2009

New Heuristics For The 0-1 Multi-dimensional Knapsack Problems

Haluk Akin
University of Central Florida

NEW HEURISTICS FOR THE 0-1 MULTI-DIMENSIONAL KNAPSACK PROBLEMS

by

M. HALUK AKIN
B.S. Yildiz Technical University, 2002
M.S. University of Central Florida, 2004

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Industrial Engineering
in the Department of Industrial Engineering and Management Systems
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2009

Major Professor: José A. Sepúlveda

ABSTRACT

This dissertation introduces new heuristic methods for the 0-1 multi-dimensional knapsack problem (0-1 MKP). 0-1 MKP can be informally stated as the problem of packing items into a knapsack while staying within the limits of different constraints (dimensions). Each item has a profit level assigned to it. They can be, for instance, the maximum weight that can be carried, the maximum available volume, or the maximum amount that can be afforded for the items. One main assumption is that we have only one item of each type, hence the problem is binary (0-1). The single dimensional version of the 0-1 MKP is the uni-dimensional single knapsack problem which can be solved in pseudo-polynomial time. However the 0-1 MKP is a strongly NP-Hard problem.

Reduced cost values are rarely used resources in 0-1 MKP heuristics; using reduced cost information we introduce several new heuristics and also some improvements to past heuristics. We introduce two new ordering strategies, decision variable importance (DVI) and reduced cost based ordering (RCBO). We also introduce a new greedy heuristic concept which we call the "sliding concept" and a sub-branch of the "sliding concept" which we call "sliding enumeration". We again use the reduced cost values within the sliding enumeration heuristic.

RCBO is a brand new ordering strategy which proved useful in several methods such as improving Pirkul's MKHEUR, a triangular distribution based probabilistic approach, and our own sliding enumeration.

We show how Pirkul's shadow price based ordering strategy fails to order the partial variables. We present a possible fix to this problem since there tends to be a high number of

partial variables in hard problems. Therefore, this insight will help future researchers solve hard problems with more success.

Even though sliding enumeration is a trivial method it found optima in less than a few seconds for most of our problems. We present different levels of sliding enumeration and discuss potential improvements to the method.

Finally, we also show that in meta-heuristic approaches such as Drexl's simulated annealing where random numbers are abundantly used, it would be better to use better designed probability distributions instead of random numbers.

This dissertation is dedicated to my parents Neslihan and Yuksel and to my lovely sister Zeynep.

ACKNOWLEDGMENTS

If there is one person who truly made this dissertation possible, it is Dr. Sepulveda. Over the years he taught me many academic concepts but more importantly he taught me how to learn, how to be objective, how to think out of the box, how to listen and many more important skills that I can't finish counting. He single handedly guided me through the PhD, I cannot count the number of instances where he bailed me out of trouble. Whenever he told me to trust him, he proved to be right. I will do my best to pass on his teachings to my students and it is my utmost wish to be worthy of the time he spent on me.

Dr. Rabelo served like a co-mentor during my time at UCF. From the first day I started working with Dr. Sepulveda I've also worked with Dr. Rabelo. He taught me many things but most importantly time and time again he showed how to think for tomorrow and not today. Clearly he is a true visionary. I'm very glad that I have been a student of his.

Dr. Reilly has always been one of my favorite professors at UCF. I've always admired his passion for the theory of operations research. He is truly one of the most well known researchers in my research area. I'm very lucky he accepted to be on my committee. This dissertation has benefited a lot from his attention to detail and his thoroughness.

I would like to thank Dr. Moraga for accepting to be on my committee despite his distance and time zone difference from Orlando. Having an expert like him who is focused in my field was a fortune for me. I appreciate his questions and comments throughout the whole dissertation process.

I would like to thank Dr. Nazzal for agreeing to be on my committee. As a young professor on campus she already had enough tasks to handle when she accepted my request. All her questions

and contributions are well appreciated. UCF is truly lucky to have a smart and young faculty member like her.

Deniz Karabacak enabled additional financial support for my doctorate education from the first day of my studies. Few other people assist graduate students in such visionary manner. I will always be grateful to her. She is truly unique and her support will never be forgotten.

I owe a lot to Ilknur Demir and her ever-lasting patience during all the downs of the last few years. Not only she helped me with the personal side of the doctorate she also supported me with all my tough times at UCF. I was fortunate to have her by my side; she is a very special person.

Honesty, hard-work, patience, integrity and strong will are just a few of the things I've seen from my grandparents. With every new day, I understand more and more how blessed I was to have them teach me the basics of life; they truly built the core of my personality. I hope one day I can be as mature, as understanding, as humble, and as wise as they are and they were.

My parents and my sister shouldered the burden of being apart for the last years. I hope the coming years will allow us to cover for the time we lost. They have always been inspirational for me. They always supported and motivated me for the better. They are truly amazing and irreplaceable.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

0-1 MKP	0-1 Multi-Dimensional Knapsack Problem
AVG	Average
DVI	Decision Variable Importance
LP	Linear Program
MAX	Maximize
MIN	Minimize
OOP	Object Oriented Programming
RCBO	Reduced Cost Based Ordering
SA	Simulated Annealing
ST. DEV.	Standard Deviation
ST	Subject to
UML	Unified Modeling Language

CHAPTER ONE: INTRODUCTION

This dissertation introduces new heuristics for the 0-1 multi-dimensional knapsack problem (0-1 MKP). Reduced cost is a rarely used resource in 0-1 MKP heuristics; using reduced cost information we improve two past heuristics, one due to Drexl (1988) and another due to Pirkul (1987). Additionally, we also introduce a new greedy heuristic concept which we would like to call the "sliding concept". We also introduce a sub-branch of the sliding concept which we would like to call "sliding enumeration". Once we combine sliding enumeration and a new decision variable ordering technique which we call "reduced cost based ordering" (RCBO) we get very competitive results for the 0-1 MKP.

The 0-1 MKP is a well known strongly NP-Hard problem which has been frequently researched and continues to receive significant attention. It is defined as follows (Fréville, 2004).

$$\begin{aligned} &\max z = \sum_{j=1}^{n} c_j x_j, \\ &s.t. \\ &\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1,\ldots,m \\ &x_j \in \{0,1\}, \qquad\qquad j = 1,\ldots,n \end{aligned} \tag{1}$$

- c_j is the profit associated with x_j
- a_{ij} is the weight associated with each x_j and dimension i combination.
- b_i is the maximum weight which dimension i can carry.

We will also present some additional necessary assumptions in later sections.

There are two significant motivations behind the academic interest. First, the 0-1 MKP is a special version of the general zero-one integer programming problem. Second, due to its well known NP-Hardness, many researchers choose the 0-1 MKP as a test bed for their new heuristics.

Algorithms and heuristics are two main approaches to solving optimization problems. When algorithms cannot solve a problem in a reasonable time usually heuristic methods are employed. More often than not, it is not practical to solve NP-Hard optimization problems with algorithms; therefore a lot of research has been dedicated to solving NP-Hard problems with heuristic approaches. This document focuses on the use of heuristic approaches on the 0-1 MKP. With the knowledge we have gained throughout this research not only we were able to come up with new heuristics, we were also able to improve some past heuristics with minor modifications.

Most of the heuristics we will introduce in this document will rely on the use of "reduced costs" generated as a result of applying the bounded variable simplex method. Even though reduced costs are a well known part of sensitivity analysis, to the best of our knowledge they have rarely been used as a heuristic input for 0-1 MKPs.

One important heuristic we will introduce in this document is called "sliding enumeration". It is a type of greedy heuristic where at each step instead of evaluating only one variable at a time we are evaluating a group of variables to help fix the next variable. For the purposes of this research we have used only enumeration in the context of sliding, hence the name sliding enumeration, however it is possible to replace the "enumeration" part with another algorithm as well. We will explain the advantages and disadvantages of other potential algorithms within the "sliding" concept.

This dissertation will be organized as follows. First, we will introduce the uni-dimensional knapsack problem and then the 0-1 MKP. Following that, we will examine the literature to see previous approaches for solving the 0-1 MKP. Next we will discuss the differences between algorithms and heuristics. Then we will introduce our own heuristic ideas. Before conclusion, we will present the experiment results on well known problem sets from the literature. Finally we will conclude with a short discussion and recommendations for future research.

CHAPTER TWO: 0-1 MULTI-DIMENSIONAL KNAPSACK PROBLEM

Before we discuss the 0-1 MKP we would like to visit the uni-dimensional 0-1 knapsack problem. This problem is equivalent to the problem of a trekker deciding which items should be packed into a knapsack. Our assumption is that the trekker has only one item of each type and all the items have a benefit value assigned to them. These items have weights assigned to them. Additionally, it is also not possible to pack all the items together. So the trekker has to leave some of the items out, obviously otherwise there would not be a problem to solve if we could pack all the items to the knapsack. The 0-1 knapsack problem is modeled as follows:

$$\begin{aligned} &\max z = \sum_{j=1}^{n} c_j x_j, \\ &s.t. \\ &\sum_{j=1}^{n} a_j x_j \le b \\ &x_j \in \{0,1\}, \quad j = 1,\ldots,n \end{aligned} \tag{2}$$

The uni-dimensional 0-1 knapsack problem is not strongly NP-Hard (Chu & Beasley, 1998). Several effective algorithms have been developed to tackle this type of problem. For a review of these algorithms, we refer the reader to Martello and Toth (1990) and Kellerer, Pferschy and Pisinger (2004).

The fairly simple structure of the uni-dimensional 0-1 knapsack problem could make us think that a linear programming (LP) relaxation could possibly produce a close-to-optimum solution for a knapsack problem, but that is far from reality. The following is a small instance

used by Vasquez and Hao (2001a) to showcase how the LP relaxation can drop us far from the optimum.

$$\max \; z = 12x_1 + 12x_2 + 9x_3 + 8x_4 + 8x_5$$
$$s.t.$$
$$11x_1 + 12x_2 + 10x_3 + 10x_4 + 10x_5 \leq 30$$
$$x_j \in \{0,1\}, \qquad j = 1,...,5$$

In this case,

x^{LP} = (1 , 1 , 0.7 , 0 , 0) with an optimal value z^{LP}=30.3

However,

x^* = (0 , 0 , 1 , 1 , 1) with an optimal value z^*=25.

Two things should be noted on this small but illustrative instance. First, the gap between z^{LP} and z^* can be substantial, 21.2% in this particular case. Second, we should also notice how different x^* is from x^{LP}, which gives us the idea that it might not always be easy to rapidly reach a good solution from x^{LP}.

Knapsack Nomenclature

Here is a review of the nomenclature we will use for the rest of this paper:

- z: This is the objective value for a given solution.
- z^*: This is the binary optimum objective value problem.
- z^{LP}: This is the relaxed LP optimum objective value of the problem.
- x_j: This is the decision variable where we keep the "packed" or "not packed" information for the j^{th} item. If the j^{th} item is packed then this variable is equal to "1", if it is not packed then x_j is equal to "0."

- x: This is a binary solution of the problem.
- x^*: This is the binary optimum solution.
- x^{LP}: This is the relaxed LP solution.
- c_j: This is the objective coefficient associated with the j^{th} item.
- a_{ij}: This is the constraint coefficient associated with each x_j and dimension i combination.
- Dimension: Constraints of the 0-1 MKP are also referred to as its dimensions.
- Fractional variable: A decision variable which is greater than "0" but less than "1" in the LP solution of the 0-1 MKP.

0-1 Multi-Dimensional Knapsack Problem

Several names can be found in the literature for the 0-1 MKP problem (Fréville, 2004): Multi-constraint 0-1 knapsack problem, m-dimensional knapsack problem, multi-dimensional knapsack problem. We will use "0-1 multi-dimensional knapsack problem" as we think it better suits the problem. We will abbreviate it as 0-1 MKP for the sake of simplicity.

The 0-1 MKP problem is formally stated as in (1). Without loss of generality we will assume the following to avoid trivial problems:

- $c_j > 0$ for all j
- $a_{ij} \leq c_i$ for all i and j
- $b_i < \sum_{j=1}^{n} a_{ij}$ for all i

In respective order, these assumptions make sure that:

- Each item is worth packing into the knapsack
- Each item fits all the knapsack dimensions

- None of the dimension constraints are redundant

Also, we should note here, that in past literature it is possible to see papers which assume a_{ij} and c_j are integers. For the purposes of our research we did not need to limit ourselves with this assumption, hence we avoid it.

Chu and Beasley (1998) note that the 0-1 MKP can also be regarded as a general statement of any zero-one integer programming problem with non-negative coefficients. Hanafi and Fréville (1998) note that the 0-1 MKPs are used as a benchmark for the general integer programming algorithms.

Fréville (2004) and Fréville and Hanafi (2005) provide an extensive review of the 0-1 MKP. The earlier work focuses on the methodologies while the latter focuses on the computational aspects. They can be considered complementary to each other.

CHAPTER THREE: REVIEW OF THE 0-1 MKP LITERATURE AND RELEVANT TOPICS

NP-Hard and NP-Complete Problems

Since we have used the term "NP-Hard" a few times already, it is timely to review it briefly. In order to explain NP-Hard we first need to visit the theory of NP-Completeness.

NP-Completeness is designed to cover "decision" problems, in other words, problems which can only have "yes" or "no" answers. NP-Complete decision problems are very hard to solve. Proving that a decision problem is NP-Complete is equivalent to declaring that the worst case instance of the problem is "probably" intractable. We use the word "probably" due to two facts. So far there has not been an NP-Complete problem which was proved to be always solvable in polynomial time; however, so far there has also not been an NP-Complete problem which was proved to be intractable. If either one of these two proofs is completed in the future, then that proof will also be true for the rest of the NP-Complete problems since NP-Complete problems can be reduced to each other in polynomial time (Garey and Johnson, 1979).

Optimization counterparts of the NP-Complete decision problems are categorized as NP-Hard problems (Garey and Johnson, 1979). For instance, the decision problem version of the traveling salesman problem is NP-Complete and its optimization counterpart is NP-Hard.

The 0-1 MKP is a strongly NP-Hard problem and its decision counterpart is an NP-Complete problem. As a result, whatever heuristic or algorithm that might be invented, the likelihood that it will solve the 0-1 MKP in polynomial time is extremely low. This is also one reason why our research focuses on heuristics and not algorithms.

Comparison of Heuristics and Algorithms

There are several good methods for solving polynomial time problems. For instance, the simplex method (Dantzig, 1957) is a very popular algorithm for solving linear programming problems. However, we should note here that simplex method is an exponential time algorithm (Klee & Minty, 1972). Karmarkar's algorithm (1984) and many subsequent variants solve linear programming problems in polynomial time.

As we previously discussed, it is not always possible to solve NP-Hard problems to optimality in a reasonable time; therefore alternative techniques have been developed to find sub-optimal solutions for these problems. Heuristics constitute a substantial part of these alternative techniques. Heuristics have two major differences from algorithms:

- They cannot prove optimality.
- They cannot indicate how far they are from to the optimal solution.

Many of the works we will cite are based on heuristics and meta-heuristics. Meta-heuristics are in general referred as a set of rules or guidelines to coordinate underlying heuristics. Formally, meta-heuristics are heuristics too, since they do not guarantee optimality. Despite the two major disadvantages listed above, heuristics are already very popular. Here are a few major meta-heuristics:

- Genetic algorithms (Goldberg, 1989)
- Simulated annealing (Laarhoven & Aarts, 1987)
- Tabu search (Glover & Laguna, 1997)
- Scatter search (Laguna & Marti, 2003)

We can refer the reader to Osman and Kelly (1996) for a brief but good review on meta-heuristics. Later in the document, we will see how some researchers previously combined algorithms and heuristics to get the better solution out of the two approaches.

Heuristic Approaches

Weingartner and Ness (1967) work on the bi-dimensional 0-1 MKP, which is treated as somewhat a special version of the 0-1 MKP. There are several papers written on the bi-dimensional 0-1 MKP. The authors use a dynamic programming approach and find approximate solutions for the problem. They solve two instances, one with 28 variables and another with 105 variables.

