{s.t.} & \sum_{i=1}^{m} x_{ik} = 1 & (j = 1, \dots, k) \\
& x \in \{0,1\}^{mk}, y \in \{0,1\}^{m}. &
\end{array}$$

In other words, $\phi(k, q)$ represents the smallest contribution that the first k clients can make to the lower bound, under the condition that they contribute q units to the left-hand side of (4.20). By convention, we set $\phi(j, q) = \infty$ if the 0-1 LP is infeasible.

The above 0-1 LP is similar to a multiple-choice knapsack problem (see Chapter 4 of [99]). The $\phi(k, q)$ can be easily computed in $O\big(mn\,|\mu|\big)$ time via DP. Once this is done, the relaxed problem reduces to minimising

$$\sum_i f_i(1 - \bar{y}_i) \; + \; \phi(n, q),$$

subject to the constraint

$$\sum_i \left(\sum_j \mu_{ij} \right) \bar{y}_i + q \; \leq \; |\mu|$$

and the conditions $\bar{y} \in \{0,1\}^m$ and $q \in \big\{0, \dots, |\mu|\big\}$. This last problem is similar to a separable knapsack problem, and can be easily solved in $O\big(m\,|\mu|\big)$ time via DP.

Chapter 5

Anomalous Behaviour of Dual-Based Heuristics

5.1 Introduction

Many important combinatorial optimisation problems arising in practice are $\mathcal{NP}$-hard. Although highly effective exact solution methods have been developed for $\mathcal{NP}$-hard problems (see, e.g., [38, 94]), large-scale instances can still pose a challenge for exact methods. In such cases, one must resort to *heuristics* (see, e.g., [30]).

This paper is concerned with what we call *dual-based* heuristics. These heuristics start by constructing a feasible solution to some kind of dual problem (such as a linear programming dual or a Lagrangian dual). Once a dual solution is obtained, they then attempt to use information from the dual solution (such as reduced costs) to guide a primal heuristic. In fact, they usually generate a *sequence* of dual solutions, which then yields a sequence of primal solutions. At the end, one can just pick the best primal solution found during the course of the procedure.

Dual-based heuristics can be regarded as primitive examples of what are now called *matheuristics*, by which is meant heuristics that draw on concepts from the traditional mathematical programming literature (see, e.g., [114]). In comparison with other matheuristics, however, dual-based heuristics have two very nice features. The first is that their running times and memory requirements tend to be bounded by a low-order polynomial in the instance size. As a result, they can easily be applied to large-scale instances. The second is that they yield both lower and upper bounds on the optimal objective value. If these bounds are close, one has the reassurance that the primal solution is of high quality.

Dual-based heuristics have proven to be highly successful on many well-known problems. This includes problems on networks (such as the *simple plant location* [16, 21,51], *Steiner tree* [126,146], *uncapacitated network design* [4] and *fixed-charge transportation* [31] problems), together with classical combinatorial optimisation problems (such as the *set covering* [7,14,32], *set partitioning* [24,59], *generalised assignment* [58] and *multi-dimensional knapsack* [3] problems).

A natural question, however, is whether there really is any useful information to be extracted from "good" dual solutions, or whether a random dual solution would do just as well. A partial answer to this question is that some dual-based heuristics have been shown to be *approximation algorithms* (see, e.g., [145]). Roughly speaking, this means that there is a bound on their worst-case error. Another partial answer is given in [147], in the context of the set covering problem (SCP). They show empirically that, if a dual solution is optimal, then the variables with zero reduced cost have a high probability of belonging to an optimal solution.

In this paper, we continue to study this question from an empirical (and statistical) viewpoint. We use the SCP and the simple plant location problem (SPLP) as test cases. For each problem, we consider several ways to construct dual solutions, and conduct extensive computational experiments. It turns out that dual-based heuristics can exhibit highly counter-intuitive behaviour. In particular, in the case of the SPLP, solving the dual exactly invariably leads to much worse primal solutions than solving the dual with a simple greedy heuristic. We provide a tentative explanation of this phenomenon, based on the presence or absence of *primal degeneracy*.

The paper has a very simple structure. In Section 5.2, we review the relevant literature. Sections 5.3 and 5.4 are devoted to the SPLP and SCP, respectively. Concluding remarks are made in Section 5.5. Throughout, we assume that the reader is familiar with the basic concepts of linear and integer programming (see, e.g., [38] for a fine treatment).

5.2 Literature Review

We now briefly review the relevant literature. We recall the basics of dual-based heuristics in Subsection 5.2.1. In Subsections 5.2.2 and 5.2.3, we cover applications of the heuristics to the SPLP and SCP, respectively.

5.2.1 Dual-based heuristics

Suppose we have formulated a combinatorial optimisation problem as an *integer linear program* (ILP) of the form

$$\min \left\{ c^T x : \ Ax \geq b, \ x \in \mathbb{Z}^n_+ \right\},$$

where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. The dual of the LP relaxation of this ILP is

$$\max \left\{ b^T \pi : \ A^T \pi \leq c, \ \pi \in \mathbb{R}^m_+ \right\}.$$

Any feasible solution π to this dual provides a lower bound for the original problem. Thus, if we wish to obtain a lower bound quickly, we can solve the dual approximately using some kind of heuristic.

In *dual ascent*, one starts with all π variables set to small values, and then iteratively increases the value of a π variable, until no more can be increased without violating dual feasibility [21, 51]. If desired, one can attempt to improve the dual solution further by applying local search. This is called *dual adjustment* in [51].

Now, for a given primal variable x_j and a given dual solution π, consider the following quantity:

$$\bar{c}_j(\pi) \,=\, c_j - \sum_{i=1}^{m} \pi_j A_{ij}.$$

We call this the *approximate reduced cost.* Intuitively speaking, if π is a near-optimal dual solution, one might expect variables with small approximate reduced cost to have a high probability of belonging to an optimal solution of the original ILP. This leads to the idea of an integrated scheme in which the approximate reduced costs are used to guide a primal heuristic [21, 51]. This is what we mean by a *dual-based* heuristic.

For some problems, *Lagrangian relaxation* (LR) can provide an attractive alternative to dual ascent. Consider an ILP of the form:

$$\min\left\{c^T x : \; Ax \geq b, \; Cx \geq d, \; x \in \mathbb{Z}^n_+\right\},$$

where $A \in \mathbb{Q}^{m\times n}$ and $C \in \mathbb{Q}^{q\times n}$. Relaxing the constraints $Cx \geq d$, with a vector $\lambda \in \mathbb{R}^q_+$ of *Lagrangian multipliers*, we obtain

$$\min\left\{c^T x + \lambda^T (d - Cx) : \; Ax \geq b, \; x \in \mathbb{Z}^n_+\right\},$$

The solution of this relaxed problem yields a lower bound. (For a theoretical treatment of LR for integer programming, see Geoffrion [70]. For a more informal treatment, together with examples of applications, see Fisher [56,57]. For a more thorough and more recent treatment, see Guignard [84].)

It is possible to use LR to drive dual-based heuristics. Indeed, for a given primal variable x_j and a given λ, the quantity

$$\bar{c}_j(\lambda) \; = \; c_j - \sum_{i=1}^{q} \lambda_j C_{ij}$$

can also be viewed as an approximate reduced cost (see, e.g., [32]).

5.2.2 Application to the SPLP

In the SPLP (a.k.a. the *Uncapacitated Facility Location Problem*), there is a set I of facilities and a set J of clients. For any $i \in I$, it costs f_i to open facility i. For any $i \in I$ and $j \in J$, it costs c_{ij} to serve client j from facility i. One must decide which facilities to open, and assign each client to an open facility, at minimum cost.

Balinski [8] formulated the SPLP as the following 0-1 LP:

$$\begin{aligned}
\min \quad & \textstyle\sum_{i\in I} f_i y_i + \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} & & \qquad (5.1)\\
\text{s.t.} \quad & \textstyle\sum_{i\in I} x_{ij} = 1 & (\forall j \in J) & \qquad (5.2)\\
& y_i - x_{ij} \geq 0 & (\forall i \in I, j \in J) & \qquad (5.3)\\
& x_{ij} \in \{0,1\} & (\forall i \in I, j \in J) & \qquad (5.4)\\
& y_i \in \{0,1\} & (\forall i \in I). & \qquad (5.5)
\end{aligned}$$

Here, x_{ij} indicates whether client j is assigned to facility i, and y_i indicates whether facility i is opened.

The LP relaxation of this formulation often gives very tight lower bounds [2,120]. On the other hand, the constraints (5.3) cause massive primal degeneracy, which makes the LP relaxation hard to solve (see, e.g., [137]).

A dual ascent approach was proposed in [21,51]. They start by showing that the dual of the LP relaxation can be written as:

$$\begin{aligned}
\max \quad & \textstyle\sum_{j\in J} v_j & & \\
\text{s.t.} \quad & \textstyle\sum_{j\in J} \max\{0, v_j - c_{ij}\} \leq f_i & (\forall i \in I) & \qquad (5.6)\\
& v_j \geq 0 & (\forall j \in J), &
\end{aligned}$$

where $v \in \mathbb{R}^{|J|}$ is the vector of dual variables for constraints (5.2). They then proceed as follows. First, set each v_j to the cost of assigning client j to its closest facility (i.e., the smallest c_{ij} value over all $i \in I$). Then, examine each client $j \in J$ in turn, and increase v_j to the next smallest c_{ij} value, or, if this renders the dual solution infeasible, by as much as possible while maintaining feasibility. Repeat until no more increases are possible.