Senju and Toyoda (1968) first calculate effective gradients for all the variables within their heuristic and order the variables in ascending order according to the effective gradient.

$$\frac{c_j}{\sum_{i=1}^{m} a_{ij} \sum_{j=1}^{n} a_{ij} - b_i} \tag{3}$$

Then they start from the variable with the highest effective gradient and set it to "1". They continue this until they reach a variable which they cannot set to "1" due to a constraint breach. In that case they skip to the next variable and try to set the next variable to "1". This goes on until either all the constraints are filled up or all the variables are tried. As a part of their research they solve one self invented problem with 60 variables and 30 constraints. Their research is similar to Dantzig's (1957) work in the sense that Dantzig proposes the ratio shown below to order the variables but his proposal is for the uni-dimensional 0-1 knapsack problem, in other words the single dimensional 0-1 MKP.

$$\frac{c_j}{a_j} \tag{4}$$

Dantzig proposes the simplex method as an upper bound methodology for the uni-dimensional 0-1 knapsack problem. He also outlines that the ratio above can help find relaxed LP solution of the uni-dimensional 0-1 knapsack problem without using the simplex method. His upper bound proposal is valid for the 0-1 MKP; the relaxed LP objective value of a 0-1 MKP serves as an upper bound for the binary objective value. However so far there is no known method to solve the relaxed LP version of the 0-1 MKP without using the simplex method.

Kochenberger, McCarl and Wyman (1974), present a heuristic for the "general MKP" problems. Variables are limited to being integers instead of being limited to binary values, therefore their method is usable over the 0-1 MKP as well. Differently from previous works, they start from an infeasible solution, while other well known heuristics start from a feasible solution where all variables are set to zero. They test their heuristic on a set of different problem types, with 50 variables at most. They find either optimal solutions or near optimal solutions for these problems.

Toyoda (1975) is one of the first researchers to develop an approximation heuristic for the 0-1 MKP. Although his paper's title is "a simplified algorithm for obtaining approximate solutions to zero-one programming problems", the problem he solves in this paper is actually the 0-1 MKP. His heuristic does not involve any kind of enumeration, instead he sets all the variables to zeroes and converts them to ones according to a preference mechanism. He solves problems with 10-30 variables and 10-20 constraints.

Sahni (1975) reviews several knapsack heuristics which yield fast results and guarantee a certain minimal closeness to the optimal solution value. The author generates several random problems to test the heuristics. Balas and Martin (1980) develop a new heuristic called "pivot and complement." The basic idea is to solve the linear problem first and then putting all slacks into the basis at minimal cost. Then they start a local search to improve their solution. The authors solve several integer problems; however they do not attempt to solve any 0-1 MKPs. Later, Aboudi and Jörnsten (1994) improve Balas and Martin's "pivot and complement" idea by integrating "tabu search" ideas into the heuristic and they solve 0-1 MKPs. We will mention this procedure further on.

Balas and Zemel (1980) propose solving the uni-dimensional 0-1 knapsack problem by focusing on a small part of the problem which they call the core. Although they do not know the exact size of the core when the heuristic starts running, they still get good results by approximating the size of the core using an LP relaxation. The authors solve only the uni-dimensional 0-1 knapsack problem, they do not attempt to solve the 0-1 MKP. The core concept is an important topic in knapsack research. We should note that the core concept was also our main inspiration behind the sliding concept which we will introduce later on.

Magazine and Oguz (1984), present a special heuristic for the 0-1 MKP. They start with a Lagrangean relaxation and continue with a special heuristic after that. They create 75 random problems with constraints varying between 20 and 1,000 and variables varying between 20 and 1,000. They evaluate their procedure against the heuristics by Kochenberger et al. (1974) and Senju and Toyoda (1968). Kochenberger et al.'s heuristic performs the worst in terms of the time complexity. However, the Kochenberger et al. procedure gives better results. Magazine and

Oguz favor their method over the other two, since it is faster and the best solutions they get are not far behind the Kochenberger et al. solutions.

Pirkul (1987) proposes a new heuristic procedure called MKHEUR to solve the 0-1 MKP. He uses a surrogate constraint methodology to decide which variables to set to "1". We will explain this method in detail later on and we will present an improvement to MKHEUR as well. He tests his heuristic on 230 self invented problems against the Balas and Martin (1980) heuristic. On around 1/3 of the problems they get better solutions than the Balas and Martin heuristic, on around 1/3 of the problems they get the same results, and on the rest Balas and Martin heuristic gets better solutions.

Volgenant and Zoon (1990), improve Magazine and Oguz's heuristic for the 0-1 MKP. Instead of one multiplier they compute more than one multiplier at a time. They also add a procedure which sharpens the upper bound. Finally they test their heuristic against the Magazine and Oguz heuristic. The new heuristic is a bit slower than the original, but it reaches better solutions in most of test problems they solve. Their test problems are randomly generated ranging from 25 to 200 in terms of variables and from 25 to 100 in terms of dimensions.

Fréville and Plateau (1993, 1996), develop a heuristic specifically for the bi-dimensional knapsack problem. Their heuristic can solve problems up to 750 variables to optimality.

Lokketangen, Jornsten and Storoy (1994) also improve the pivot and complement heuristic by Balas and Martin (1980). In the original heuristic by Balas and Martin, there exists a "search" part where the moves are made according to a strict priority list, namely Type1, Type2 and Type 3 moves. However in the renewed heuristic the search part is conducted by a tabu

heuristic instead of a priority list. They run their heuristic on 57 classical problems solved by Drexl (1988) and reach optimal solution in 38 of the problems.

Fréville and Plateau (1994), develop a preprocessing procedure for the 0-1 MKP. Their heuristic provides sharper upper and lower bounds for the test problems they solve. The test problems they use range from 2 to 30 in terms of dimensions and 15 to 105 in terms of variables.

Hanafi, Fréville and Abdellaoui (1996) compare two different heuristic approaches for the 0-1 MKP. Within the first approach they compare genetic algorithms, threshold accepting algorithms, tabu search, noising methods and a method called AGNES. They examine 54 of the 57 problems from Drexl (1988). AGNES finds the optimal solution in 52 of these 54 problems. Secondly, they implement several published heuristics, Glover and Kochenberger's (1996) tabu search methodology finds the optimal solution in all 54.

Hill and Reilly (2000), focus on the effects of the coefficient correlation structure. Some of the authors we cited so far already discuss the negative effects of the coefficient correlation structure (Balas and Zemel, 1980). Hill and Reilly choose the bi-dimensional 0-1 knapsack problem to test the effects of correlation. Their empirical results show that under both Pearson and Spearman correlation factors the correlation of coefficients undermines the two solution procedures they investigate, Toyoda (1975) and CPLEX. They also show that if a correlated coefficient structure is complemented with low slackness on the constraints the procedure performs at its worst. Reilly (IN PRESS), provide the most thorough review of the implicit correlation induction techniques for four classical optimization problems. He also quantifies the correlations induced by these techniques and particularly emphasizes that in many synthetically generated problem sets correlations do no vary enough, hence it is not possible to see the effects

of the heuristics and algorithms over different correlation levels even though it is repeatedly shown in the literature that correlation levels have significant effect over the procedure performance.

Bertsimas and Demir (2002), develop an approximate dynamic programming approach for the 0-1 MKP. They compare their heuristic with the results of Chu and Beasley (1998). They report that their method's time performance is at least one order of magnitude faster than the Chu and Beasley method; however, their results are not as close to the optimum as the Chu and Beasley results.

Osorio, Glover and Hammer (2002), develop surrogate constraints to solve the 0-1 MKP problem. They use the problems from the OR-Library by Beasley (1990). First they solve 54 of the problems solved by Drexl (1988) and then they solve the 270 solved by Chu and Beasley (1998). For the first 54 problems they reach optimum solutions in less than 3 seconds. For the 270 problems, on average they improve the Chu and Beasley's results on these problems too.

Akcay, Li and Xu (2007) develop a greedy heuristic for the general MKP problem. However, their heuristic also proves itself worthy in the 0-1 MKP. On small sized 0-1 MKP problems they reach very good results in a very short time with problems up to 50-100 variables and to 10-200 constraints. However as the number of variables and constraints increase they lose their dominance. In general all 0-1 MKP heuristics get worse when the number of constraints increases, their deterioration is less affected by the number of variables, but Akcay's heuristic is more prone to an increasing number of constraints than other compared heuristics, particularly against Pirkul's (1987) MKHEUR.

Volgenant and Zwiers (2007) use partial enumeration to solve the 0-1 MKPs. So instead of enumerating all the variables, they enumerate only some of the variables. The number of variables they decide to enumerate is a function of the time they are willing to spare for the solution. The problems they solve have between 100-500 variables and 5-30 constraints. They solve some problems from the OR library by Beasley (1996). They dominate Chu and Beasley's (1998) results in all these problems.

Cho, Hill, Moore and Reilly (2008) exploit the empirical knowledge of constraint slackness structure on bi-dimensional 0-1 MKPs to effectively solve them. First, they pre-process the problems and evaluate their slackness structure. Then within three well known heuristics they decide which heuristic is better for that structure. Their new heuristic works better on their problem set compared to old methods. Then they proceed to test this new approach on the Chu and Beasley (1998) test problems. Again their approach performs better against the heuristics they used. Finally they create an alternative version of the Chu and Beasley problem set in which the correlation structure and coefficient distribution also plays an important role. They contend that this new problem set is a better problem set for the 0-1 MKP. Their new approach performs the best on this problem set as well.

Meta-Heuristic Approaches

Tabu Search

Dammeyer and Voss (1993) use tabu search to solve the 0-1 MKP. They test their heuristic on the 57 problems from Drexl (1988). They find the optimal solution for 44 of the problems. Aboudi and Jörnsten (1994), also use tabu search to solve the 0-1 MKP problem. They

use the pivot and complement heuristic as a subroutine to their tabu search procedure. They use the 57 problems Drexl (1988) used as a problem set for his heuristic. In 49 (86%) of the instances they report finding the optimal solution. For the rest of the problems they report a maximum gap of 1.04% from the optimal solution.

Glover and Kochenberger (1996) develop a tabu search methodology where the search is allowed to go into the infeasible area when necessary and oscillate between the feasible and infeasible areas. Their method finds the optimal value in all 57 of the problems solved by Drexl (1988). This paper is the first of its kind in terms of solving all 57 problems to optimality. Hanafi and Fréville (1998), also develop a tabu search approach to solve the 0-1 MKP. Their heuristic is based on strategic oscillation and surrogate constraint information. Initially they form surrogate constraints and therefore reduce the problem size. Then they configure their heuristic to make oscillations on the border of the solution space. Their procedure goes back and forth between infeasibility and feasibility until it decides to jump to another location in solution space. Their heuristic reaches optimality in 54 of the problems they solve and improve the solutions in 5 of the 7 remaining problems they solve. The problems' variable count ranges from 100 to 500 and the number of constraints is between 15 and 25. They use 54 problems which Drexl (1988) also solved and reach optimality on all of those, and then they solve 7 hard problems from Glover and Kochenberger (1996) on which the optimal solution is not known. Another paper which also uses strategic oscillation technique is due to Lokketangen and Glover (1998).

Vasquez and Hao (2001a), solve the 0-1 MKP with a hybrid approach. They combine linear programming and tabu search to solve the problem. The authors first solve a relaxed version of the 0-1 MKP problem with the simplex algorithm. In the second phase they use a tabu

heuristic to improve their solution. They find the optimum solution on 56 problems previously solved by Drexl (1988). Next, they compare their heuristic with Chu and Beasley's (1998) heuristic, Vasquez and Hao's heuristic gives better solutions. Finally they solve the 11 problems (Glover & Kochenberger, 1996) published at http://hces.bus.olemiss.edu/tools.html . These problems have variables up to 2,500 and constraints up to 100. Vasquez and Hao also dominates the best known solutions on this site. These problems are some of the largest problem sizes one can find in the current literature.

Vasquez and Vimont (2005) improve Vasquez and Hao's (2001a) work and introduce more variable fixing into the methodology. This helps them improve the results. However their new methodology has a downside, it does not always create solutions where future searching is possible. Therefore good solutions are not always guaranteed.

Genetic Algorithms

Thiel and Voss (1994), present their experiences on implementing genetic algorithms (GA) on the 0-1 MKP. They conclude that simple implementations of GA's on their own are not powerful enough to get good solutions on large 0-1 MKPs. They believe using ideas from tabu search or other heuristics could improve GA heuristics. They run a hybrid heuristic mixing tabu search with GA on the 57 problems solved by Drexl (1988) and empirically support their conclusion that basic GA implementations are not enough for the 0-1 MKP. However, four years later, Chu and Beasley (1998) get very good results using genetic algorithm. Chu and Beasley, develop one of the first successful genetic algorithm approaches to the 0-1 MKP. They solve 55 of the 57 classical problems solved by Drexl (1988). They reach the optimal solution in all 55

instances. They also solve 270 randomly generated test problems with variables up to 100-500 and constraints 5-30. They also diversify the tightness value within their test problems. They use three different tightness values: 0.25, 0.50, and 0.75. They define tightness ("$\propto$") as follows:

$$\propto = \frac{c_i}{\sum_{j=1}^{n} w_{ij}} \tag{5}$$

They find better solutions than the CPLEX MIP solver in almost all of these 270 instances. These random problems are generated using the methodology suggested by Fréville and Plateau (1994).

Ant Colony

Leguizamon and Michalewicz (1999) develop one of the first ant colony optimization methods for the 0-1 MKP. They lay pheromone on only the selected variables (variables which are equal to "1"). They solve 31 test problems from Chu and Beasley (1998) and find the optimum value in more than half of these problems.

Fidanova (2002, 2003) reports two similar methodologies for the 0-1 MKP. He uses ant colony optimization in both of his works. In his methodology he suggests to lay pheromone on successively selected variables. He compares his work with a previous ant colony optimization method called MAX-MIN and some problems from Chu and Beasley (1998). He reports dominating all the results from the MAX-MIN heuristic.

Alaya, Solnon and Ghedira (2004) use ant colony optimization to solve the 0-1 MKP heuristically. They lay pheromone on all pairs of different objects in the new solutions. Therefore instead of only focusing on "selected" variables this methodology focuses on pairs of variables

(including not selected variables when necessary, which are equal to "0"). This methodology somewhat brings the ant colony optimization method closer to the genetic algorithms. First they test their procedure on two sets from Chu and Beasley (1998). They compare their results with Leguizamon and Michalewicz's (1999) where on some solutions they get better solutions, on some they tie, and on some they get worse solutions. Then they compare their solutions with Chu and Beasley's results, where Chu and Beasley dominates in all instances, Alaya et al. can only tie in some of the solutions. Finally, on this problem set they compare their results with Fidanova (2002) where they outperform almost all of Fidanova's results and tie on only two instances out of the 29 compared instances. Then as a third test set they solve five test problems due to Vasquez and Hao (2001a) with 5 dimensions and 500 variables, Vasquez and Hao have better results on all five instances.

Other Meta-Heuristic Approaches

Drexl (1988) presents one of the first "meta-heuristic" approaches for the 0-1 MKP. He implements a simulated annealing approach to solving the problem. He applies his methodology on 57 previously solved literature problems. In less than 1 second he either finds the optimal solution or gets 1% close to the optimal solution in the worst case. These test problems have variables ranging from 6 to 105 and dimensions ranging from 5 to 30. In later sections we will introduce a simple probabilistic method which can be used within Drexl's simulated annealing to get better results.

Moraga, DePuy and Whitehouse (2005) implement their unique meta-heuristic approach, "Meta-RaPS," to solve the 0-1 MKP. They find the optimal values in 55 of the 56 problems they

solve out of the 57 problems solved by Drexl (1988). In large problems, they outperform all other methods except Chu and Beasley's (1998) genetic algorithm. Especially, as the variable size increases Chu and Beasley's perform better than the Meta-RaPS. Meta-RaPS is a greedy based heuristic, therefore they report reaching these good results very quickly.

Algorithms

Horowitz and Sahni (1974) present one the earliest and most cited algorithms for the uni-dimensional 0-1 knapsack problem. Their initial algorithm actually solves the problem of "*generating all possible combinations of r numbers summing up to M*." They report their algorithm working faster than other algorithms in the literature. Then they modify their algorithm to solve the uni-dimensional 0-1 knapsack problem and report advances compared to previous works. Nauss (1976) improves the Horowitz and Sahni algorithm and dominates its results. Nauss's algorithm decreases the computation time by 1/3. It also works much better as the number of variables increase.

Martello and Toth (1977) improve a previous upper bound method by Dantzig (1957). Dantzig basically solves the LP relaxation of the problem to get an upper bound to the uni-dimensional 0-1 knapsack problem. Martello and Toth however create a sharper upper bound with a revised algorithm. Also they complement their upper bound approach with a new branch and bound algorithm which in the end works faster than the Horowitz and Sahni (1974) algorithm.

Shih (1979) solves the 0-1 MKP problem using a branch-bound method. Although he does not improve the solutions to previous problems in the literature, he improves the solution

time. He solves 30 problems with 5 constraints and their variables range from 30 to 90. Previous algorithms need around 109 minutes to solve some of these problems. However the Shih algorithm takes 13 minutes in the worst case. He also suggests that his algorithm can be extended for the generalized knapsack problem.

Gavish and Pirkul (1985) explore different relaxation approaches, namely "linear", "Lagrange", "surrogate" and "composite" relaxations. They also investigate the relations between these relaxations. Finally they develop a branch and bound algorithm, which starts off from a relaxation. They report that their algorithm is at least one order of magnitude faster than Shih's (1979) algorithm. Kellerer et al. (2004) cites Gavish and Pirkul's algorithm as the most efficient published algorithm.

James and Nakagawa (2005) develop a methodology where they can repeatedly solve enumerated sub-knapsack problems of 0-1 MKPs for an exact solution. Their methodology is quite memory intensive, the machine they run their algorithm on has 18GB of memory. They choose 29 problems from Chu and Beasley (1998), namely the 500 variable 5 constraint problems. They reach optimality in 24 of these problems in at worst 20.55 hours. They also use 250 variable and 5 constraint problems as a second problem set and show that they can reach optimality in less than 3 hours.

Combining Algorithms and Heuristics

In recent years a new approach for solving NP-Hard problems has evolved. It can be summarized as combining "algorithms" and "heuristics." The combination of heuristics and algorithms can be categorized as follows (Puchinger & Raidl, 2005).

- Collaborative combinations
    - Sequential execution
    - Parallel or intertwined execution
- Integrative combinations
    - Incorporating exact algorithms in metaheuristics
    - Incorporating metaheuristics in exact algorithms

A limited survey of some selected works is due to Dumitrescu and Stützle (2003). They focus on combinations of heuristics and exact algorithms in which further research is promising.

Clements et al. (1997) combine a local heuristic method with an integer programming algorithm. First they use the heuristic to create a set of good solutions, and then they use column generation techniques to improve the solution. They work on scheduling problems which arise on fiber-optic cable manufacturing. Their solution outperforms tabu working alone on the problem both in terms of speed and the quality of the solution.

Marino, Prugel-Bennett, and Glass (1999) combine genetic algorithms and linear programming to solve graph coloring problems (Jensen & Toft, 1994). Genetic algorithms perform poorly on graph coloring problems in general due to the symmetric nature of the problem. However the population based approach of this technique allows large jumps in the search space which is an upside of the procedure. Researchers use genetic algorithms as the main method here to move within the search space. They use the linear assignment method to generate an optimal permutation of colors within a cluster of nodes. Their heuristic slows down after 200 generations. They believe changing the parameters of their implementation could improve the speed of the heuristic.

Denzinger and Offermann (1999) utilize a branch and bound algorithm and a genetic algorithm in a sequence to solve a job-shop scheduling problem. Once one procedure stops working the other one starts working. They pass on variables to each other in terms of the condition of the solution when they stopped and information they collected during their search. The procedure which receives the information filters it in accordance with what it can use technically. They do this research in the framework of a new technique called Teams for Cooperative Heterogeneous Search (TECHS). This framework would allow different heuristics and algorithms to work together. Their experiments show that for the procedures they utilize within the framework the results are either equal to or better than the procedures working on their own on the same problem.

Filho and Lorena (2000) use a combination of genetic algorithms and column generation to approximately solve the graph coloring problem. In the first phase a constructive GA builds an initial pool of columns. Then solving the problem is attempted a column generation method.

Burke, Cowling and Keuthen (2001) solve the asymmetric traveling salesman problem with a combination of metaheuristics and an algorithm. The metaheuristics are used to search the solution space and once the heuristics find a good solution area the exact algorithm gets in to find the best solution in that area.

French, Robinson and Wilson (2001) use a combination of genetic-algorithms and branch and bound to solve a maximum satisfiability problem. First the genetic algorithm works to find a good solution. Once the solution is at an acceptable level then the branch and bound algorithm starts working on the problem. Their results show that on some instances the method may perform better than the B&B or the GA algorithm working alone.

Plateau, Tachat and Tolla (2002), focus on combining evolutionary algorithms and interior point methods to solve 0-1 programs. Their local search provides a set of distinct solutions in the first step. Following that, the evolutionary algorithm starts using this set to find the optimum solution. They use Chu and Beasley's (1998) problem set to evaluate their method. They report finding close to optimal solutions.

Cotta and Troya (2003) embed a branch and bound algorithm in an evolutionary algorithm. They use the B&B as an operator in the evolutionary algorithm.

Kostikas and Fragakis (2004) develop a genetic algorithm which works with a branch and bound algorithm to solve a mixed integer linear programming problem. First GA picks up the nodes for the B&B algorithm to start from and then the B&B algorithm starts improving in those nodes. In general there are several node selection methods available to the analyst; however these methods show good results only on specific problems. One of their implementations performs almost head to head with an established algorithm, namely best projection. The authors believe future research could improve their implementations performance.

Feltl and Raidl (2004) create a hybrid genetic algorithm to solve the generalized assignment problem. They initialize with a linear programming algorithm and the GA picks up when the LP ends. Their implementation tends to give better results in shorter time on large problem instances.

Reduced Cost Values

When the regular simplex method is used to solve LPs, the basic variables' reduced costs are equal to zero at the optimum solution. Non-basic variables' reduced cost values are negative.

However as we will explain in the following subsection reduced cost values can be both negative and positive for the 0-1 MKPs since we will use method called the "bounded variable simplex method." Nash and Sofer (1996) define the reduced cost values as follows under the bounded variable problem:

$$\hat{c}_j = c_j - \sum_{j=1}^{n} \lambda_i a_{ij} \tag{6}$$

When we solve the relaxed LP, for the 0-1 MKP, if a variable is equal to zero it will have a negative reduced cost value, if it is equal to one it will have a positive reduced cost value, if it is a fractional variable its reduced cost value will be zero.

While analyzing our results for Decision Variable Importance (DVI) ordering strategy, which we will explain in upcoming sections, one thing which attracted our attention was the fact that our ordering looked similar to the reduced cost values of the decision variables at the LP optimum. To further understand the effect, we performed a quick analysis over the Hill and Reilly problem set where we have enough problems to help us draw conclusions. We ordered all the decision variables in each problem according to their reduced cost in descending order. Then, we calculated the percentage of times the variables at those ranks are a "one" in the optimal solution.

Figure 1 represents 560 of the 2240 problems in the Hill and Reilly problem set. In these 560 problems tightness is 0.30 for both of the constraints. We see the percentage of times when a variable at a given rank is a "1" at the optimal solution. It is seen that first ranked variable 1 is 100%, this means for the 560 problems we analyzed the decision variable with the highest reduced cost value always turned out to be a "1" in the optimal solution.

To complement the information relayed in Figure 1, we present Figure 2 which shows the standard deviation of being a "1" in the optimal solution or not. The area where standard deviation peaks is where we see the partial variables. This area is where the split item would have been if the problem was a uni-dimensional knapsack problem. Hence, this is also the area which the "core" algorithms aim to solve. These figures are a good illustration of why core algorithms are not always successful, even variables far from the peak of the standard deviation can sometimes be a "1" in the optimal solution.

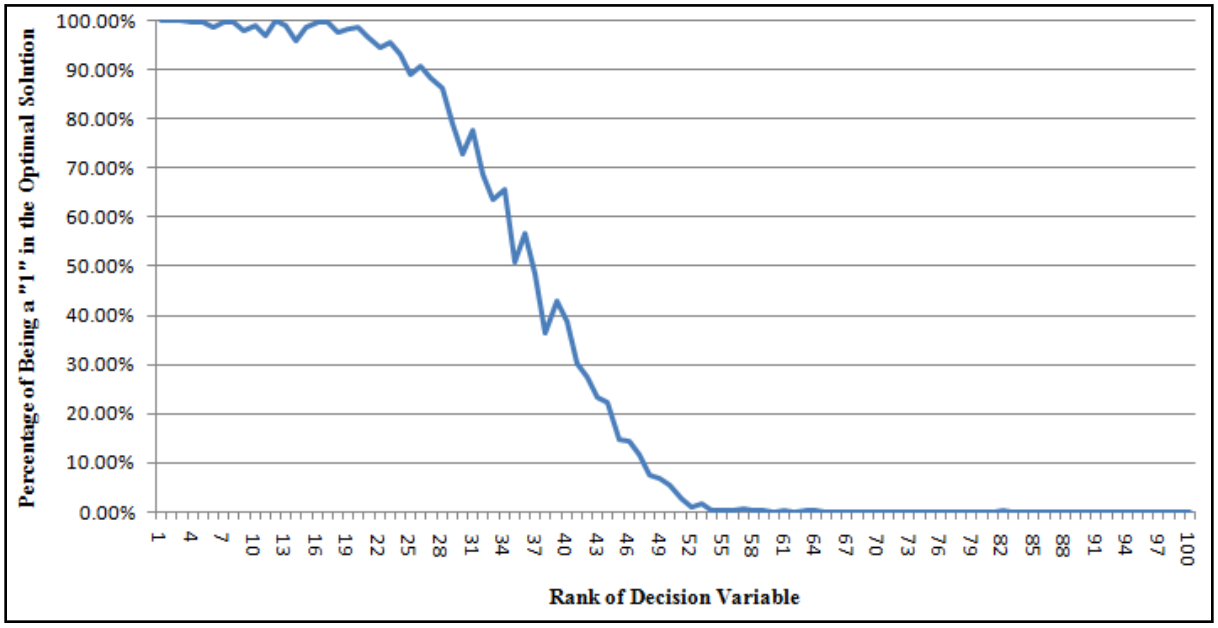

Figure 1 Ordering decision variables according to reduced costs (Tightness: 0.30)

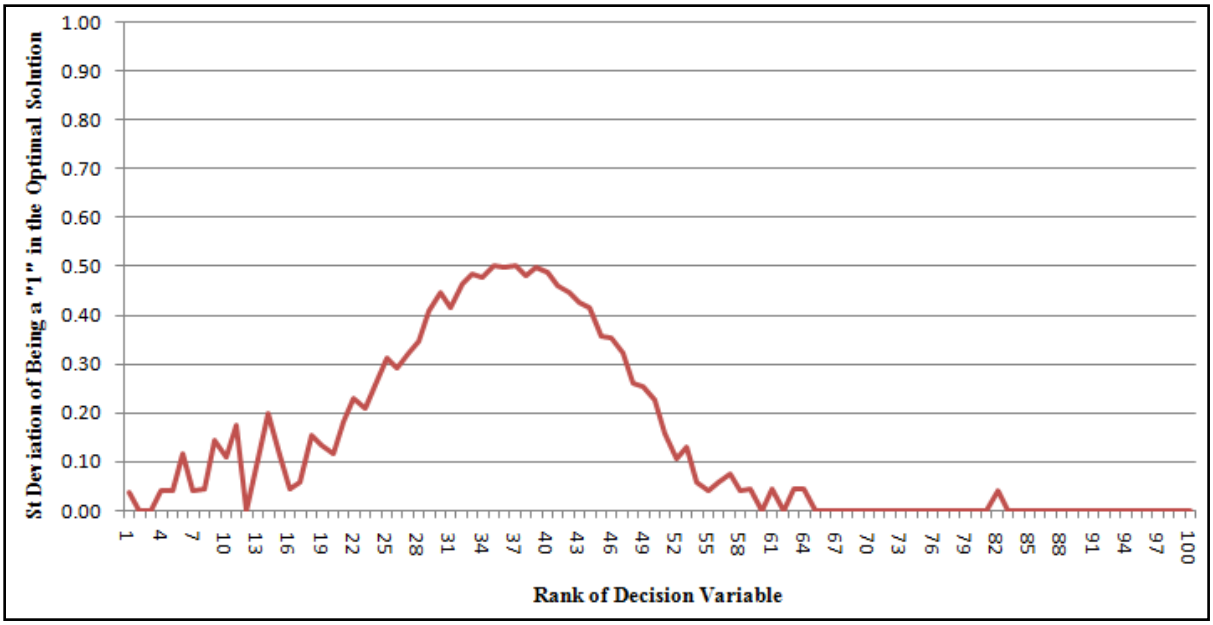

Figure 2 Ordering decision variables according to reduced costs (Tightness: 0.30)

Figure 3 and Figure 4 again show 560 problems from the Hill and Reilly problem set, but this time the constraints' tightness is 0.70. In both of the figures, just like it was in the previous figures, it is apparent that the probability of being a "one" in the optimal solution decreases as the rank according to the reduced costs at the LP optimum decreases. Again, the indecisiveness increases around the fractional values where the reduced cost is equal to "0". It should also be note that for tighter problems (tightness=0.30) there are fewer variables in the solution (fewer items packed in the knapsack) just like we would have expected.

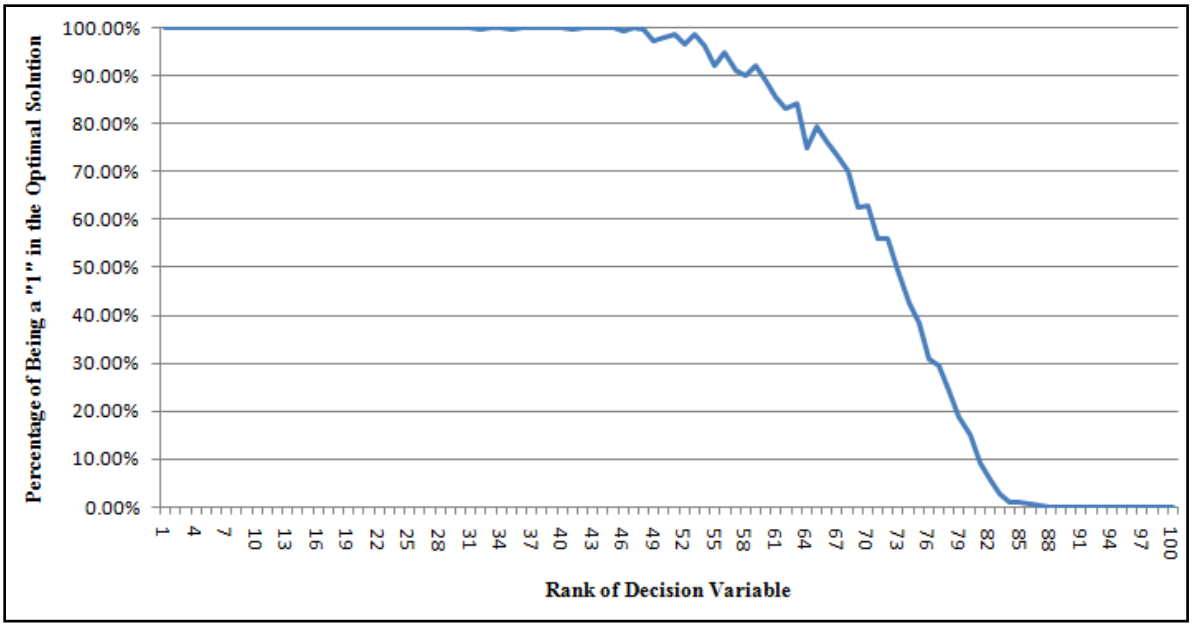

Figure 3 Ordering decision variables according to reduced costs (Tightness: 0.70)

Given our analysis, it is rather interesting that authors with the very good meta-heuristic researches on the 0-1 MKP employ random numbers to choose entering or exiting variables within their heuristics. Looking at these figures it is clear that decision variables are not equally likely to be in the optimal solution. For instance Drexl (1988) uses a random number when he chooses a variable to add to the solution at hand and again uses a uniform distribution to choose the variable to be dropped. Moraga et al. (2005) which have one of the best results in the literature using a non-complex method also use random numbers when their method picks a new variable to diversify a greedy heuristic at hand. Hence, in upcoming sections we will offer an alternative method to random selection.