One can check that, for a given v, the approximate reduced costs of the facilities are just the slacks of the constraints (5.6). Moreover, for any pair i, j, the quantity $\max\{0, v_j - c_{ij}\}$ can be viewed as an estimate of the dual price of the constraint (5.3). This led Erlenkotter to propose a primal heuristic, in which one iteratively opens facilities whose approximate reduced cost is zero until, for each client $j \in J$, there is an open facility such that $v_j \geq c_{ij}$.

Erlenkotter [51] pointed out that one can sometimes obtain improved dual solutions by applying local search. His *dual adjustment procedure* seeks a decrease of δ in some v_j such that two other v_js can increase by δ. Körkel [104] and Janáček & Bunza [93] presented faster implementations of dual adjustment. More sophisticated dual search procedures were developed in [86,127]. All of these procedures yield a sequence of dual solutions, which in turn enables one to obtain several primal solutions rather than just one.

An analogous Lagrangian approach, in which constraints (5.2) are relaxed, was studied in [16,66]. Galvão and Raggi [66] found good results by running Erlenkotter's procedure first, and then using the subgradient method to improve the dual solution further. Beasley [16] devised a purely Lagrangian framework, which uses the subgradient method to determine the multipliers, and calls a primal heuristic periodically.

Some related approaches can be found in [19,83,111]. Guignard [83] solved the Lagrangian dual using 'Lagrangian dual ascent' and proposed a method to strengthen the relaxation by incorporating some cutting planes that she called *Benders inequalities.* Beltran-Royo *et al.* [19] proposed to split each equation (5.2) into two inequalities, and relax only one of them in Lagrangian fashion. This method is called *semi-Lagrangian* relaxation. It gives stronger bounds, but the relaxed problem is harder to solve. Finally, Letchford & Miller [111] showed how to speed up dual ascent by computing a 'base level' for the dual multipliers. They also devised an improved primal heuristic. It is based on the classical 'drop' heuristic [52], but drops facilities in decreasing order of approximate reduced cost.

5.2.3 Application to the SCP

In the SCP, we are given positive integers m and n, a family of sets $S_1, \ldots, S_n \subset \{1, \ldots, m\}$, and a cost c_j for $j = 1, \ldots, n$. The task is to find a minimum-cost collection of sets such that each member of $\{1, \ldots, m\}$ is contained in at least one set in the collection. The SCP can be formulated as a 0-1 LP of the form:

$$\min \left\{ c^T x : \ Ax \geq e_m, \ x \in \{0,1\}^n \right\}, \tag{5.7}$$

where x_j indicates whether the jth set has been selected, the columns of $A \in \{0,1\}^{mn}$ encode the sets, and e_m is the all-ones vector of order m.

Since the upper bounds of one on the x variables are redundant, the dual of the LP relaxation can be written in the simple form:

$$\max \left\{ e_m^T \pi : \ A^T \pi \leq c, \ \pi \in \mathbb{R}_+^m \right\}.$$

Balas and Ho [5] were the first to apply dual ascent to the SCP. They begin with π set to the all-zeroes vector. They then sort the rows of A in non-decreasing order of density (i.e., number of '1's), and then go through the sorted list, pushing each π variable up to its maximum possible value. To construct primal solutions, they set to one all x variables for which $\bar{c}_j(\pi) = 0$, and then drop variables, if necessary, to make the cover minimal. Beasley [13] followed a similar scheme.

Fisher & Kedia [59] improved the scheme in two ways. First, they added a local search phase to improve the dual solution. Second, they used an improved primal heuristic, which proceeds as follows. Start with an empty cover. For $j = 1, \ldots, n$, let $\kappa(j) \subset \{1, \ldots, m\}$ be the set of *currently uncovered* rows that have a 1 in column j, and compute $\tilde{c}_j = c_j - \sum_{i \in \kappa(j)} \pi_i$. (Note that $\tilde{c}_j$ lies between the original cost c_j and the approximate reduced cost $\bar{c}_j(\pi)$.) Set to one the x variable that minimises $\tilde{c}_j / |\kappa(j)|$. Recompute the $\tilde{c}_j$, and repeat until a cover is obtained. At the end, the cover is again made minimal if necessary.

Lagrangian relaxation has been applied to the SCP by many authors (e.g., [5,7,13, 14,17,32,34,85]). Balas & Ho [5] incorporated it into their branch-and-bound scheme, by occasionally using it to improve the lower bounds from dual ascent. Beasley [13] improved the framework in [5], by proposing several effective 'problem reduction' procedures. These reduce the size of the problem, which makes the dual easier to solve. Beasley [14] proposed a Lagrangian heuristic that used reduced-cost fixing and computes a feasible solution after every subgradient iteration. Beasley & Jörnsten [17] enhanced the algorithm in [13], by replacing the heuristic to obtain a 'prime cover' from [5] with the one from [14] (they also added some cutting planes). Balas & Carrera [7] introduced *dynamic subgradient optimisation*, that re-states the Lagrangian dual whenever a significant proportion of variables have been fixed. Finally, Ceria *et al.* [34] devised a heuristic for large-scale instances, which selects a subset of the columns based on the Lagrangian reduced costs $\bar{c}_j(\lambda)$.

In all of the papers mentioned in the last paragraph, all of the linear constraints are relaxed, and the subgradient method is used to get good multipliers. To construct primal solutions, most of those authors simply set x variables to one in non-decreasing order of $\bar{c}_j(\lambda)$ until a cover is obtained, and then make the cover minimal, if necessary, by setting a few variables back to zero. To date, the best-performing Lagrangian scheme is that of Caprara *et al.* [32], who used a clever technique to improve convergence in the subgradient method, sophisticated rules for fixing primal variables, and a primal heuristic analogous to that of [59] (mentioned above).

An initial study of the value of dual information for the SCP has been made in [147]. They give evidence that, when the dual solution is *optimal*, variables with zero reduced cost have a high probability of belonging to an optimal solution.

5.3 The Simple Plant Location Problem

This section is concerned with the SPLP. In Subsections 5.3.1 and 5.3.2, we present deterministic and randomised procedures, respectively, for generating dual and primal solutions. In Subsections 5.3.3 and 5.3.4, we present and analyse computational results obtained with the deterministic and randomised procedures, respectively.

5.3.1 Deterministic procedures

Dual ascent is fast and simple, but the dual solution obtained is typically sub-optimal. We found it useful to compute an optimal dual solution as well. To do this, we simply solve the LP relaxation of (5.1)–(5.5). (This is easy using modern LP solvers, even for quite large instances.)

We also experimented with using Lagrangian relaxation instead of dual ascent, as in [16]. The idea is to relax the constraints (5.2), and then use the subgradient method to generate a near-optimal vector of Lagrangian multipliers, from which approximate reduced costs can be computed. Unfortunately, we did not find the results illuminating, because the multipliers and primal solutions encountered depended heavily on the starting point, the step size, the stopping rule, and so on.

To generate a primal solution from a given dual solution, we use the *dual-based drop* heuristic that was presented in [111]. This heuristic, which is a modified version of the classical *drop* heuristic of Feldman *et al.* [52], is described in Algorithm 3. In our experience, the heuristic tends to perform slightly better than Erlenkotter's primal heuristic. Moreover, it is fast. (Specifically, it can be implemented to run in $O(mn \log m)$ time, where m is the number of facilities and n the number of clients.)

In order to provide a benchmark, we also consider a primal heuristic that does *not* exploit dual information. This heuristic is identical to Algorithm 3, except that we sort the facilities in non-increasing order of cost f_i rather than $\bar{f}_i$. We call this latter heuristic the *cost-based* drop heuristic.

5.3.2 Randomised procedures

The algorithms presented in the previous subsection yield only a small number of dual and primal solutions. Since we are interested in exploring potential correlations between the quality of dual and primal solutions, we really want many solutions of each type. This led us to devise randomised versions of the algorithms, which produce many solutions, rather than just one. After trying several alternative ways to generate dual solutions, we settled on the following two methods.

1. *Randomised Dual Ascent.* As already noted by Erlenkotter [51], the precise dual vector obtained via dual ascent depends on the order in which the clients are considered. So we simply run dual ascent a large number of times, with the clients sorted in a random order in each major iteration.