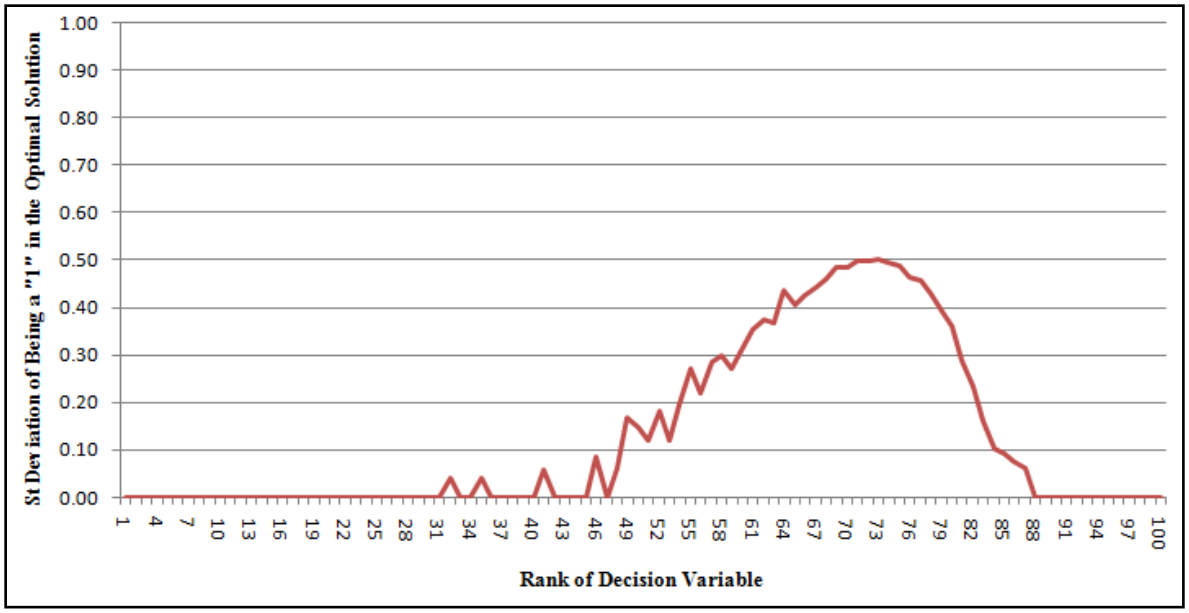

Figure 4 Ordering decision variables according to reduced costs (Tightness: 0.70)

Furthermore, it is also rather interesting that in all researches which use ordering strategies similar to a reduced cost ordering, none of them employ a tie-breaking method for the fractional decision variables. For the fractional values, Pirkul's value is always equal to "1" and the reduced cost is always equal to "0." For a 30 constraint problem this means it is potentially possible to leave 30 variables unordered within them. Thus, this potentially leads to highly different results for different implementations. Hence, in upcoming sections we will offer an ordering strategy called RCBO which employs a tie-breaking method, we will also offer an improvement to Pirkul's MKHEUR heuristic using a tie-breaking method.

Bounded Variable Simplex Method

In our research we abundantly use the relaxed version of the 0-1 MKP. We solve it with additional constraints, and we collect data from its results, etc. So it is important to note how a relaxed version of the 0-1 MKP is solved. One might think the simplex method in its usual form is applied, which would mean that, when we relax the binary constraints, we explicitly define a constraint for each and every binary variable. However that is not the case.

The following formulation shows a how a relaxed 0-1 MKP can be modeled with additional constraints explicitly defined.

$$\begin{aligned} \max z &= \sum_{j=1}^{n} c_j x_j, \\ s.t. \sum_{j=1}^{n} a_{ij} x_j &\le b_i, \quad i = 1,\ldots,m \\ x_j &\le 1, \quad j = 1,\ldots,n \\ 0 &\le x_j, \quad j = 1,\ldots,n \end{aligned} \tag{7}$$

There are several disadvantages to such a formulation. First and foremost it increases the matrix size substantially. For a 100 variable problem this way of modeling adds 100 new constraints to the problem. Considering that the simplex method is not a polynomial time algorithm and it travels on the outer vertices of the feasible polytope, increasing the number of vertices harms the solution time. Also, the increased problem size increases the memory requirements. Hence legacy software do not use the simplex method's regular form to solve relaxed binary problem. They employ a method which is known as "bounded variable simplex method" (Nash and Sofer, 1996). ILOG CPLEX and MS Excel Solver are just two sample

software which use this method to solve relaxed binary problems. In order to use this method we model the problem as follows:

$$\begin{aligned} \max z &= \sum_{j=1}^{n} c_j x_j \\ s.t. &\sum_{j=1}^{n} a_{ij} x_j \le b_i, \quad i = 1,...,m \\ & 0 \le x_j \le 1, \quad j = 1,...,n \end{aligned} \tag{8}$$

In this model the variable boundaries are modeled as implicit constraints. We do not add new constraints to the original problem; we only note that some or all variables have upper bounds. The reader shall recall that the simplex method uses the non-negativity property of the variables to stay at the borders of the feasible polytope by making sure no variable goes below zero. The bounded variable simplex method makes sure the variables do not go below zero or above the upper bound. Therefore it stays at the border of the feasible polytope of the relaxed binary problem. The obvious advantage is that the problem matrix stays as it is and the solution time does not significantly increase.

There are also side effects of this method, which are useful to our studies. When the bounded method is used the non-basic variables are either equal to zero or the upper bound. In our binary case the upper bound is "1". Since the non-basic variables can take positive values, such variables also have positive reduced cost values. The regular simplex method would have negative reduced cost values for the non-basic variables. However, in this case non basic variables equal to "0" will have negative reduced cost values and non-basic variables at the upper bound will have positive reduced cost values.

Ordering Strategies

There are several intuitive approaches which could help determine a decision variable's likelihood of being a "one" or "zero" in the 0-1 MKP. For instance, one intuitive approach would be trying to order the variables according to their coefficients in the objective function. The decision variable with the largest objective function coefficient could be assumed to be the most likely variable to be a "one" in the optimal solution. However, in reality this intuitive method fails to provide a good order of the decision variables.

Ordering strategies constitute an important part of many heuristics developed for the 0-1 MKP. Many authors use these strategies at different levels of their heuristics. Among other names, these strategies are also known as "bang for buck" ratios (Fréville, 2004) or "pseudo-utility ratios" (Moraga et al., 2005). We prefer to call these methods "ordering strategies" because that is essentially what they to do for us, order the variables. To our best knowledge the first and best known ordering strategy for the uni-dimensional 0-1 knapsack problems is due to Dantzig (1957). Dantzig's formulation is as follows:

$$\frac{c_j}{a_j} \tag{4}$$

The importance of Dantzig's ordering strategy is two-fold. First, this sometimes optimal ordering is a good starting solution for heuristic approaches. Second, it allows the researcher to find the LP relaxed solution without using any linear programming algorithm. All that is needed is to allocate the available resources to decision variables using Dantzig's ordering strategy and

the resulting solution will be the LP optimal solution. Hence it allows us to solve the relaxed LP of the uni-dimensional 0-1 knapsack problem without using the simplex method.

A logical extension of Dantzig's work on the 0-1 MKP comes from Pirkul (1987). Pirkul's main idea is to order decision variables according to the following value:

$$\frac{c_j}{\sum_{i=1}^{m} \lambda_i a_{ij}} \tag{9}$$

c_j and a_{ij}, are the objective and constraint coefficients respectively. λ_i stands for a multiplier determined by a specific procedure detailed by Pirkul. Pirkul, however, suggests and Chu and Beasley (1998) confirm replacing λ_i with the shadow price of constraint i is also possible and the ordering gives results with the same quality. On an interesting note, Chu and Beasley (1998) uses the Pirkul values as a repair operator within their genetic algorithm when they reach an infeasible solution.

Pirkul avoided the use of shadow prices since solving a linear program at the time was an expensive process hence he uses the multipliers instead of the shadow prices. Today linear program solvers are very fast and advanced, therefore it is natural to use shadow prices instead of multipliers. One shall notice Pirkul's ordering strategy is very similar to Dantzig's ordering strategy. Both strategies give out the same order when used on the uni-dimensional 0-1 knapsack problem.

The problem with Pirkul's method is it starts to decay as the problem size increases, particularly with the addition of constraints. Akcay et al. (2007), for instance, particularly notes this decay and claims that their method, PECH, would catch up with Pirkul's method and surpass

it on larger problems. We believe we were able to find the reason behind Pirkul's decay. Pirkul's method cannot differentiate between the decision variables which are fractional at the relaxed LP optimum. This comes from a simple fact that the reduced cost for a fractional decision variable in the 0-1 multi-dimensional knapsack problem has to be equal to zero. As we know the reduced cost can be calculated as follows, where y_i stands for the shadow price of the i^{th} constraint and $\hat{c}_j$ stands for the reduced cost of the j^{th} decision variable:

$$\hat{c}_j = c_j - \sum_{j=1}^{n} y_i a_{ij} \tag{10}$$

For "j" where the reduced cost is equal to zero we have the following equality:

$$c_j = \sum_{j=1}^{n} y_i a_{ij} \tag{11}$$

Thus Pirkul's value has to be equal to "one" as follows:

$$\frac{c_j}{\sum_{j=1}^{n} y_i a_{ij}} = 1 \tag{12}$$

Our claim is, the particular decay of Pirkul's heuristic method MKHEUR is due to this simple fact we have outlined above. As the number of fractional variables at the LP optimum increases the more MKHEUR decays.

There are also other well known ordering strategies in the literature. Moraga et al. (2005) lists several of those. Two of those listed strategies are as follows::

Eilon et al.'s (1971) simplest greedy rule:

$$\frac{c_j}{\sum_{j=1}^{n} \frac{a_{ij}}{b_j}} \tag{13}$$

Moraga's (2002) dynamic greedy rule:

$$\frac{c_j}{\sum_{j=1}^{n} \frac{a_{ij}}{b_j - CW_j}} \tag{14}$$

Dantzig's, Pirkul's, and Eilon et al.'s strategies can be called "simple" in the sense that we calculate the values "only once," right before we start our heuristic. However, Moraga's "dynamic greedy rule" is dynamic. CW is the used resources in constraint j. It needs to be recalculated every time after a new decision variable is converted from zero to one or vice versa.

Our research, for the most part, will focus on simple ordering strategies. However we should note there is a lot of merit to dynamic ordering strategies and future research should investigate the use of dynamic ordering strategies.

Real Life Applications

The 0-1 MKP can be used to model many real-life problems. One of the most mentioned of these problems is the allocation of capital investments (Lorie & Savage, 1955). There are also some more recent and interesting real-life applications. Beaujon, Martin & McDonald (2001) focus on a real-life 0-1 MKP. They need to decide which R&D projects to fund for the "General Motors R&D Center." Besides solving the problem with binary properties, they also investigate the effects of relaxing the binary property. Since it is possible to partially fund projects in the real

world, this is a valid and interesting investigation. A recent interesting real life problem can be seen in Vasquez and Hao (2001b). They model the daily management of a remote sensing satellite called SPOT, so that they can ease the decision making process on which photographs will be attempted the next day. They solve 7 problems which have up to 2,355 variables and 35,933 constraints. Since there are no known optimal solutions to these problems, they do not exactly know how good they performed but compared to a previous heuristic devised for the same problem they perform better on all instances. Cherbaka and Meller (2008) solve the problem of choosing which parts to produce in-house and which parts to produce outside for a sheet-metal facility.

Other Important Work and a Summary

Beasley (1990) starts hosting test problems on an e-mail server. The interested users can e-mail the server and the server e-mails them back the test problems of their interest. Later on this idea turns into a web site (Beasley, 1996).

In Table 1 we present a summary of the important works and their approaches.

Table 1. Methodology used to solve the 0-1 MKP on select works

	Exact				Approximate				
	LP Relax.	Interior Point	Enumeration	Branch & Bound	Tabu	Ant Colony	Simulated Annealing	GA	Other
Toyoda(1975)									X
Shih (1979)				X					
Magazine and Oguz (1984)									X
Gavish and Pirkul (1985)				X					
Drexl (1988)							X		
Volgenant and Zoon (1990)									X
Aboudi and Jörnsten (1994)					X				
Lokketangen et al. (1994)					X				X
Thiel and Voss (1994)					X			X	
Chu and Beasley (1998)								X	
Vasquez and Hao (2001a)	X				X				
Osorio et al. (2002)									X
Plateau, Tachat and Tolla (2002)		X						X	
Moraga et al. (2005)									X
Vasquez and Vimont (2005)	X				X				
James and Nakagawa (2005)			X						
Volgenant and Zwiers (2007)									X
Cho et al. (2008)									X

In Table 2 we show which problem sets these previous papers used to test their approaches.

Table 2. Problem sets used on select literature works

	Drexl (1988)	Chu and Beasley (1998)	Cho et al. (2008)	Glover and Kochenberger (1996)	Other
Toyoda(1975)					X
Shih (1979)					X
Magazine and Oguz (1984)					X
Gavish and Pirkul (1985)					X
Drexl (1988)	X				
Volgenant and Zoon (1990)					X
Aboudi and Jörnsten (1994)	X				
Lokketangen et al. (1994)	X				
Thiel and Voss (1994)	X				
Chu and Beasley (1998)	X	X			
Vasquez and Hao (2001a)	X	X		X	
Osorio et al. (2002)	X	X			X
Plateau, Tachat and Tolla (2002)		X			
Moraga et al. (2005)	X	X			
Vasquez and Vimont (2005)		X			
James and Nakagawa (2005)		X			
Volgenant and Zwiers (2007)		X			
Cho et al. (2008)		X	X		

In the next section we will present a brief comparison of heuristics and algorithms.

CHAPTER FOUR: METHODOLOGY

We will present several methodologies in this chapter. All of these methods are linked to the use of "reduced costs" in one way or another. First we will introduce two new ordering strategies: reduced cost based ordering (RCBO) and decision variable importance (DVI). Next, we will introduce the sliding concept and the sliding enumeration method. Sliding enumeration will use the ordering strategies we just mentioned previously. Additionally, based on the information we have accumulated through the use of reduced costs we will introduce advanced and better performing versions of two past works, Pirkul's MKHEUR (1987) and Drexl's simulated annealing (1988) methods. Finally, we will briefly discuss a local search design which we will use to improve the results of a sliding enumeration experiment based on RCBO.

Ordering Strategies

Decision Variables Importance (DVI)

In this section we will break from the traditional approaches like Pirkul's ordering strategy based on shadow prices or other strategies and we will explore a rather less intuitive but novel method. This method solves LP problems close to the relaxed solution of the 0-1 MKP. Once we sum up the "solutions" from this series of solutions, we obtain a vector in which we claim the decision variables with higher values are more likely to be a "one" in the optimal solution. Unfortunately, the order we obtain is not perfect. Some variables get relatively higher values even though they are "zero"s in the optimal solution. Still, our experiment results show that this method can aid us finding optima in many problems.

The reader would remember we have previously shown the relaxed version of the 0-1 MKP in (8). Vasquez and Hao (2001a) previously showed that the sum of the optimum solution for the relaxed 0-1 MKP is very close to the sum of the optimum solution for the binary version. In order to better explain DVI, let us consider the following 0-1 MKP instance.

$$\max z = 4x_1 + 5x_2$$

$$3x_1 + 2x_2 \leq 4$$

$$2x_1 + 3x_2 \leq 3$$

$$x_j \in \{0,1\}$$

Graphical representation of the relaxed version of the sample problem is shown below.