2. *Perturbed LP.* We solve the LP relaxation of (5.1)–(5.5), but add a small random

Algorithm 3: Dual-Based Drop Heuristic for SPLP

```
input : positive integers $m$, $n$, cost vectors $f \in \mathbb{Z}_+^m$, $c \in \mathbb{Z}_+^{mn}$,
        dual solution $v \in \mathbb{Z}_+^n$.
Temporarily open all facilities;
Temporarily assign each client to the nearest facility;
Let $U$ be the cost of the initial solution;
for all $i \in I$ do
    Let $\bar{f}_i = f_i - \sum_{j \in J} \max\{0, v_j - c_{ij}\}$;
end
Sort the facilities in non-increasing order of $\bar{f}_i$ (breaking ties at random);
for $i$ = 1 to $m$ do
    Let $k$ be the $i$th facility in the sorted list;
    Let $\Delta$ be the increase in total cost that would be incurred by closing
      facility $k$ and re-assigning each of its clients to the next nearest open
      facility;
    if $\Delta < 0$ then
        Close facility $k$ and re-assign its clients;
        Set $U := U + \Delta$;
    end
end
output: SPLP solution and associated upper bound $U$.
```

number, uniformly distributed in $[-0.25, 0.25]$, to the right-hand side of each constraint (5.2). The effect of the perturbation is to make some of the v_j more "attractive" than others in the dual. For the instances that we tested, 0.25 was the smallest value that enabled us to generate a reasonable number of distinct dual solutions. (We will see in Subsection 5.3.4 that those dual solutions were all optimal or near-optimal.)

In order to provide a benchmark, we also designed a randomised version of the cost-based drop heuristic, that we described in the previous subsection. Instead of selecting the next facility in the sorted list as the next candidate to be dropped, we select a facility at random from the next five facilities in the list. (Five was the smallest value that led to a diversity of primal solutions.)

5.3.3 Results with deterministic procedures

An extensive collection of benchmark SPLP instances is available here:

`http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/`

After some experimentation, we selected four families of instances:

- `K-median` instances, constructed as in Anh *et al.* [2]. In these instances, the facility and client locations are points in the unit square. For $m = n = 250$, we created five instances with large facility costs (`Kmed-L`), five with medium facility costs (`Kmed-M`) and five with small facility costs (`Kmed-S`). These instances have small integrality gaps, and we were able to find the optimal solutions easily using `CPLEX`.

- The `M*` instances of Kratica *et al.* [107]. In these instances, there is a negative correlation between the facility costs and assignment costs. They are hard to solve and have fairly large gaps. There are 22 instances: five each with $m = n \in \{100, 200, 300, 500\}$, one with $m = n = 1000$, and one with $m = n = 2000$. Optimal solution values can be found in [127].

- The KG instances. These were created by Ghosh [71] using a similar scheme to that of Koerkel [104]. They too are hard to solve and have fairly large gaps. They come in two types, symmetric and asymmetric. We denote these by KG-S and KG-A, respectively. We selected the 30 smallest ones, for which optimal solutions are known (see, e.g., [55, 127]). They have $m = n = 250$.

- The instances from Kochetov & Ivanenko [103]. They only have $m = n = 100$, but they are designed to have very large integrality gaps (over 25% in all cases). There are 90 instances in total, and optimal solutions are now known for all of them. We selected the 30 hardest ones, called Gap-C instances.

We implemented the lower- and upper-bounding procedures in C, and we used the CPLEX callable library (v. 12) to solve all LP relaxations. Out of interest, we computed dual solutions using both simplex and barrier methods. (In the case of the barrier method, we switched off "crossover", to ensure that we found a dual solution that lies near the centre of the optimal face in the dual.)

For each instance and each procedure, we computed the gap between the resulting bound and the optimum, expressed as a percentage of the optimum. Table 5.1 displays, for each set of instances and each procedure, the average gap over the instances in the given set.

As expected, the K-median instances are the easiest, and the Gap-c instances the most challenging. Also as expected, dual ascent consistently yields worse lower bounds than LP relaxation.

As for the four options for producing upper bounds, the best upper bounds are obtained when the dual-based heuristic is applied to an optimal dual solution obtained via the barrier method, (This fact was confirmed with a binomial test, which gave $p < 10^{-10}$.) In our view, this provides strong evidence for the claim that dual information can be useful when driving a primal heuristic. Moreover, the upper bounds from the heuristic based on dual ascent are better than those from the cost-based heuristic on all instances apart from the M* and Gap-C instances. (This was confirmed with sign tests.)

Table 5.1: SPLP: Average percentage gaps with deterministic procedures.

		lower bounds		upper bounds			
Set	$m = n$	ascent	LP	ascent	simplex	barrier	cost-based
Kmed-L		0.54	0.00	2.09	1.97	0.36	11.8
Kmed-M	250	0.30	0.00	0.72	0.82	0.00	7.17
Kmed-S		0.00	0.00	0.02	0.04	0.00	0.03
MO	100	6.06	2.85	2.07	21.4	0.86	0.83
MP	200	7.27	4.20	2.64	20.7	0.21	1.31
MQ	300	7.21	4.04	1.47	23.7	0.28	1.07
MR	500	10.2	6.52	3.41	27.6	0.63	1.29
MS	1000	10.1	6.69	6.18	8.75	0.26	0.32
MT	2000	12.7	9.18	0.76	7.69	0.37	1.30
KG-S-A		0.31	0.13	0.14	0.94	0.11	0.47
KG-S-B	250	1.48	0.96	0.64	3.42	0.37	1.35
KG-S-C		4.31	3.29	0.78	7.34	0.49	2.27
KG-A-A		0.30	0.14	0.16	0.86	0.11	0.48
KG-A-B	250	1.50	0.98	0.56	3.34	0.31	1.33
KG-A-C		4.11	3.13	0.79	8.08	0.67	2.32
Gap-C	100	68.8	28.2	43.1	41.0	32.0	40.9

The big surprise, however, is that the dual-based heuristic performs very poorly when the simplex method is used to compute the dual solution. This is so despite the fact that the dual solution is guaranteed to be optimal. A likely explanation for this unusual behaviour is the following. As mentioned in Subsection 5.2.2, the primal LP is massively degenerate. This means that there are typically a huge number of alternative optimal dual solutions to the LP. Of those, the simplex method will select just one (more or less arbitrary) basic solution, which will be an extreme point of the optimal face in the dual. It seems that the "central" dual solution provided by barrier leads to much more reliable reduced costs than the "extreme" one found by simplex.

To test this hypothesis, we inspected the LP solutions in detail. It turns out that, for most of the instances, over 90% of the basic variables took the value zero in the LP solutions obtained by simplex. Moreover, around 90% of the facility reduced costs were zero, with the remaining 10% bein g very large. When barrier was used instead, both of these phenomena disappeared.

5.3.4 Results with randomised procedures

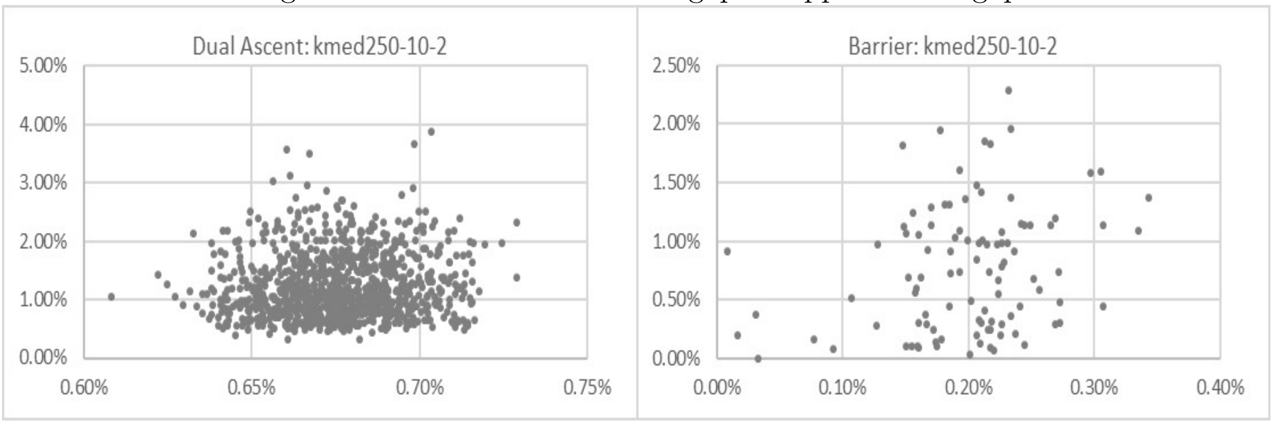
Now we turn our attention to the randomised procedures. Table 5.2 has a similar interpretation to Table 5.1, except that each figure is averaged, not only over the instances in the given set, but also over 100 runs of the randomised procedures. (The missing entries for the instances `MS` and `MT` are due to memory problems in the LP solver.)

By comparing Table 5.2 with Table 5.1, we see that randomisation has little effect on the lower bound obtained with dual ascent. (This is to be expected, since there is nothing special about the order of the clients in the input file.) We also see that perturbing the primal LP causes the lower bound to deteriorate only a little on average, which confirms our belief that the perturbed LP approach yields near-optimal dual vectors. As for the upper bounds, randomisation has no noticeable effect in most cases. When barrier is used, however, the perturbation tends to cause a small worsening. (This too can be confirmed with a sign test.)

Table 5.2: SPLP: Average percentage gaps with randomisation.

		lower bounds		upper bounds			
Set	$m = n$	ascent	LP	ascent	simplex	barrier	cost-based
Kmed-L		0.55	0.12	1.19	6.73	0.52	11.0
Kmed-M	250	0.29	0.02	0.67	5.97	0.17	7.25
Kmed-S		0.00	0.00	0.02	0.04	0.00	0.03
MO	100	6.07	3.24	2.47	19.9	0.95	3.56
MP	200	7.29	4.65	2.03	22.0	0.60	2.52
MQ	300	7.21	4.46	1.91	23.2	0.49	2.20
MR	500	10.2	7.08	3.17	30.3	0.92	2.23
MS	1000	10.1	7.08	3.14	—	0.26	1.48
MT	2000	12.7	9.55	2.75	—	—	1.60
KG-S-A		0.31	0.15	0.13	0.95	0.11	0.48
KG-S-B	250	1.47	1.03	0.69	3.38	0.45	1.37
KG-S-C		4.31	3.49	0.94	6.65	0.57	2.17
KG-A-A		0.30	0.16	0.16	0.91	0.12	0.48
KG-A-B	250	1.50	1.05	0.60	3.37	0.37	1.35
KG-A-C		4.10	3.33	1.04	7.02	0.62	2.27
Gap-c	100	68.8	29.9	42.6	39.9	29.9	41.1

Figure 5.1: SPLP: lower bound gap vs upper bound gap



Since randomisation leads to multiple primal and dual solutions, a natural strategy is to run the procedures a fixed number of times, and then pick the best lower and upper bounds obtained. The results for this approach can be found in Table 5.3. We see that this leads to vastly improved upper bounds in almost all cases. Nevertheless, the relative ordering remains much the same, with barrier performing best and simplex performing poorly in most cases.

Finally, for each instance and each of three methods (randomised dual ascent, perturbed simplex and perturbed barrier), we produced scatterplots to see whether good lower bounds tended to lead to good upper bounds. Figure 5.1 shows two such scatterplots for one of the `Kmed-S` instances. The horizontal and vertical axes represent the percentage gap for the lower and upper bound, respectively. The scatterplots indicate that the correlation, if any, is very weak. The same behaviour was found for other instances and methods. In general, we found that the correlation coefficient varied between -0.1 and 0.2, with no discernable pattern.

All things considered, we believe that randomised dual ascent is a very promising approach for large-scale SPLP instances. It is extremely easy to implement and, in our experiments, it was about 100 times faster than the perturbed LP approach. It also has the advantage that one does not need to use LP software.

Table 5.3: SPLP: Average percentage gaps when randomisation applied and best of 100 bounds taken.

		lower bounds		upper bounds			
Set	$m = n$	ascent	LP	ascent	simplex	barrier	cost-based
Kmed-L		0.49	0.00	0.17	1.71	0.00	7.82
Kmed-M	250	0.13	0.00	0.14	0.82	0.00	6.49
Kmed-S		0.00	0.00	0.00	0.00	0.00	0.03
MO	100	5.93	2.85	0.43	8.29	0.25	0.10
MP	200	7.23	4.20	0.35	10.8	0.12	0.11
MQ	300	7.18	4.04	0.02	12.8	0.23	0.11
MR	500	10.2	6.52	0.29	19.0	0.49	0.47
MS	1000	10.1	6.69	0.05	—	0.00	0.05
MT	2000	12.7	9.18	0.55	—	—	0.75
KG-S-A		0.29	0.13	0.04	0.66	0.03	0.43
KG-S-B	250	1.45	0.96	0.25	2.01	0.19	0.98
KG-S-C		4.29	3.29	0.10	2.92	0.11	0.75
KG-A-A		0.28	0.14	0.06	0.60	0.05	0.39
KG-A-B	250	1.48	0.98	0.18	2.13	0.10	0.97
KG-A-C		4.09	3.13	0.19	3.04	0.18	0.87
Gap-c	100	56.4	28.2	17.1	21.2	12.9	28.6

5.4 The Set Covering Problem

Now we move on to the SCP. This section is structured in exactly the same way as the previous one.

5.4.1 Deterministic procedures

To obtain our initial dual solutions for the SCP, we use the same three approaches that we proposed for the SPLP in Subsection 5.3.1: dual ascent, LP via simplex and LP via interior-point. The only difference is that, for dual ascent, we use the procedure of Balas & Ho [5] described in Subsection 5.2.3. Our implementation of this procedure runs in $O\big(m(n + \log m)\big)$ time. In practice, it is very fast.

To generate a primal solution from a given dual solution, we use the heuristic of Fisher & Kedia [59], described in Subsection 5.2.3. Our implementation runs in $O(mn)$ time and is extremely fast in practice.

Again, in order to provide a benchmark, we also considered a primal heuristic that does *not* use dual information. After some experimentation, we settled on Algorithm 4, which is similar to a heuristic in Balas & Ho [5]. With some care, it can be implemented to run in $O(mn)$ time.

5.4.2 Randomised procedures

To obtain a large number of good quality dual solutions for the SCP, we use the same two approaches that we used for the SPLP, i.e., *randomised dual ascent* and *perturbed LP*. In each iteration of randomised dual ascent, instead of picking the next row in the sorted list, we pick one row at random from the next five in the list. As for the perturbed LP approach, we simply solve the LP relaxation of (5.7), but add a small random number to each component of e_m. As in the case of the SPLP, we let these random numbers be uniformly distributed in $[-0.25, 0.25]$.

Finally, we devised a randomised version of Algorithm 4. In each major iteration, instead of selecting the column that minimises $c_j/|s(j)|$, we select a column at random

Algorithm 4: Greedy Heuristic for Set Covering

```
input  : positive integers $m$, $n$, cost vector $c \in \mathbb{Z}^n_+$.
Set $M := \{1, \ldots, m\}$, $N := \{1, \ldots, n\}$, $C := \emptyset$ and $U := 0$;
repeat
    for all $j \in N$ do
        Let $s(j)$ be the set of rows in $M$ covered by column $j$;
        if $s(j) = \emptyset$ then
            Set $N := N \setminus \{j\}$;
        end
    end
    Let $k \in N$ be the column that minimises $c_j/|s(j)|$;
    Set $M := M \setminus s(k)$, $N := N \setminus \{k\}$, $C := C \cup \{k\}$ and $U := U + c_k$;
until $M = \emptyset$;
for all $j \in C$ do
    if $C \setminus \{j\}$ is a cover then
        set $C := C \setminus \{j\}$ and $U := U - c_j$;
    end
end
output: Minimal cover $C$ and associated upper bound $U$.
```

from the five columns with smallest values of $c_j/|s(j)|$. The running time remains $O(mn)$.

5.4.3 Results with deterministic procedures

As in the case of the SPLP, we implemented all of our lower- and upper-bounding procedures in C. We ran our code on the standard collection of benchmark SCP instances, which are available in the OR-Library [15]. For these instances, the number of variables n is in $\{3000, 4000, 5000, 10000\}$, and the number of constraints m is set to $n/10$. Another parameter is the *density*, which is the expected proportion of non-zero entries in the matrix A. This parameter takes values in $\{2\%, 5\%, 10\%, 20\%\}$. There are 8 sets of instances, labelled 'A' to 'H', with different parameter settings. In each set, there are 5 instances, making 40 instances in total. Data on these instances can be found in the first three columns of Table 5.4. Optimal values are known for the sets A to F [33]. For sets G and H, we used the best-known upper bounds, again taken from [33].

The last six columns in Table 5.4 have a similar meaning to the corresponding ones in Table 5.1. The only difference is that, for sets G and H, the average percentage gaps are with respect to the best-known upper bounds, rather than the optimum.

The results are very different than the ones for the SPLP. In the first place, the lower bounds from dual ascent are of very poor quality, and even the ones from LP relaxation are not great. This is in line with the well-known result that the cost of an optimal SCP solution can be as much as $\ln m$ times larger than the LP bound (see, e.g., [133] and the references therein). As for the upper bounds, there is no evidence that any of the three dual-based heuristics performs better than the cost-based heuristic. In fact, if anything, they look slightly worse. However, the differences are not statistically significant (at the 0.05 level).

Table 5.4: SCP: average percentage gaps with deterministic procedures

			lower bounds		upper bounds			
Set	n	density	ascent	LP	ascent	simplex	barrier	cost-based
A	3000	2%	25.1	1.51	6.42	6.52	10.5	6.05
B	3000	5%	39.1	7.69	6.93	9.47	10.0	5.07
C	4000	2%	27.8	2.32	6.35	7.27	7.65	5.21
D	4000	5%	41.0	8.27	5.54	9.96	9.19	9.66
E	5000	10%	62.5	24.7	11.9	14.1	10.6	9.99
F	5000	20%	77.1	36.4	15.8	14.5	17.4	15.8
G	10000	2%	47.3	10.1	8.89	8.78	9.51	7.53
H	10000	5%	62.5	23.5	11.9	11.8	11.5	11.9
		Mean	47.8	14.3	9.22	10.3	10.8	8.91

5.4.4 Results with randomised procedures

Table 5.5 presents the gaps obtained with the randomised procedures. By comparing Table 5.5 with Table 5.4, we see that randomisation has little effect on the lower bounds. For the upper bounds, however, randomisation causes the cost-based heuristic to perform much worse. Oddly, it also seems to cause the ascent-based heuristic to perform better. Indeed, there is now a clear ordering, with dual ascent coming first and cost-based coming last. This was confirmed by sign tests (Table 5.6).