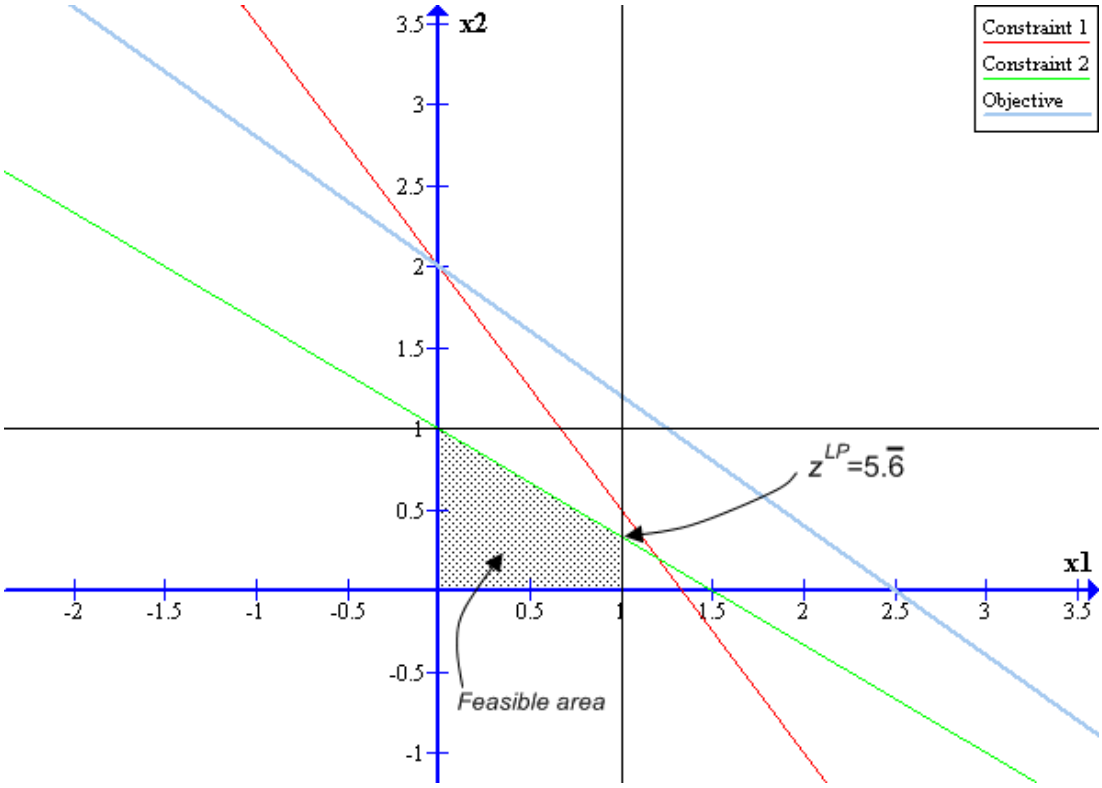

Figure 5. Graphical representation of the relaxed version of the sample problem

The relaxed LP optimum solution is as follows:

$$x^{LP} = (\ 1,\ 0.3333\ )$$

The relaxed LP objective value is as follows:

$$z^{LP} = 5.66666$$

Sum of the relaxed LP optimum solution decision variables is:

$$\sum_{j=1}^{n} x_j = 1.33333$$

Now we will collect data around the relaxed LP optimum solution by solving a series of LP problems. A sample LP problem will be as follows, it is also the center of our data collection region:

$$\max z = 4x_1 + 5x_2$$

$$3x_1 + 2x_2 \leq 4$$

$$2x_1 + 3x_2 \leq 3$$

$$x_1 + x_2 = 1.33333$$

$$0 \leq x_{all} \leq 1$$

We will change the "right hand side" of the third constraint ($x_1+x_2=1.33333$) by steps of "0.1" up and down until the total change is at most "0.5." The "0.5" value was chosen arbitrarily for this sample problem. In Figure 6, the dashed lines show the lines over which we will collect data.

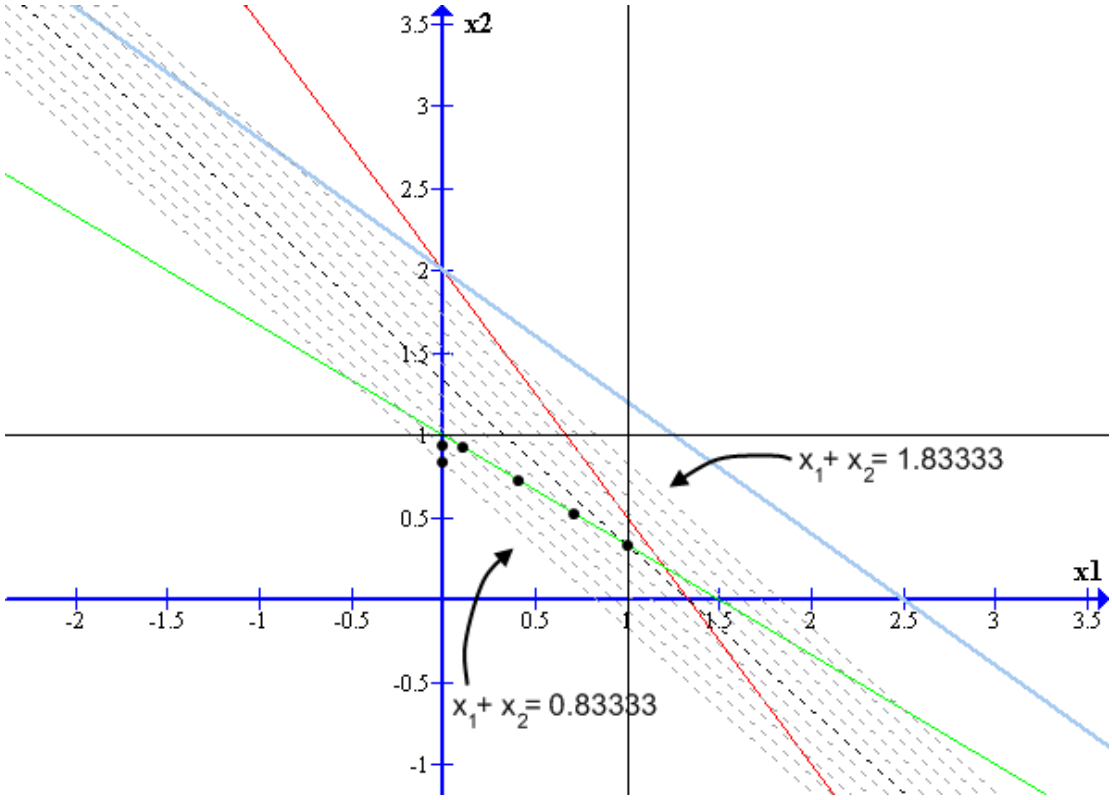

Figure 6. Collecting data using a series of LP solutions

The new constraints which we use to modify the original relaxed problem are shown by the dashed lines in Figure 6. The feasible points from which we collected data are shown by dark circles in Figure 6. Table 3 lists all the points considered for data collection including the infeasible ones.

The total of the data we collected for each of the decision variable are as follows:

- $x_1 = 2.19997$
- $x_2 = 4.10001$

x_2 has a larger value than x_1 therefore we conclude that it is more likely to be "1" in the optimum solution. In this particular instance this is a good ordering since the binary optimum solution is as follows:

- x*= (0, 1)
- z*= 5

Table 3. Data collected from the 11 points

$\sum_{x=1}^{n} x_i$	x_1	x_2	z_{LP}
1.83333		Infeasible	
1.73333		Infeasible	
1.63333		Infeasible	
1.53333		Infeasible	
1.43333		Infeasible	
1.33333	1	0.33333	5.66666
1.23333	0.69999	0.33334	5.46666
1.13333	0.39999	0.73334	5.26666
1.03333	0.09999	0.93334	5.06666
0.93333	0	0.93333	4.66665
0.83333	0	0.83333	4.16665
Total	2.19997	4.10001	

If we had equal importance values for the decision variables we would have used the objective coefficient as a tie breaker. The decision variable with a larger objective coefficient would be considered to be more likely to be a "1."

The generalization of the method presented in this section is shown in Figure 7. In the generalization φ stands for the step size we have used in the above example. ρ stands for half of the region we cover around the relaxed LP solution.

```
Load instance; Input parameters ρ, φ;
Form array DVI to hold the importance values for all the decision variables;
Solve integrity relaxed version of the instance;
Center value = Sum of the values of the decision variables in x^LP;
upper_bound = center value + ρ;
While  num_iter ≤ (2ρ/φ)+1
        rhs = upper_bound-((num_iter-1) x φ);
        Add new constraint to the instance: ∑x = rhs;
        Solve integrity relaxed version of the instance with the new constraint;
        Add the values of the decision variables in x^LP to DVI;
        Remove the new constraint from the instance;
        num_iter = num_iter+1;
End_while
Output DVI;
```

Figure 7 Pseudo-code of detecting decision variable importance

To summarize it all, DVI is a method to determine an order for the decision variables. It is not a heuristic in its own. It needs to be used within a solution methodology in order to help our optimization process. It can be used within a greedy heuristic where we try to assign the highest valued decision variable as "1" and assign "0" when it is infeasible. Or it can be used in other heuristic methods which require an ordering of the decision variables to work. We should note here since DVI has several parameters we need to do some experiments to find a good set of values for those parameters. In the following sections we will do such experiments to find good parameter values.

There are two main disadvantages of DVIM. First, it is not intuitive and not easy to explain. Thus, it is hard to re-implement. Second, it uses the simplex method several times before it gives results. As we know, the simplex method is an exponential method and heavily relying on it may not turn out to be efficient, particularly on large problems.

Reduced Cost Based Ordering (RCBO)

Reduced cost based ordering (RCBO) was invented as a result of our studies with DVI. Once we had some interim experiment results of DVI within several heuristic techniques we noticed the similarities between the reduced cost values and DVI values. As the reader would know, reduced cost values are used within the simplex method to pick the next entering variable to the solution. The decision variable with the highest reduced cost value is entered to the solution in each iteration. Hence we decided to test the use of reduced cost values as an ordering strategy as well and called it the "reduced cost based ordering".

As we have explained previously, reduced cost values within a bounded variable simplex method are can get a different range of values compared to the reduced cost values in the regular simplex method. The decision variables can get both negative and positive reduced cost values. If the variable is at its upper bound it gets a positive reduced cost value, if it is at its lower bound then it gets a negative reduced cost value.

The main benefit of RCBO over DVI is its elegance and simplicity. It is trivial to use RCBO since all the legacy optimization software in the market can report the reduced cost values following a bounded variable simplex method solution. And all these software automatically

solve binary problems' relaxed versions using the bounded variable simplex method by default as long as the user defines around the variables specifically.

A particularly important point we should note here is how we handle two decision variables which have equal reduced cost values. At that point we check the x_i^{LP} for both variables and give higher priority to the variable with higher value. Additionally, if the solution values are equal as well we compare the objective coefficients of the variables and give priority to the variable with higher objective function coefficient.

```
Load instance;
Solve integrity relaxed version of the instance;
Order variables in the ascending order of reduced cost values.
        Use solution values as the first tie-breaker
        Use objective coefficients as the second tie breaker
Output the order of the variables ;
```

Figure 8 Pseudo-code of reduced cost based ordering (RCBO)

In the next section we will introduce the sliding concept and sliding enumeration method. It is possible to use RCBO within the sliding enumeration method. In later section we will also present some experiments on RCBO's use.

The "Sliding" Concept

The sliding concept is a novel type of greedy heuristic. In general, a greedy heuristic takes an ordering of variables as input and tries to assign variables the value "1" according to the given order. If it is infeasible to assign the value "1", then the heuristic assigns "0" and proceeds

to the next variable in the given order. Sliding concept, however, instead of checking the feasibility for only the next variable, considers a number of the next variables and finds the optimum solution for those variables. From the optimal solution of those variables we extract the value of the variable with the highest value according to the given ordering strategy.

Figure 9 illustrates the sliding concept for a "ten" variable problem. In this instance we are evaluating "five" variables at a time. At each step we perform an exact algorithm to find the optimum for the evaluated variables and we fix the variable with the highest importance at each step. The variables under evaluation are shown by yellow squares and the variable we are going to fix has a green circle.

It is possible to use different exact methods within the sliding concept. Our choice was to use a simple enumeration where all possible combinations of the variables are enumerated. We call that method "sliding enumeration" and we explain it in the next sub section. However one can use another exact method such as:

- Branch and bound
- An enumeration technique designed for the problem at hand
- A legacy optimization software

As we have mentioned previously, we are going to employ a simple enumeration technique which allows us to roughly optimize up to 20 variables at a time. Using a simple enumeration in the very first experiments of the sliding concept helps us avoid unexpected complexities in the very first implementation of the sliding concept.

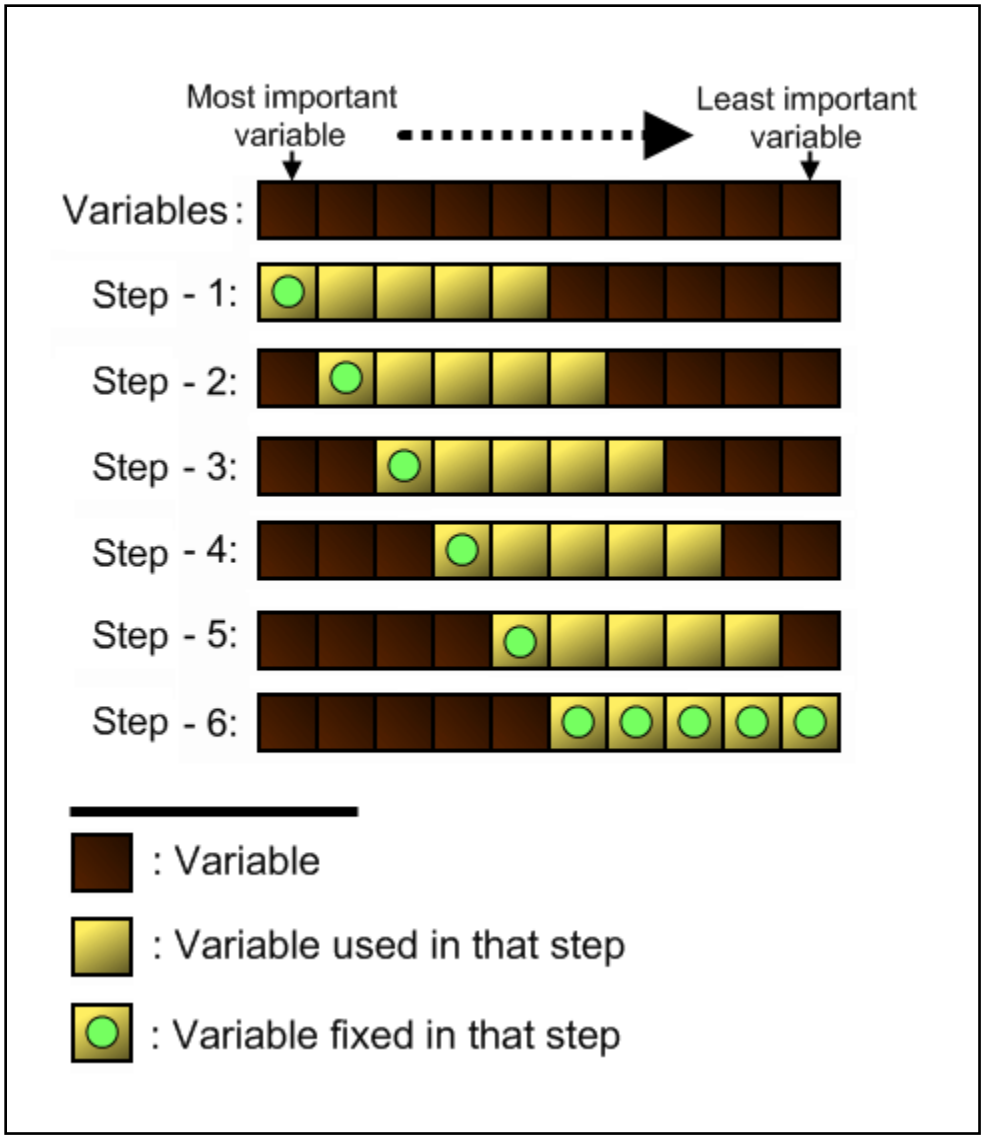

Figure 9 Sliding concept illustration for 10 variables and sliding width=5

Sliding Enumeration

In the previous sections we have introduced two methods (DVI and RCBO) which can order the decision variables. Following that we introduced the sliding concept which can exploit

the ordering methods we introduced. Therefore now we need a methodology which can be used within the sliding concept to optimize a given small problem.