Table 5.7 shows the results obtained when randomisation is applied 100 times and the best bounds are retained. As in the case of the SPLP, this leads to vastly improved upper bounds in almost all cases. Nevertheless, the relative ordering remains much the same, with dual ascent performing best and cost-based coming last in most cases. This confirms our belief that dual information, if used intelligently, can be useful for guiding primal heuristics. It also makes us more convinced that randomised dual ascent is a promising technique for producing upper bounds quickly.

Observe that, in the case of the SCP, the choice between simplex and barrier has little effect on the quality of the resulting upper bounds. This is very different from

Table 5.5: SCP: average percentage gaps with randomisation

			lower bounds		upper bounds			
Set	n	density	ascent	LP	ascent	simplex	barrier	cost-based
A	3000	2%	23.6	2.49	5.76	6.72	7.04	10.4
B	3000	5%	39.6	9.26	7.22	8.29	8.49	13.7
C	4000	2%	28.4	3.44	6.49	6.76	6.89	11.7
D	4000	5%	43.3	9.93	7.45	8.96	9.30	14.1
E	5000	10%	63.8	26.9	9.72	11.4	11.3	15.9
F	5000	20%	75.1	38.8	13.0	15.3	15.5	19.3
G	10000	2%	44.5	11.9	8.10	10.2	10.2	10.7
H	10000	5%	62.5	25.5	12.4	13.8	13.6	16.2
		Mean	47.6	16.0	8.74	10.2	10.3	13.9

Table 5.6: SCP: Results of sign tests on the quality of the upper bounds.

Hypothesis:	ascent beats simplex	simplex beats barrier	barrier beats cost
Result:	$p < 10^{-6}$	$p < 0.01$	$p < 10^{-12}$

Table 5.7: SCP: average percentage gaps when randomisation applied and best of 100 bounds taken.

			lower bounds		upper bounds			
Set	n	density	ascent	LP	ascent	simplex	barrier	cost-based
A	3000	2%	17.5	1.51	2.07	2.86	2.77	4.16
B	3000	5%	30.5	7.69	0.79	0.79	1.59	3.72
C	4000	2%	21.4	2.32	3.00	2.90	3.08	4.52
D	4000	5%	34.2	8.27	2.13	3.44	2.86	4.35
E	5000	10%	52.1	24.7	1.41	2.07	2.07	3.55
F	5000	20%	65.8	36.4	4.40	4.40	4.40	7.16
G	10000	2%	37.8	10.1	4.09	5.77	5.77	6.50
H	10000	5%	55.2	23.5	7.42	7.39	6.93	10.2
		Mean	39.3	14.3	3.16	3.70	3.68	5.52

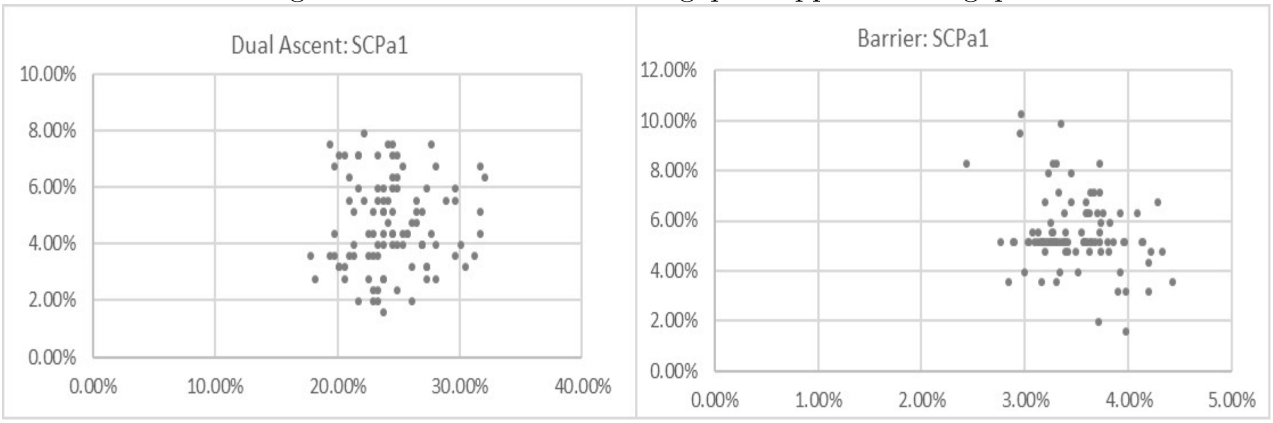
what we saw for the SPLP. The reason is probably that primal degeneracy is less of an issue for the SCP. To confirm this, we check the LP solutions in detail. In the solutions obtained by the simplex method, only around 10% of the basic variables were at zero, compared with around 90% in the case of the SPLP.

Finally, Figure 5.2 shows two typical scatterplots obtained for the SCP. As in the case of the SPLP, there is no obvious positive correlation between the quality of the lower and upper bounds. In fact, the scatterplot on the right suggests a small negative correlation. In general, we found that the correlation coefficient varied between -0.1 and 0.1, with no discernable pattern.

5.5 Conclusion

At the time of writing, dual-based heuristics have been around for about fifty years. Although more sophisticated heuristics (and indeed meta-heuristics) exist, dual-based heuristics are easy to code, scale well with problem size, and tend to give solutions of acceptable quality in practice. We have shown, however, that they can behave in

Figure 5.2: SCP: lower bound gap vs upper bound gap



counter-intuitive ways. In particular, (a) if the primal LP is highly degenerate, then the simplex method typically yields "very misleading" dual solutions, and (b) when randomisation is used, the dual solutions obtained via dual ascent may be "much more useful" than optimal dual solutions obtained by either simplex or interior-point methods.

The main lesson from our work is that, when faced with a given combinatorial optimisation problem, it is well worth coding and testing several different heuristics for finding dual solutions, before jumping to conclusions about the effectiveness of dual-based heuristics. We believe that it is worth coding and testing more than one primal heuristic as well.

Chapter 6

Conclusion

In this final chapter, we summarise the main contributions of this thesis and make some suggestions for further research.

6.1 Summary

In Chapter 1, we introduced the field of integer programming and combinatorial optimisation, and gave an overview of the key concepts (such as algorithms, $\mathcal{NP}$-hardness, relaxation, duality, branching, cutting, and heuristics). We also argued that there was a need for further work on various relaxations (especially surrogate relaxation), and on matheuristics based on those relaxations.

In Chapter 2, we presented some theoretical results related to surrogate and group relaxation. We proved that, when only *inequalities* are surrogated, the surrogate dual is solvable in pseudo-polynomial time under certain conditions. (These conditions hold, for example, when the surrogate relaxed problem is a 0-1 or general-integer Knapsack Problem.) We also found that, when only *equations* are surrogated, the surrogate dual exhibits unusual complexity behaviour: the optimum multipliers can be found in polynomial time, but the solution of the relaxed problem itself is $\mathcal{NP}$-hard to compute. Finally, we showed that the group relaxation is $\mathcal{NP}$-hard for the 0-1 and general-integer Knapsack Problems, and *strongly* $\mathcal{NP}$-hard for the Set Packing Problem.

In Chapter 3, we considered surrogate relaxation for the Multidimensional Knapsack Problem (MKP). We presented a new exact algorithm for solving the surrogate dual, based on the simplex method, cutting planes and binary search. (We remark that our algorithm is completely different from the classical methods for the surrogate dual, which are all heuristic in nature and based on variations of the subgradient method.) We also presented a primal heuristic, which attempts to exploit information generated in the cutting-plane phase.

The computational results in Chapter 3 were mixed. On the positive side, our algorithm is fast, being the first to solve the surrogate dual exactly for MKP instances of meaningful size. Moreover, the primal heuristic tends to produce solutions of good quality. On the other hand, the upper bounds from the surrogate dual were typically only a little better than the ones obtained from the standard Linear Programming (LP) relaxation. (This last fact is surprising, given that many authors have recommended applying surrogate relaxation to the MKP.)

In Chapter 4, we explored in more depth the potential of using surrogate relaxation within matheuristics for combinatorial optimisation problems. We selected three specific problems: the MKP, the 3-dimensional Assignment Problem (AP3) and the Simple Plant Location Problem (SPLP). To set the surrogate multipliers for each problem, we used a fast heuristic, based on scaling and rounding certain dual prices from the LP relaxation. We then solve the surrogate relaxation via Dynamic Programming (DP), and use the output from the DP to construct several 'partial solutions' to the original problem. The matheuristic then 'repairs' each of the partial solutions to make it feasible.

The computational results in Chapter 4 were again mixed. The matheuristics gave excellent primal solutions for the MKP and AP3, but not for the SPLP. Moreover, the bounds that we obtained by applying surrogate relaxation (upper bounds for the MKP and lower bounds for the AP3 and SPLP) were somewhat disappointing, often being even weaker than the standard LP bounds. (This may be because we were using a simple heuristic to generate the multipliers.)

In Chapter 5, we turn our attention to matheuristics that are based on LP duality rather than surrogate duality. We investigate (a) whether LP dual information is useful at all for driving a matheuristic, and (b) whether it is necessary to have optimal (or near-optimal) dual solutions in order to get good solutions from dual-based matheuristics. For this, we considered two specific combinatorial optimisation problems: the SPLP and the Set Covering Problem (SCP). For each problem, we considered several ways to solve the LP dual: the simplex method, an interior-point method, and dual ascent. We also developed several deterministic and randomised matheuristics for each of the three problems.

In our view, the computational results in Chapter 5 show convincingly that it is generally better to use dual information than not. More surprisingly, we found that good dual solutions do not always lead to good primal solutions. Indeed, for both the SCP and SPLP, the primal solutions tended to be better when the dual LP was solved approximately, by dual ascent, than when it was solved exactly, by the simplex method. Moreover, in the case of the SPLP, the primal solutions tended to be much better when the dual LP was solved by an interior-point method rather than the simplex method. In other words, when the primal LP is massively degenerate, a "central" near-optimal dual solution tends to be much more useful than to an "extreme" optimal one.

6.2 Suggestions for Future Research

In this last section, we mention some possible avenues for future research.

In Chapter 2, we presented some complexity results for surrogate relaxation. A closely related technique is *composite* relaxation (CR), which is a hybrid of Lagrangian and surrogate relaxation [75,79]. Consider again an ILP of the form (2.1). In CR, we choose vectors $\lambda, \mu \in \mathbb{Q}^m_+$, and solve the relaxed problem

$$\max\Big\{c^T x + \lambda^T\big(b - Ax\big) : \big(\mu^T A\big)x \le \mu^T b,\ x \in \mathcal{X}\Big\}.$$

In theory, CR can produce better upper bounds than both Lagrangian and surrogate relaxation. Unfortunately, the dual function is not even quasi-convex in λ and μ. In fact, it may have local minima that are not global minima [97]. Thus, in practice, the "composite dual" is even harder to solve than the surrogate dual. It would be good to settle the complexity status of the composite dual and related problems. We conjecture that, even in the inequality case, the composite dual is strongly $\mathcal{NP}$-hard. As for the Level Set Problem in CR, we do not even know whether it lies in $\mathcal{NP}$ or co-$\mathcal{NP}$.

In Chapter 3, we implemented an exact algorithm for solving the surrogate dual of the MKP. Although the algorithm was simplex-based, and therefore not guaranteed to converge in pseudo-polynomial time, it performed very well in practice. The natural next step would be to adapt the algorithm to other combinatorial optimisation problems with inequality constraints, such as the SCP. Another interesting topic for future research would be to find an efficient way to embed our upper-bounding procedure within an exact branch-and-bound algorithm for the MKP.

In Chapter 4 we showed that surrogate relaxation can also be used to good effect within a matheuristic scheme. There are several possible ways in which the matheuristic scheme could be extended or improved. First, one could adapt it to other combinatorial optimisation problems. Second, one could attempt to improve the primal solutions, for example by using some kind of local search procedure. Lastly, one could experiment with alternative methods for computing the surrogate multipliers. (In principle, one could use one of the existing iterative methods for the surrogate dual, mentioned in Subsection 5.2.1. One would however have to be careful to get a good trade-off between bound quality and running time.)

Finally, in Chapter 5, we explored the impact of dual solution quality on the performance of dual-based heuristics. The most surprising outcome is that clearly sub-optimal dual solutions (obtained via dual ascent) are often more useful than provably optimal dual solutions (obtained by simplex or interior-point methods). It would be worthwhile experimenting with some other combinatorial optimisation problems, to see if this phenomenon appears elsewhere. It would also be good to perform analogous experiments with heuristic based on Lagrangian relaxation. (That is, do good Lagrangian multipliers tend to lead to good primal solutions?)

The main lesson from our work is that, if one wishes to compute both lower and upper bounds for any given combinatorial optimisation problem, it is well worth coding and testing (a) several different relaxations, (b) several different methods for solving the relaxations (either exactly or approximately), and (c) several different primal heuristics. If care is taken to design and analyse the algorithms, and appropriate data structures are used when implementing them, one can often obtain bounds of good quality in very reasonable computing times.

Bibliography

[1] E.H.L. Aarts & J.K. Lenstra (eds) (2003) *Local Search in Combinatorial Optimization.* Princeton, NJ: Princeton University Press.

[2] S. Ahn, C. Cooper, G. Cornuéjols & A.M. Frieze (1988) Probabilistic analysis of a relaxation for the p-median problem. *Math. Oper. Res.*, 13, 1–31.

[3] E. Angelelli, R. Mansini & M.G. Speranza (2010) Kernel search: a general heuristic for the multi-dimensional knapsack problem. *Comput. Oper. Res.*, 37, 2017–2026.

[4] A. Balakrishnan, T.L. Magnanti & R.T. Wong (1989) A dual-ascent procedure for large-scale uncapacitated network design. *Oper. Res.*, 37, 716–740.

[5] E. Balas & A. Ho (1980) Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Math. Program. Study*, 12, 37–60.

[6] E. Balas & M.J. Saltzman (1991) An algorithm for the three-index assignment problem. *Oper. Res.*, 39, 150–161.

[7] E. Balas & M.C. Carrera (1996) A dynamic subgradient-based branch-and-bound procedure for set covering. *Oper. Res.*, 44, 875–890.

[8] M. Balinski (1965) Integer programming: methods, uses, computation. *Manag. Sci.*, 12, 254–313.

[9] M.O. Ball (2011) Heuristics based on mathematical programming. *Surveys in Oper. Res. & Manag. Sci.*, 16, 21–38.

[10] C. Barnhart (1993) Dual-ascent methods for large-scale multi-commodity flow problems. *Nav. Res. Logist.*, 40, 305–324.

[11] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh & P.H. Vance (1998) Branch-and-price: column generation for solving huge integer programs. *Oper. Res.*, 46, 316–329.

[12] M.S. Bazaraa, H.D. Sherali & C.M. Shetty (2006) *Nonlinear Programming: Theory and Algorithms*, 3rd edn. Hoboken, NJ: Wiley.

[13] J.E. Beasley (1987) An algorithm for set covering problems. *Eur. J. Oper. Res.*, 31, 85–93.

[14] J.E. Beasley (1990) A Lagrangian heuristic for set covering problems. *Nav. Res. Logist.*, 37, 151–164.

[15] J.E. Beasley (1990) OR-Library: distributing test problems by electronic mail. *J. Oper. Res. Soc.*, 41, 1069–1072.

[16] J.E. Beasley (1993) Lagrangian heuristics for location problems. *Eur. J. Oper. Res.*, 65, 383–399.

[17] J.E. Beasley & K. Jörnsten (1992) Enhancing an algorithm for set covering problems. *Eur. J. Oper. Res.*, 58, 293–300.

[18] R.E. Bellman (1957) *Dynamic Programming.* Princeton, NJ: Princeton University Press.

[19] C. Beltran-Royo, J.-P. Vial & A. Alonso-Ayuso (2012) Semi-Lagrangian relaxation applied to the uncapacitated facility location problem. *Comput. Optim. Appl.*, 51, 387–409.

[20] D.P. Bertsekas (2016) *Nonlinear Programming*, 3rd edn. Belmont, MA: Athena Scientific.

[21] O. Bilde & J. Krarup (1977) Sharp lower bounds and efficient algorithms for the simple plant location problem. *Ann. Discr. Math.*, 1, 79–88.

[22] N. Boland, A.C. Eberhard & A. Tsoukalas (2015) A trust region method for the solution of the surrogate dual in integer programming. *J. Optim. Th. Appl.*, 167, 558–584.

[23] E. Boros (1986) On the complexity of the surrogate dual of 0–1 programming. *Zeit. Oper. Res.*, 30, A145–A153.

[24] M.A. Boschetti, A. Mingozzi & S. Ricciardelli (2008) A dual ascent procedure for the set partitioning problem. *Discr. Optim.*, 5, 735–747.

[25] M. Boschetti, V. Maniezzo & M. Roffilli (2010) Decomposition techniques as metaheuristic frameworks. In V. Maniezzo, T. Stützle & S. Voß (eds) *Matheuristics*, pp. 135–158. Boston, MA: Springer.

[26] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi & P. Michelon (2010) A multi-level search strategy for the 0–1 multidimensional knapsack problem. *Discr. Appl. Math.*, 158, 97–109.

[27] S. Boyd & L. Vandenberghe (2004) *Convex Optimization.* Cambridge: Cambridge University Press.

[28] R. Burkard, M. Dell'Amico & S. Martello (2012) *Assignment Problems* (Revised Reprint). Philadelphia, PA: SIAM.