We will illustrate a method in this section which we call a "sliding enumeration." It will aid us get good results using a good order. A work similar to the "sliding enumeration" is due to Reilly, Gonsalvez & Mount-Campbell (1990). They find fixed satellite service orbital allotments with a method called k-permutation algorithm. The k-permutation algorithm starts from an initial solution and re-evaluates "k" adjacent satellites using enumeration to check if they can find better solutions. Similar to our method they continuously move their evaluation space over the ordering of the satellites. However, different from our approach, they continuously re-order the satellites and start working on the new order. Also different from our approach, they increase the number of adjacent satellites evaluated up to a previously determined maximum number if they cannot find better solutions.

The following arbitrary example has its variables ordered according to the "objective coefficients". It could have been ordered by another strategy as well such as DVI or RCBO.

$$\begin{aligned} \max z \;=\; & 10\,x_1 + \;5x_2 + 4x_3 + 2x_4 + \;\; x_5 \\ & 3x_1 + 4x_2 + 5x_3 + 3x_4 + 4x_5 \quad \leq 10 \\ & 6\,x_1 + 5x_2 + 2x_3 + \;6x_4 + 7x_5 \quad \leq \;\; 8 \\ & x_j \in \{0,1\} \end{aligned}$$

In sliding enumeration we only enumerate a fixed number of variables at a time. For this problem we will enumerate only "3 variables at a time". We start from left and slide the enumeration to right, hence the name "sliding enumeration".

Figure 10 shows the first step of sliding enumeration. The 6th combination gives the best feasible result, hence the optimum for those three variables. We fix the value of the variable with the highest importance, which is x_1, as "1", x_1's value at the optimum combination.

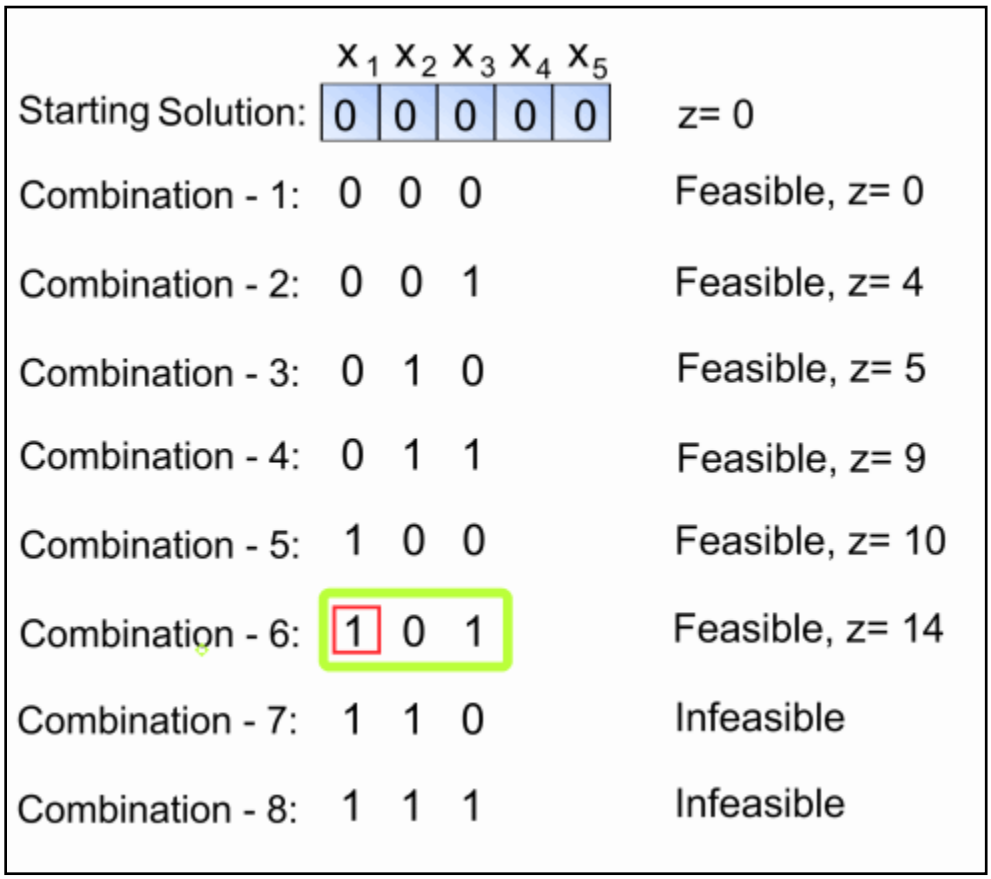

Figure 10. Sample sliding enumeration, step 1

Next, we slide the enumeration one step right in Figure 11. Enumeration starts with the 2nd variable and this time the 3rd combination gives the best feasible result. So we fix x_2 , the variable with the highest importance, to "0".

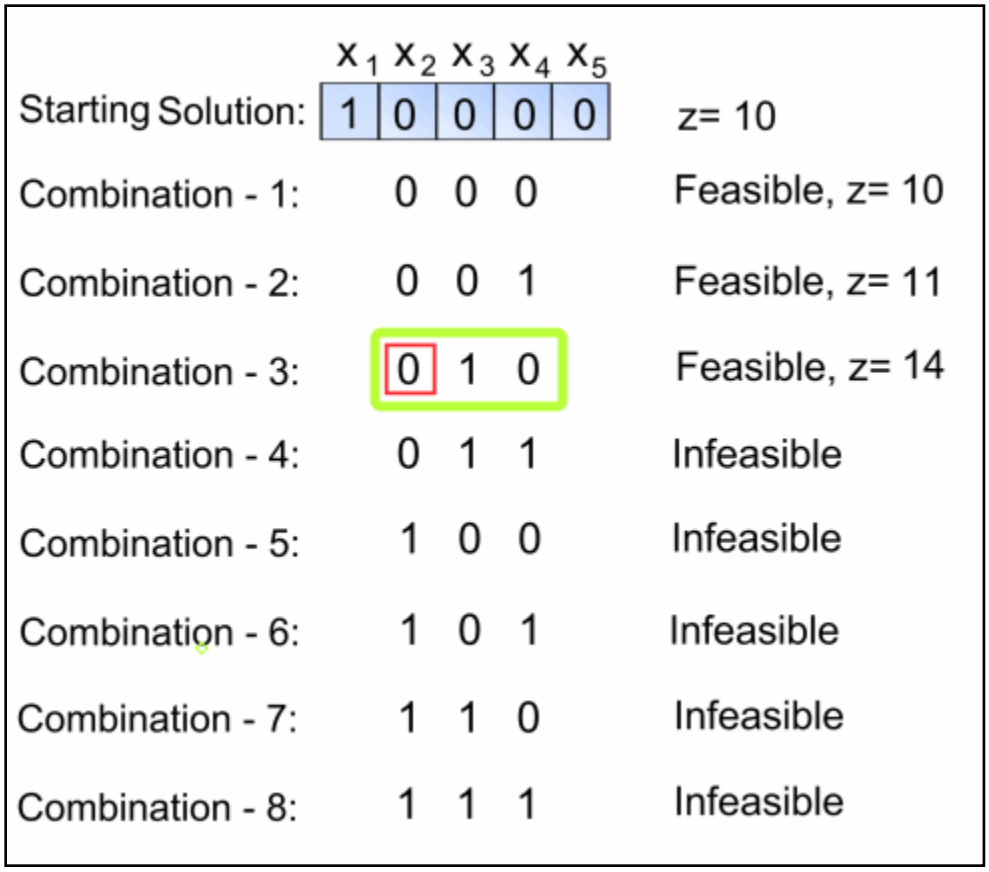

Figure 11. Sample sliding enumeration, step 2

Finally, we slide the enumeration to right for one more step. Now the reader should notice in this 3rd enumeration we reached to the end of the decision variables therefore in this enumeration we will fix all the variables. In Figure 12, combination 5 gives the best feasible result and therefore we will fix all final three decision variables according to combination 5. The solution we reached is:

$$x= (1, 0, 1, 0, 0)$$

The objective value is:

$$z=14$$

For this small instance it turns out this is the optimum solution as well, however sliding enumeration does not necessarily find optimums, it is a heuristic procedure.

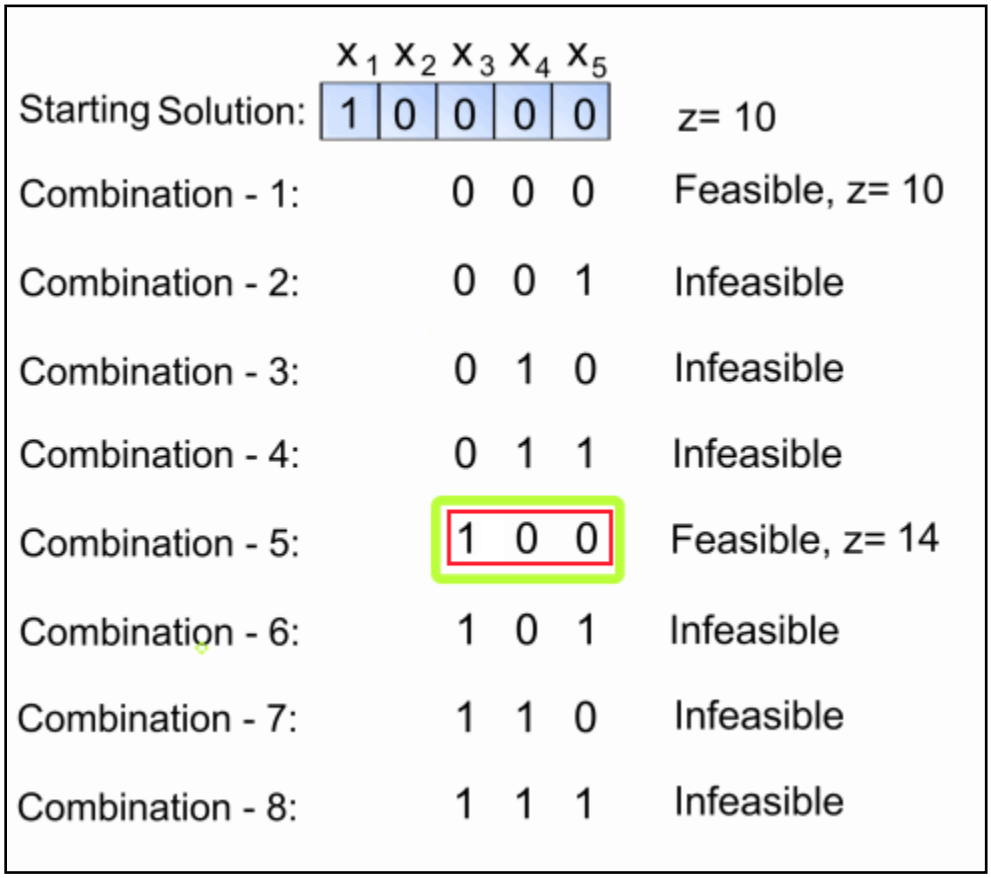

Figure 12. Sample sliding enumeration, step 3

For any given binary problem the total number combinations we will normally consider can be calculated as follows:

$$(n-\propto+1)2^{\propto} \tag{15}$$

n: Number of variables in the problem

$\propto$: Number of variables to be enumerated at one step

For instance if we want to use a $\propto$ =12, over a 100 variable problem. The number of total combinations we will calculate will be:

$$(100\text{-}12\text{+}1)2^{12} = 364{,}544$$

It is possible to think that sliding enumeration is a brute force approach to binary problems, however, if we were to enumerate the 100 variable binary solution space the total number of combinations would be:

$$2^{100} = 1.27 \text{ x } 10^{30}$$

The ratio of the sliding enumeration combinations ($\propto$=12) to the whole enumeration combinations is as follows:

$$\frac{364{,}544}{1.27 \text{ x } 10^{30}} = 2.88\, x\, 10^{-25}$$

2.88 x 10^{-25} is a very small ratio and it is far from what we could call brute force. Figure 13 shows the generalization of the sliding enumeration.

In our implementation we have deviated from the pseudo-code with a minor revision. Instead of starting from an all “zero”s (origin) solution we start from an all “one”s solution. If this first solution is optimum we do not proceed to the rest of the combinations. This small trick particularly improves efficiency in problems which are not tight, since the optimum tends to have many “one”s.

```
Load instance; Input parameters α;
enumeration_array= A 2 dimensional array to hold all binary combinations of α variables;
While  num_iter ≤ (n- α +1)
        While  num_comb ≤ 2^α
        best_comb = 0;
                While  num_var ≤ α
                solution[num_var] = enumeration_array[num_comb][ num_var];
                num_var = num_var+1;
                End_while
                If solution is feasible
                        If obj value ≥ best_obj_value
                        best_obj_value = obj value;
                        best_comb = num_comb;
                        End_if
                End_if
        num_comb = num_comb +1;
        End_while
                If num_iter < (n- α +1)
                solution[num_iter] = enumeration_array[best_comb][1];
                End_if
                Else
                        While  final_iter ≤ n
                        solution[num_iter] = enumeration_array[best_comb][ final_iter];
                        final_iter = final_iter+1;
                        End_while
                End_else
        num_iter = num_iter+1;
End_while
Output solution;
```

Figure 13. Pseudo-code of sliding enumeration

Discrete Right Triangular Distribution

In this section we will introduce a probabilistic approach which can be used within heuristics relying on the random choice of entering and exiting variables. Greedy heuristics try to assign the value "1" according to a given ordering strategy in 0-1 MKPs. However the problem with greedy heuristics is that they can lead to local optima and get stuck there. Therefore meta-

heuristics are developed to make sure the heuristic can be pulled out of the local optima. Meta-RaPS (Moraga et al, 2005) is a good example of a meta-heuristic which can improve greedy heuristics.

Many times this improvement is done using a random number to pick an entering or exiting decision variable at a given current solution. The problem with this approach is, more often than not, all decision variables are treated equally when we choose the entering or the exiting variable. In other words all variables have the same chance to enter or leave the solution. Based on our analysis of the reduced costs which we shared previously, we suggest that the decision variables should not be treated equally.

We propose we can use a discrete right triangular distribution (Rider, 1963) based on RCBO to model the likelihood of a variable being in the optimum solution or not. RCBO was explained in detail previously so we proceed to the explanation of the right triangular distribution and its usage.

Figure 14 shows the discrete right triangular distribution's probability masses. The first variable has the highest probability and the last variable has the lowest probability. The probability decreases "equally" from variable to variable.