[29] R. Burkard, R. Rudolf & G. Woeginger (1996) Three-dimensional axial assignment problems with decomposable cost coefficients *Discr. Appl. Math.*, 65, 123–139.

[30] E.K. Burke & G. Kendall (eds) (2014) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (2nd edn.). New York: Springer.

[31] E. Buson, R. Roberti & P. Toth (2014) A reduced-cost iterated local search heuristic for the fixed-charge transportation problem. *Oper. Res.*, 62, 1095–1106.

[32] A. Caprara, M. Fischetti & P. Toth (1999) A heuristic method for the set covering problem. *Oper. Res.*, 47, 730–743.

[33] A. Caprara, M. Fischetti & P. Toth (2000) Algorithms for the set covering problem. *Ann. Oper. Res.*, 98, 353–371.

[34] S. Ceria, P. Nobili & A. Sassano (1998) A Lagrangian-based heuristic for large-scale set covering problems. *Math. Program.*, 81, 215–228.

[35] D.-S. Chen & S. Zionts (1976) Comparison of some algorithms for solving the group theoretic integer programming problem. *Oper. Res.*, 24, 1120–1128.

[36] D.-S. Chen, R.G. Batson & Y. Dang (2010) *Applied Integer Programming.* Hoboken, NJ: Wiley.

[37] P.C. Chu & J.E. Beasley (1998) A genetic algorithm for the multidimensional knapsack problem. *J. Heur.*, 4, 63–86.

[38] M. Conforti, G. Cornuéjols & G. Zambelli (2014) *Integer Programming.* Cham, Switzerland: Springer.

[39] W.J. Cook (2010) Fifty-plus years of combinatorial integer programming. In: M. Jünger *et al.* (eds) *50 Years of Integer Programming: 1958-2008*, pp. 387–430. Berlin: Springer.

[40] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank & A. Schrijver (1998) *Combinatorial Optimization.* New York: Wiley.

[41] G. Cornuéjols, C. Michini & G. Nannicini (2012) How tight is the corner relaxation? Insights gained from the stable set problem. *Discr. Optim.*, 9, 109–121.

[42] Y. Crama & J.B. Mazzola (1994) On the strength of relaxations of multidimensional knapsack problems. *INFOR*, 32, 219–225.

[43] Y. Crama & F. Spieksma (1992) Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *Eur. J. Oper. Res.*, 60, 273-279.

[44] G.B. Dantzig (1960) On the significance of solving linear programming problems with some integer variables. *Econometrica*, 28, 30–44.

[45] G.B. Dantzig (1998) *Linear Programming and Extensions* (revised edn). Princeton, NJ: Princeton University Press.

[46] G.B. Dantzig & P. Wolfe (1960) Decomposition principle for linear programs. *Oper. Res.*, 8, 101–111.

[47] F. Della Croce & A. Grosso (2012) Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem. *Comput. Oper. Res.*, 39, 27–31.

[48] T. Dokka, A.N. Letchford & M.H. Mansoor (2021) On the complexity of surrogate and group relaxation for integer linear programs. *Oper. Res. Lett.*, 49, 530–534.

[49] T. Dokka, I. Mourtos & F.C. Spieksma (2017) Fast separation for the three-index assignment problem. *Math. Program. Comput.*, 9, 39–59.

[50] M.E. Dyer (1980) Calculating surrogate constraints. *Math. Program.*, 19, 255–278.

[51] D. Erlenkotter (1978) A dual-based procedure for uncapacitated facility location. *Oper. Res.*, 26, 992–1009.

[52] E. Feldman, F.A. Lehrer & T.L. Ray (1966) Warehouse location under continuous economies of scale. *Manag. Sci.*, 12, 670–684.

[53] M. Fischetti (2019) *Introduction to Mathematical Optimization.* Independently published.

[54] M. Fischetti & M. Monaci (2008) How tight is the corner relaxation? *Discr. Optim.*, 5, 262–269.

[55] M. Fischetti, I. Ljubić, M. Sinnl (2017) Redesigning Benders decomposition for large-scale facility location. *Manag. Sci.*, 63, 2146–2162.

[56] M.L. Fisher (1981) The Lagrangian relaxation method for solving integer programming problems. *Manag. Sci.*, 27, 1–18.

[57] M.L. Fisher (1985) An applications-oriented guide to Lagrangian relaxation. *Interfaces*, 15, 10–21.

[58] M.L. Fisher, R. Jaikumar & L.N. Van Wassenhove (1986) A multiplier adjustment method for the generalized assignment problem. *Manag. Sci.*, 32, 1095–1103.

[59] M.L. Fisher & P. Kedia (1990) Optimal solution of set covering/partitioning problems using dual heuristics. *Manag. Sci.*, 36, 674–688.

[60] M.L. Fisher, B.J. Lageweg, J.K. Lenstra & A.R. Kan (1983) Surrogate duality relaxation for job shop scheduling. *Discr. Appl. Math.*, 5, 65–75.

[61] A. Fréville (2004) The multidimensional 0–1 knapsack problem: an overview. *Eur. J. Oper. Res.*, 155, 1–21.

[62] A. Fréville & S. Hanafi (2005) Multidimensional 0–1 knapsack problem: bounds and computational aspects. *Ann. Oper. Res.*, 139, 195–227.

[63] A. Fréville & G. Plateau (1993) An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem. *Eur. J. Oper. Res.*, 68, 413–421.

[64] A. Fügenschuh & B. Höfler (2006) Parametrized GRASP heuristics for three-index assignment. In J. Gottlieb & G.R. Raidl (eds) *Evolutionary Computation in Combinatorial Optimization*, pp. 61–72. Berlin: Springer.

[65] R.D. Galvão, L.G. Acosta Espejo & B. Boffey (2000) A comparison of Lagrangean and surrogate relaxations for the maximal covering location problem. *Eur. J. Oper. Res.*, 124, 377–389.

[66] R. Galvão & L. Raggi (1989) A method for solving to optimality uncapacitated location problems. *Ann. Oper. Res.*, 18, 225–244.

[67] M.R. Garey & D.S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.

[68] B. Gavish & H. Pirkul (1985) Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Math. Program.*, 31, 78–105.

[69] Gendreau, M., & Potvin, J. Y. (eds.) (2010) *Handbook of metaheuristics* (Vol. 2, p. 9). New York: Springer.

[70] A.M. Geoffrion (1974) Lagrangean relaxation for integer programming. *Math. Program. Study*, 2, 82–114.

[71] D. Ghosh (2003) Neighborhood search heuristics for the uncapacitated facility location problem. *Eur. J. Oper. Res.*, 150, 150–162.

[72] K.C. Gilbert & R.B. Hofstra (1988) Multidimensional assignment problems. *Decis. Sci.*, 19, 306–321.

[73] F. Glover & R. Woolsey (1972) Aggregating Diophantine equations. *Zeit. Oper. Res.*, 16, 1–10.

[74] F. Glover (1989). Tabu search—part I. *ORSA Journal on computing*, 1(3), 190–206.

[75] F. Glover (1975) Surrogate constraint duality in mathematical programming. *Oper. Res.*, 23, 434–451.

[76] R.E. Gomory (1958) Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.*, 64, 275–278.

[77] R.E. Gomory (1965) On the relation between integer and non-integer solutions to linear programs. *Proc. Nat. Acad. Sci.*, 53, 260–265.

[78] R.E. Gomory (1969) Some polyhedra related to combinatorial problems. *Lin. Alg. Appl.*, 2, 451–558.

[79] H.J. Greenberg & W.P. Pierskalla (1970) Surrogate mathematical programming. *Oper. Res.*, 18, 924–939.

[80] M. Grötschel, L. Lovász & A.J. Schrijver (1988) *Geometric Algorithms and Combinatorial Optimization.* New York: Wiley.

[81] D.A. Grundel & P.M. Pardalos (2005) Test problem generator for the multidimensional assignment problem. *Comput. Optim. Appl.*, 30, 133–146.

[82] H. Gu (2018) Local cuts for 0-1 multidimensional knapsack problems. In R. Sarker *et al.* (eds) *Data and Decision Sciences in Action.* Cham, Switzerland: Springer.

[83] M. Guignard (1988) A Lagrangian dual ascent algorithm for simple plant location problems. *Eur. J. Oper. Res.*, 35, 193–200.

[84] M. Guignard (2003) Lagrangean relaxation. *Trabajos de Operativa (TOP)*, 11, 151–228.

[85] S. Haddadi (1997) Simple Lagrangian heuristic for the set covering problem. *Eur. J. Oper. Res.*, 97, 200–204.

[86] P. Hansen, J. Brimberg, D. Urošević & N. Mladenovič (2007) Primal-dual variable neighborhood search for the simple plant-location problem. *INFORMS J. Comput.*, 19, 552–564.

[87] D. Harel & Y. Feldman (2004) *Algorithmics: The Spirit of Computing* (3rd edn). Harlow, Essex: Addison-Wesley.

[88] E. Helly (1923) Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32, 175–176.

[89] D.S. Hochbaum (1982) Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11, 555–556.