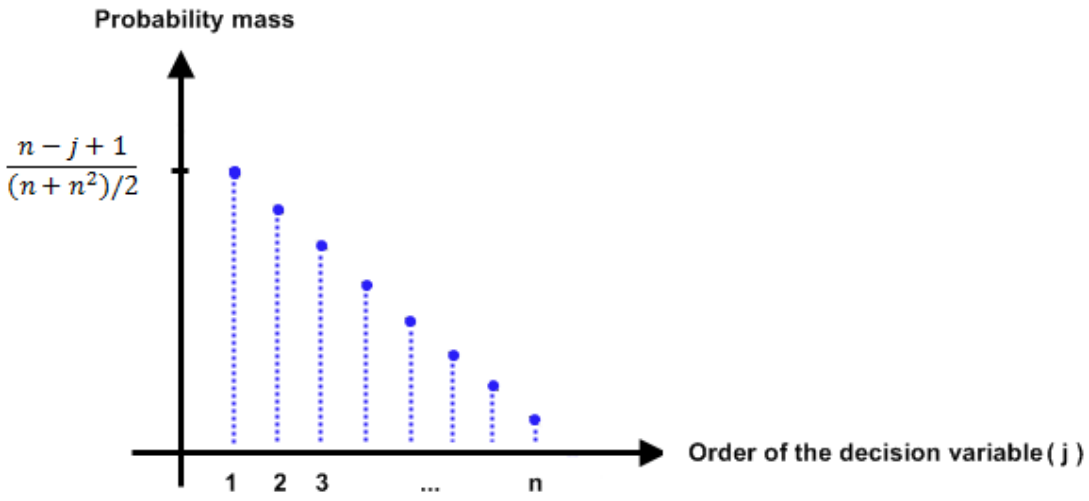

Figure 14 Probability mass graph for the right triangular distribution

The cumulative distribution function of the distribution is as follows:

$$\frac{j\,(2n-j+1)}{n+n^2} \tag{16}$$

We will use this method within Drexl's (1988) simulated annealing meta-heuristic. He uses random numbers to pick entering and exiting decision variables. Instead of random numbers we will use the distribution defined above to pick the entering variables. For picking the exiting variables we will use the opposite of this distribution, hence the variable which is most likely to enter the variable will be the variable least likely to exit a solution.

Local Search

One of the things we have noticed during our experiments was that a good heuristic such as sliding enumeration produces good solutions which are either optimal or very close to the

optimal solution. Usually to reach the optimal all we needed to do would be to drop a few items from the knapsack and add a few items from the left out ones.

Therefore we are going to use a drop add methodology to search for better solutions in the close neighborhood. For instance if we start a local search to drop 1 item and add 1 item (Drop 1 add 1) that means we will check all the neighbor solutions which can be reached by changing one of the ones in the initial solution to zero and changing one of the zeroes to one. In our experiments we will search up to the "drop 4 add 4" area. If during our search we find a better solution we will go back and start a brand new search from that solution. Figure 15 and Figure 16 show the design of our local search.

In the following sections whenever we refer to "local search" we refer to the following pseudo-code. The reader would notice this pseudo-code does not take any input parameters, hence this method does not get modified from problem to problem.

```
Load input solution; best_obj_value=objective value of the input solution; best solution=input solution; improvement=false;
While(improvement=false)
    If(Improvement=false)
        Search Drop0Add1 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop1Add1 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop1Add2 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop1Add3 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop1Add4 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop2Add1 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop2Add2 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop2Add3 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
    If(Improvement=false)
        Search Drop2Add4 around best_solution;
        If obj_value ≥ best_obj_value
        best_solution=DropAdd_solution, improvement=true;
        End_if
    End_if
```

Figure 15. Pseudocode of the local search logic part 1 of 2

```
        If(Improvement=false)
                Search Drop3Add1 around   best_solution;
                If obj_value ≥ best_obj_value
               best_solution=DropAdd_solution, improvement=true;
               End_if
        End_if
        If(Improvement=false)
                Search Drop3Add2 around   best_solution;
                If obj_value ≥ best_obj_value
               best_solution=DropAdd_solution, improvement=true;
               End_if
        End_if
        If(Improvement=false)
                Search Drop3Add3 around   best_solution;
                If obj_value ≥ best_obj_value
               best_solution=DropAdd_solution, improvement=true;
               End_if
        End_if
        If(Improvement=false)
                Search Drop3And4 around   best_solution;
                If obj_value ≥ best_obj_value
               best_solution=DropAdd_solution, improvement=true;
               End_if
        End_if
        If(Improvement=false)
                Search Drop4Add1 around   best_solution;
                If obj_value ≥ best_obj_value
               best_solution=DropAdd_solution, improvement=true;
               End_if
        End_if
         If(Improvement=false)
                 Search Drop4Add2 around   best_solution;
                 If obj_value ≥ best_obj_value
                 best_solution=DropAdd_solution, improvement=true;
                 End_if
         End_if
         If(Improvement=false)
                 Search Drop4Add3 around   best_solution;
                 If obj_value ≥ best_obj_value
                 best_solution=DropAdd_solution, improvement=true;
                 End_if
         End_if
        If(Improvement=false)
                Search Drop4Add4 around   best_solution;
                If obj_value ≥ best_obj_value
               best_solution=DropAdd_solution, improvement=true;
               End_if
        End_if
End_while
Output best_solution
```

Figure 16 Pseudocode of the local search logic part 2 of 2

CHAPTER FIVE: RESULTS

Problem Sets

We have obtained several problem sets from the literature for this experiment. Table 4 lists the 2,847 instances we have available at hand by their sources.

Table 4. Problem sets available to our research

Problem set	# of problems	Instance with the maximum # of constraints	Instance with the maximum # of variables
Drexl (1988)	56	105x2	60x30
Chu and Beasley (1998)	270	500x30	500x30
Cho et al. (2008)	270	250x25	250x25
Glover and Kochenberger (1996)	11	2500x100	2500x100
Hill and Reilly (2000)	2240	100x2	100x2

We should note that the 325 problems from Beasley's (1996) online OR-Library include 270 problems from Chu and Beasley (1998) and 55 problems from Drexl (1988). Originally, Drexl solves 57 problems from the literature but unfortunately Beasley's OR-Library has only 55 of those. One of our committee members kindly provided one of the missing two problems (The Fleischer problem), thus we had 56 of the Drexl problems. The original 57 problems and the 56 problems we have available to our research are shown in Table 5.

Chu and Beasley's and Cho et al.'s problem sets are similar in some ways. They both have 270 problems. They both have 9 combinations of constraints and variables. And they both have 30 problems in each combination. Cho et al.'s problem set was created with more

controlled parameters compared to Chu and Beasley's problem set in order to gain a better understanding of different heuristics' strengths over different properties. The largest problems in the Chu and Beasley problem set has 500 variables and 30 constraints and the largest problem in the Cho et al. problem set has 250 variables and 25 constraints.

Table 5. List of problems in the Drexl problem set

Problem sets	Original number of problems	Number of problems available to our research
Wei Shih (Also called WEISH)	30	30
Petersen (Also called PET)	7	7
Hansen and Plateau (Also called HP)	2	2
Weingartner (Also called WEING)	8	8
Senju and Toyoda (Also called SENTO)	2	2
Fleisher (Also called FLEI)	1	1
Fréville and Plateau (Also called PB)	7	6
Total	57	56

Hill and Reilly's problem set has 2240 problems with 100 variables and 2 constraints. Even though they have over two thousand problems with the same size they have different correlation and slackness characteristics for these problems. This allows the researcher to better understand the heuristics they are experimenting.

Finally, Glover and Kochenberger problem set is the largest problem set available in the literature.

Experiment Infrastructure

In any experiment which is conducted over the computer, the following are the important parts of the infrastructure, hence this section discusses them:

- How the problems are stored
- Software used
- Main programming language

Development Environment

We have used Eclipse Ganymede as our integrated development environment. Our experiment was completely coded in Java. CPLEX 11 was used to solve LP problems. In order to solve problems back to back we have designed the database shown in Figure 17 and entered all the problems which we have presented in Table 4 into this database. Our database is a Microsoft SQL 5 Express database which is freely available for non-commercial purposes.

The database we have introduced above coupled with our Java code allows us to solve a specific subset of the problems with great ease. For instance to solve the problems from the Chu and Beasley data set with 100 variables we would introduce this query in a text file called sqlquery.txt:

"*select ProblemID from ProblemsTable where ProblemGroup='ChuAndBeasley270' and VariableQuantity='100'*"

If we only wanted to solve problems from the Chu and Beasley data set with 100 variables and 30 constraints the following query would be enough:

"*select ProblemID from ProblemsTable where ProblemGroup='ChuAndBeasley270' and VariableQuantity='100' and ConstraintQuantity='30'*"

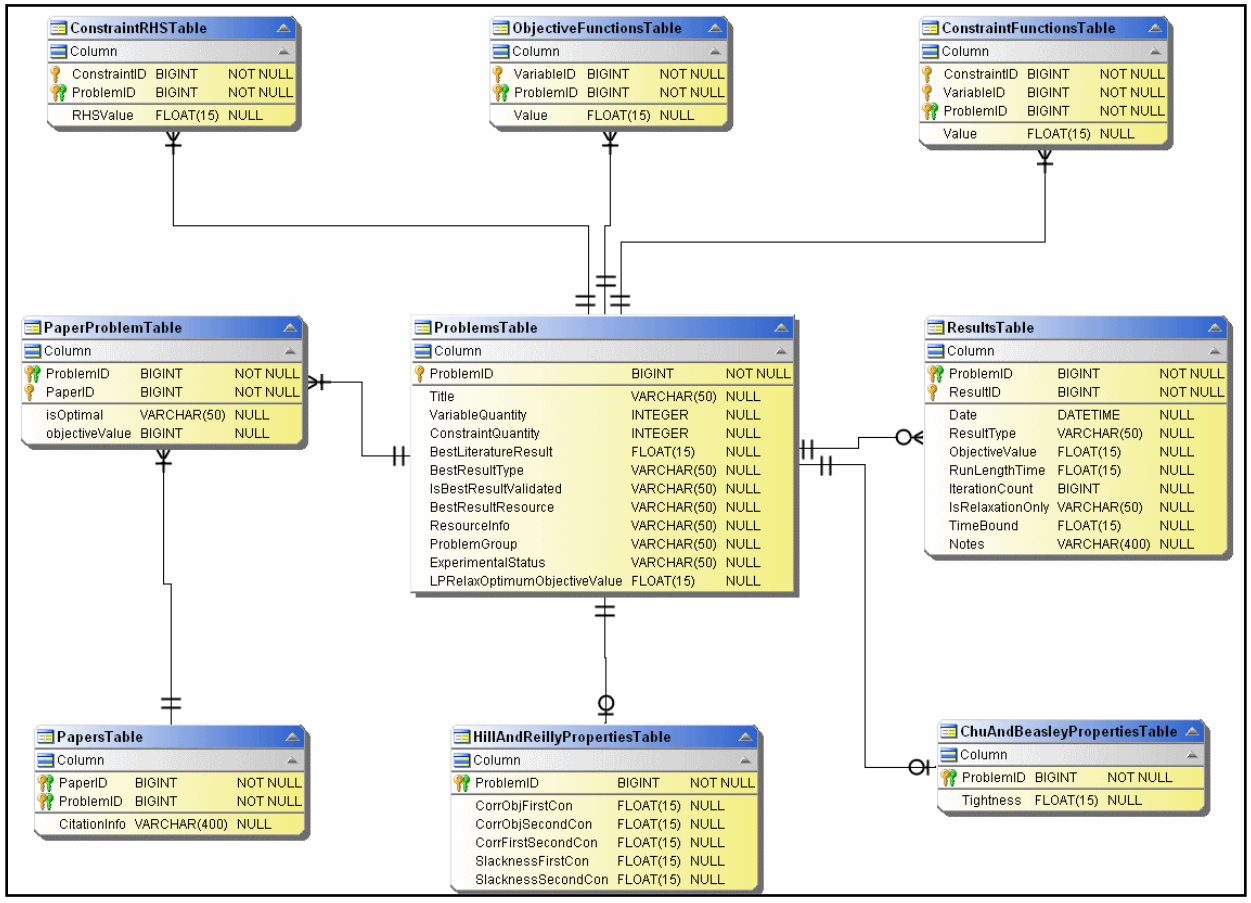

Figure 17. Entity-relationship (ER) diagram of the SQL database

The Java code does not stop until it conducts the experiment over all the problems defined by the query in "sqlquery.txt".

Here is a short description for the tables shown in Figure 17:

- ProblemsTable: This is the center of the database. Most of the tables are connected to this table. It holds a ProblemID generated by us which helps keep every problem unique. It

also holds the name of the problem, number of variables, number of constraints, best known objective value (also if the best known objective value is optimum or not), information on where we got the problems from (physically). Also a note for ourselves to remember "if we validated the best known objective value or not" is included.

- PapersTable: A table to keep citation info for all the papers mentioned in the database.
- PaperProblemTable: A table to link the PapersTable and ProblemsTable.
- ConstraintFunctionsTable: A table to keep the constraints for each problem.
- ResultsTable: This table keeps the results of our experiments.
- ConstraintRHSTable: A table to keep the right hand side (RHS) values for each constraint in each problem.
- ObjectiveFunctionsTable: A table to keep the objective functions for each problem.
- ChuAndBeasleyPropertiesTable: A table to keep some attributes specific to the 270 problems from the Chu and Beasley (1998) problem set.
- HillAndReillyPropertiesTable: A table to keep some attributes specific to Hill and Reilly (2000) problem set.

Figure 18 shows the class diagram of our development environment. Class diagrams are an essential part of the Unified Modeling Language (UML). They help us understand the relationship of classes to each other. They help us see methods, attributes within classes and some more detail about these methods and attributes like the inputs and outputs and if they are public or private.

The AkinMain class is the main() class which is used to start any experiment. AkinMain class has the capability to conduct experiments on a given array of problems. This array is pulled from a Microsoft SQL express database. It accepts arguments such as:

- The problems which will be solved
- The experiment which will be conducted
- Parameters specific to that experiment

Heuristic class is a super class to the specific heuristic classes within our system. We have a super class to keep some of the common methods which we will need for our heuristics. Some methods do not change from heuristic to heuristic, for instance “feasibility check”, “objective calculation” and some other functions are common methods. Many different heuristics use these methods. It is inefficient to re-write these methods all over again, therefore we keep these common methods in the super class and use them in the sub classes through the process called “extension” or “inheritance” which is an important feature of object oriented programming, hence Java as well.

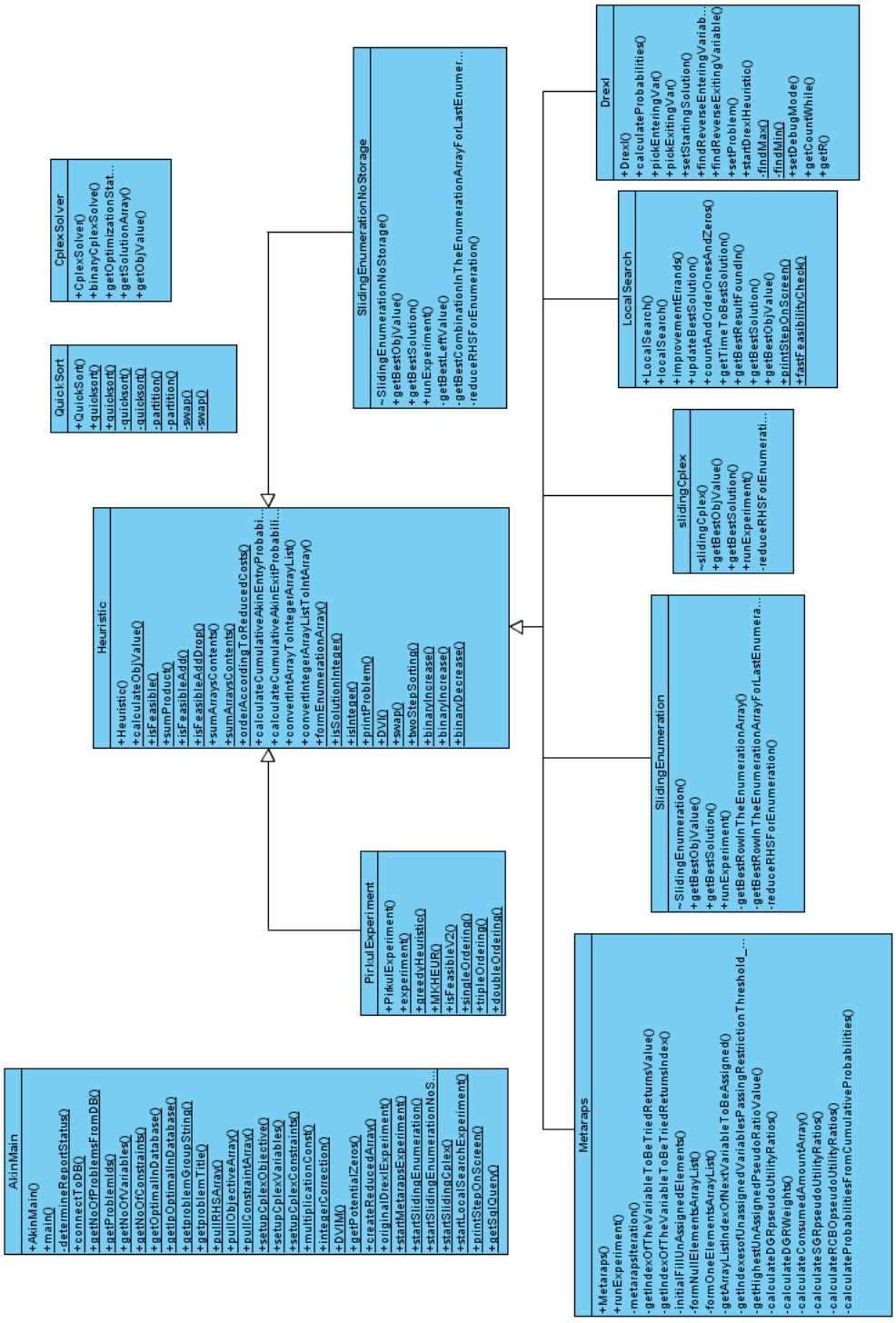

Figure 18 Class diagram of our development environment

There are also some helper classes within our heuristic. For instance we have the QuickSort class which has O(nlogn) complexity (Hoare, 1962). This class helps us order variables whenever we need. For example, within RCBO, following our initial solution of the relaxed version of the problem we order the decision variables according to their reduced cost values and we use the QuickSort class for this purpose.