[90] A.J. Hoffman & J.B. Kruskal (1956) Integral boundary points of convex polyhedra. In H.W. Kuhn & A.J. Tucker (eds.) *Linear Inequalities and Related Systems*, pp. 223—246. Princeton, NJ: Princeton University Press.

[91] J.H. Holland (1975) *Adaption in Natural and Artificial Systems.* Cambridge, MA: MIT Press.

[92] L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai, & L. Shi (2021) *Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends.* arXiv:2111.06257.

[93] J. Janáček & L. Buzna (2008) An acceleration of Erlenkotter–Körkel's algorithms for the uncapacitated facility location problem. *Ann. Oper. Res.*, 164, 97–109.

[94] M. Jünger *et al.* (eds) (2009) *50 Years of Integer Programming: 1958–2008.* Berlin: Springer.

[95] K. Kaparis & A.N. Letchford (2008) Local and global lifted cover inequalities for the multidimensional knapsack problem. *Eur. J. Oper. Res.*, 186, 91–103.

[96] R.M. Karp (1972) Reducibility among combinatorial problems. In R.E. Miller *et al.* (eds) *Complexity of Computer Computations*, pp. 85–103. New York: Plenum.

[97] M.H. Karwan & R.L. Rardin (1980) Searchability of the composite and multiple surrogate dual functions. *Oper. Res.*, 28, 1251–1257.

[98] M.H. Karwan & R.L. Rardin (1984) Surrogate dual multiplier search procedures in integer programming. *Oper. Res.*, 32, 352–69.

[99] H. Kellerer, U. Pferschy & D. Pisinger (2004) *Knapsack Problems.* Berlin: Springer.

[100] S.-L. Kim & S. Kim (1998) Exact algorithm for the surrogate dual of an integer programming problem: subgradient method approach. *J. Optim. Th. Appl.*, 96, 363–375.

[101] S. Kirkpatrick, C.D. Gelatt & M.P. Vecchi (1983) Optimization by simulated annealing. *Science*, 220, 671–680.

[102] V. Klee & G.J. Minty (1972) How good is the simplex algorithm? In O. Shisha (ed.) *Proceedings of the Third Symposium on Inequalities*, pp. 159–175. New York: Academic Press.

[103] Y. Kochetov & D. Ivanenko (2005) Computationally difficult instances for the uncapacitated facility location problem. In T. Ibaraki, K. Nonobe & M. Yagiura (eds) *Metaheuristics: Progress as Real Problem Solvers*, pp. 351–367. Boston, MA: Springer.

[104] M. Körkel (1989) On the exact solution of large-scale simple plant location problems. *Eur. J. Oper. Res.*, 39, 157–173.

[105] B.H. Korte & J. Vygen (2018) *Combinatorial Optimization: Theory and Algorithms*, 6th edn. Heidelberg: Springer.

[106] J. Krarup & P.M. Pruzan (1983) The simple plant location problem: survey and synthesis. *Eur. J. Oper. Res.*, 12, 36–81.

[107] J. Kratica, D. Tosic, V. Filipovic & I. Ljubic (2001) Solving the simple plant location problem by genetic algorithm. *RAIRO Oper. Res.*, 35, 127–142.

[108] A.H. Land & A.G. Doig (1960) An automatic method of solving discrete programming problems. *Econometrica*, 28, 497–520.

[109] C. Lemaréchal (2001) Lagrangian relaxation. In M. Jünger & D. Naddef (eds) *Computational Combinatorial Optimization*, pp. 115–160. Heidelberg: Springer Verlag.

[110] A.N. Letchford (2003) Binary clutter inequalities for integer programs. *Math. Program.*, 98, 201–221.

[111] A.N. Letchford & S.J. Miller (2012) Fast bounding procedures for large instances of the simple plant location problem. *Comput. Oper. Res.*, 39, 985–990.

[112] L.A.N. Lorena & F.B. Lopes (1994) A surrogate heuristic for set covering problems. *Eur. J. Oper. Res.*, 79, 138–150.

[113] G.S. Lueker (1975) Two NP-complete problems in nonnegative integer programming. *Technical Report No. 178*, Department of Electrical Engineering, Princeton University.

[114] V. Maniezzo, T. Stützle & S. Voß (eds) (2010) *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming.* Boston, MA: Springer.

[115] V. Maniezzo, M.A. Boschetti & T. Stützle (2021) *Matheuristics: Algorithms and Implementations.* Cham: Springer.

[116] R. Mansini & M.G. Speranza (2012) CORAL: an exact algorithm for the multidimensional knapsack problem. *INFORMS J. Comput.*, 24, 399–415.

[117] R. Martí, P.M. Pardalos & M.G. Resende (eds) (2018) *Handbook of Heuristics.* Cham: Springer.

[118] R.R. Meyer (1974) On the existence of optimal solutions to integer and mixed-integer programming problems. *Math. Program.*, 7, 223–235.

[119] J.E. Mitchell (2011) Branch and cut. In J.J. Cochran *et al.* (eds) *Wiley Encyclopedia of Operations Research and Management Science.* New York: Wiley.

[120] J.G. Morris (1978) On the extent to which certain fixed-charge depot location problems can be solved by LP. *J. Oper. Res. Soc*, 29, 71–76.

[121] M.G. Narciso & L.A.N. Lorena (1999) Lagrangean/surrogate relaxation for generalized assignment problems. *Eur. J. Oper. Res.*, 114, 165–177.

[122] G.L. Nemhauser & L.A. Wolsey (1988) *Integer and Combinatorial Optimization.* New York: Wiley.

[123] M.W. Padberg (1973) On the facial structure of set packing polyhedra. *Math. Program.*, 5, 199–215.

[124] M.W. Padberg & G. Rinaldi (1987) Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Oper. Res. Lett.*, 6, 1–7.

[125] G. Polya (1945) *How to Solve It.* Princeton, NJ: Princeton University Press.

[126] T. Polzin & S.V. Daneshmand (2001) Improved algorithms for the Steiner problem in networks. *Discr. Appl. Math.*, 112, 263–300.

[127] M. Posta, J.A. Ferland & P. Michelon (2014) An exact cooperative method for the uncapacitated facility location problem. *Math. Program. Comput.*, 6, 199–231.

[128] J. Puchinger, G.R. Raidl & U. Pferschy (2010) The multidimensional knapsack problem: structure and algorithms. *INFORMS J. Comput.*, 22, 250–265.

[129] P. Raghavan & C.D. Tompson (1987) Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7, 365–374.

[130] G.R. Raidl (2015) Decomposition based hybrid metaheuristics. *Eur. J. Oper. Res.*, 244, 66–76.

[131] C.R. Reeves (ed.) (1993) *Modern Heuristic Techniques for Combinatorial Problems.* New York: Wiley.

[132] J.-P. Richard & S.S. Dey (2010) The group-theoretic approach in mixed integer programming. In M. Juenger *et al.* (eds) *50 Years of Integer Programming 1958–2008*, pp. 727–801. Berlin: Springer.

[133] R. Saket & M. Sviridenko (2012) New and improved bounds for the minimum set cover problem. In A. Gupta *et al.* (eds), *Proc. APPROX-RANDOM XV*, pp. 288–300. Berlin: Springer.

[134] S. Sarin, M.H. Karwan & R.L. Rardin (1987) A new surrogate dual multiplier search procedure. *Nav. Res. Logist.*, 34, 431–450.

[135] S. Sarin, M.H. Karwan & R.L. Rardin (1988) Surrogate duality in a branch-and-bound procedure for integer programming. *Eur. J. Oper. Res.*, 33, 326–333.

[136] A. Schrijver (1986) *Theory of Linear and Integer Programming.* Chichester: Wiley.

[137] M.J. Todd (1982) An implementation of the simplex method for linear programming problems with variable upper bounds. *Math. Program.*, 23, 34–49.

[138] UflLib: a library of instances of the uncapacitated facility location problem. Available at `http://resources.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/`

[139] R.J. Vanderbei (2020) *Linear Programming: Foundations and Extensions*, 5th edn. Cham: Springer.

[140] V.V. Vazirani (2001) *Approximation Algorithms.* Berlin: Springer.

[141] V. Verter (2011) Uncapacitated and capacitated facility location problems. In H.A. Eiselt & V. Marianov (eds) *Principles of Location Science*, pp. 25–37. Heidelberg: Springer.

[142] Y. Vimont, M. Boussier, M. Vasquez (2008) Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *J. Combin. Optim.*, 15, 165–178.

[143] H.P. Williams (2013) *Model Building in Mathematical Programming*, 5th edn. New York: Wiley.

[144] D.P. Williamson & D.B. Shmoys (2011) *The Design of Approximation Algorithms.* Cambridge: Cambridge University Press.

[145] D.P. Williamson (2002) The primal-dual method for approximation algorithms. *Math. Program.*, 91, 447–478.

[146] R.T. Wong (1984) A dual ascent approach for Steiner tree problems on a directed graph. *Math. Program.*, 28, 271–287.

[147] B. Yelbay, Ş.İ. Birbil & K. Bülbül (2015) The set covering problem revisited: an empirical study of the value of dual information. *J. Indust. Manag. Optim.*, 11, 575–594.