Hardware Characteristics

We conducted all experiments on a personal computer with the following hardware characteristics:

- CPU Type: Intel Core Duo T9400
- CPU Speed: 2.53GHz
- Total Memory Size: 4GB

DVI Parameter Setting Experiment

We have previously explained the DVI method. Now we will use the DVI method within the sliding enumeration method to determine good parameters for DVI. The sliding enumeration width will be "10" for all the experiments. Our objective is to determine a good boundary method, a good step size and a good boundary length. For the boundary method we had two choices: "percentage" and "exact". Within the percentage method we had four levels to experiment with: 1%, 2%, 5%, and 10%. Within the exact method we had ten levels to experiment with: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. Finally, we had six step size levels, 0.01, 0.05, 0.1, 0.2, 0.5, and 1. We used a full factorial experiment design which meant we would need to do

84 experiments to determine the best parameters. The 84 experiments in total took about one day to complete. We have conducted the experiments shown in Table 6, Table 7, and Table 8 to determine the best parameters.

Table 6 Experiment parameters for the method DVI experiments

Experiment No	Boundary size (% of variables)	Step Size
1	1	0.01
2	2	0.01
3	5	0.01
4	10	0.01
5	1	0.05
6	2	0.05
7	5	0.05
8	10	0.05
9	1	0.1
10	2	0.1
11	5	0.1
12	10	0.1
13	1	0.2
14	2	0.2
15	5	0.2
16	10	0.2
17	1	0.5
18	2	0.5
19	5	0.5
20	10	0.5
21	1	1
22	2	1
23	5	1
24	10	1

We decided to test for two boundary methods: An exact boundary and a percentage boundary. The exact boundary adds an exact amount to $\sum_{x=1}^{n} x_i$ and finds the upper bound of our experiment and subtracts the same exact amount from $\sum_{x=1}^{n} x_i$ and finds the lower bound of our experiment. The percentage method however takes a percentage of $\sum_{x=1}^{n} x_i$ and uses that amount

to calculate the upper and lower bound of our DVI experiment. For instance if $\sum_{x=1}^{n} x_i$ were equal to 50, and our percentage parameter were 50%, than we would have added 25 to 50 and used 75 as the upper bound and subtracted 25 from 50 and use 25 as the lower bound.

Table 7 Experiment parameters for the exact boundary method, Experiments 1 to 30

Experiment No	Boundary size	Step Size
1	1	0.01
2	2	0.01
3	3	0.01
4	4	0.01
5	5	0.01
6	6	0.01
7	7	0.01
8	8	0.01
9	9	0.01
10	10	0.01
11	1	0.05
12	2	0.05
13	3	0.05
14	4	0.05
15	5	0.05
16	6	0.05
17	7	0.05
18	8	0.05
19	9	0.05
20	10	0.05
21	1	0.1
22	2	0.1
23	3	0.1
24	4	0.1
25	5	0.1
26	6	0.1
27	7	0.1
28	8	0.1
29	9	0.1
30	10	0.1

The main benefit of percentage boundary over exact boundary is it adapts to the problem. So as a result of these experiments we will see if it is better to adapt to the problem or if it is better to use fixed bounds on all problems

Table 8 Experiment parameters for the exact boundary method, Experiments 31 to 60

Experiment No	Boundary size	Step Size
31	1	0.2
32	2	0.2
33	3	0.2
34	4	0.2
35	5	0.2
36	6	0.2
37	7	0.2
38	8	0.2
39	9	0.2
40	10	0.2
41	1	0.5
42	2	0.5
43	3	0.5
44	4	0.5
45	5	0.5
46	6	0.5
47	7	0.5
48	8	0.5
49	9	0.5
50	10	0.5
51	1	1
52	2	1
53	3	1
54	4	1
55	5	1
56	6	1
57	7	1
58	8	1
59	9	1
60	10	1

Parameter: Step Size

Table 9 shows the performance of different step size values over both boundary methods. It is clear that as the step size is reduced, the number of optima increase and the average % deviation from optimum decrease. The best step size for both of the boundary methods is 0.01 without any dispute. The disadvantage of 0.01 is that the experiment time starts to increase dramatically between 0.05 and 0.01. However we can tolerate 140 ms per experiment on average. So we propose 0.01 as the best step size to use within the DVI method. Hence, in the following tables we will only show the results of the experiments where the step size is equal to .01.

Table 9 Performance of different step size values over both boundary methods

Boundary Method	Step Size	Avg. % Deviation From Optimum	No of Optima	Avg. of Experiment Time (ms)
Exact	0.01	0.11	18186	142
	0.05	0.11	18179	36
	0.1	0.11	18173	23
	0.2	0.11	18111	16
	0.5	0.11	18014	12
	1	0.12	17803	13
Percentage	0.01	0.07	7477	148
	0.05	0.07	7476	39
	0.1	0.07	7467	25
	0.2	0.08	7426	18
	0.5	0.08	7370	13
	1	0.11	7143	12

Parameter: Boundary Method and Its Level

In this section instead of showing overall results in one table we will show our results over different problem sets table by table. The main reason behind this approach is our problem sets have different characteristics and considering the number of levels in our experiments these diverse characteristics could accidentally manipulate our understanding of the results.

Table 10 shows the results over the Cho et al. problem set which has 270 problems. Overall the best results in this table are on the exact boundary method with a boundary size of two. The percentage method does not turn out to be the better method for the problem set in terms of average percentage deviation, however it is the better in terms the number of optima.

Table 10 Boundary method and level's performance results over the Cho et al problem set

Boundary Method	Boundary Size	Avg. % Deviation From Optimum	Number of Optima	Avg. Experiment Time(ms)
Exact	1	0.18	109	34
	2	0.14	122	62
	3	0.19	122	90
	4	0.21	110	118
	5	0.27	106	147
	6	0.29	100	174
	7	0.28	98	202
	8	0.36	87	231
	9	0.43	71	258
	10	0.5	61	287
Percentage	1%	0.18	102	21
	2%	0.17	113	37
	5%	0.16	125	83
	10%	0.17	120	160

Table 11 shows the results over the Chu and Beasley problem set which has 270 problems. Again the exact method performs better than the percentage method. However this time the boundary size of one performs better than the boundary size of two. Average experiment times per problem are similar within the two different boundary methods.

Table 11 Boundary method and level's performance results over the Chu and Beasley set

Boundary Method	Boundary Size	Avg. % Deviation From LP Optimum	Number of Optima	Avg. Experiment Time(ms)
Exact	1	0.86	7	74
	2	0.89	5	126
	3	0.93	1	177
	4	0.95	2	230
	5	1.01	1	281
	6	1.07	1	330
	7	1.12	1	383
	8	1.2	0	434
	9	1.28	0	485
	10	1.36	0	537
Percentage	1%	0.9	6	115
	2%	0.9	4	207
	5%	0.93	2	492
	10%	1.12	1	987

Table 12 shows the results over the Drexl problem set which has 56 problems. The reader should remember this is the smallest problem set we have, it has only 56 problems. The results in this table are in line with the previous results. Again the exact method performs better. Specifically, an exact boundary of one turns out to be the best boundary for the Drexl problem set. However the reader should notice the small difference between the exact boundary of one and two. While we find 40 optima in one level we find 39 optima in the other.

Table 12 Boundary method and level's performance results over the Drexl problem set

Boundary Method	Boundary Size	Avg. % Deviation From Optimum	Number of Optima	Avg. Experiment Time(ms)
Exact	1	0.18	40	22
	2	0.19	39	40
	3	0.42	38	59
	4	0.53	39	77
	5	0.39	38	97
	6	0.29	37	113
	7	0.27	39	133
	8	0.31	34	150
	9	0.36	33	167
	10	0.31	37	187
Percentage	1%	0.21	40	8
	2%	0.21	39	12
	5%	0.22	39	26
	10%	0.19	40	50

Table 13 shows the results over the Hill and Reilly problem set which has 2240 bi-dimensional 0-1 MKPs. Again the better results are found by the exact method. Even though the percentage deviation for the exact boundary of two and percentage of five are equal, the number of optima is different.

Table 14 again shows the results over the Glover and Kochenberger problem set with 11 problems. The results are very similar on this big sized problem set as well. The exact method performs better than the percentage method. The best exact bound turns out to be "two" as it were in some of the previous problem sets.

Table 13 Boundary method and level's performance results over the Hill and Reilly set

Boundary Method	Boundary Size	Avg. % Deviation From Optimum	Number of Optima	Avg. Experiment Time(ms)
Exact	1	0.03	1725	23
	2	0.02	1835	43
	3	0.02	1830	62
	4	0.02	1797	81
	5	0.03	1742	101
	6	0.03	1720	119
	7	0.04	1651	140
	8	0.06	1589	157
	9	0.07	1511	177
	10	0.09	1408	196
Percentage	1%	0.04	1586	13
	2%	0.03	1682	23
	5%	0.02	1824	51
	10%	0.03	1754	97

Table 14 Boundary method and level's performance over Glover and Kochenberger problems

Boundary Method	Boundary Size	Avg. % Deviation From LP Optimum	Number of Optima	Avg. Experiment Time(ms)
Exact	1	0.45	0	884
	2	0.39	0	1234
	3	0.43	0	1596
	4	0.45	0	1981
	5	0.47	0	2301
	6	0.47	0	2662
	7	0.49	0	3017
	8	0.47	0	3370
	9	0.49	0	3733
	10	0.52	0	4069
Percentage	1%	0.45	0	4052
	2%	0.43	0	7350
	5%	0.46	0	17551
	10%	0.53	0	34629

In conclusion, the exact method is the clear choice over the percentage method. It is however not that easy to choose the level of the exact bound. An exact bound of one and two both performed well. Since we need to make a choice between the two levels we choose "2" as our preferred exact level. This choice particularly relies on the fact that this level performs very well on the Glover and Kochenberger problem set which has the largest problems. We realize that this level was not the best for the Drexl and the Chu and Beasley problem sets however Glover and Kochenberger's problem sizes are very important to us, solving larger problems are very important hence our choice of "2" as the exact bound level.

Sliding Enumeration Parameter Setting Experiments

This section discusses experiment results of the method which we call "sliding enumeration". It consists of two phases. First, ordering the decision variables according to an ordering strategy. Second, conducting a series of enumerations. Its pseudo-code was presented in Figure 13.

Ordering Strategies

We will experiment with the sliding enumeration method with two different ordering strategies:

Reduced cost based ordering (RCBO), the pseudo-code of this method was previously presented in Figure 8. This method orders the decision variables in a binary problem according to their reduced cost values obtained by solving them using the bounded variable simplex method. When the reduced values of two variables are equal we use the LP relaxed solution values of

those variables as the primary tie-breaker and the objective coefficients as the secondary tie-breakers.

For decision variable importance (DVI), the pseudo-code of DVI was previously presented in Figure 7. For this experiment we have used “exact bounds” which are equal to “2” and a step size of “0.1” for DVI.

Results

We experiment with 20 levels of the enumeration width, from 1 to 20. We use all the problem sets except the Glover and Kochenberger set in this experiment.

Figure 19 shows the time performance of the RCBO based sliding enumeration. The reader can see that the time performance starts to get worse particularly around “16” and “17”. Since this is a technique based on enumeration such an exponential behavior was to be expected. Figure 20 for DVI shows similar results. The time performance starts to get dramatically worse around “16” and “17”. We drew these figures to find out exactly where infeasible long experiment times would start.

In terms of the time performance one could probably choose “16” as the best enumeration width for this implementation. However, we believe 20 is not a bad choice as well, after all, because 8 seconds per problem on average is not an extremely long time and when the objective is to get very good results it would be acceptable to use “20” as the enumeration width.

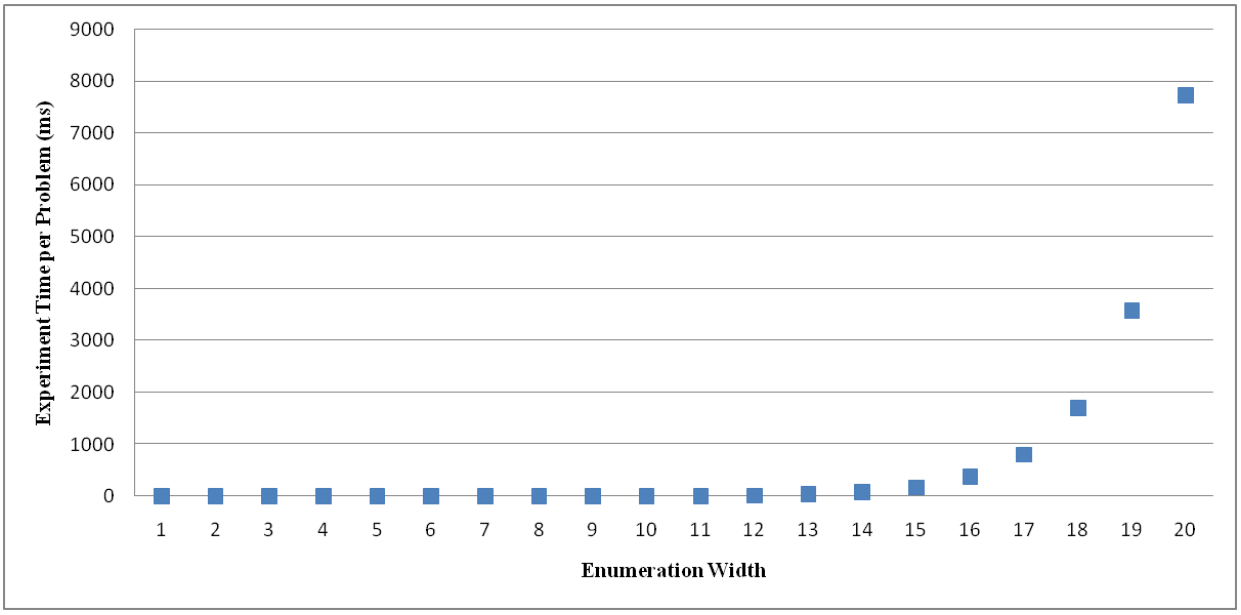

Figure 19 Experiment times with RCBO ordering strategy

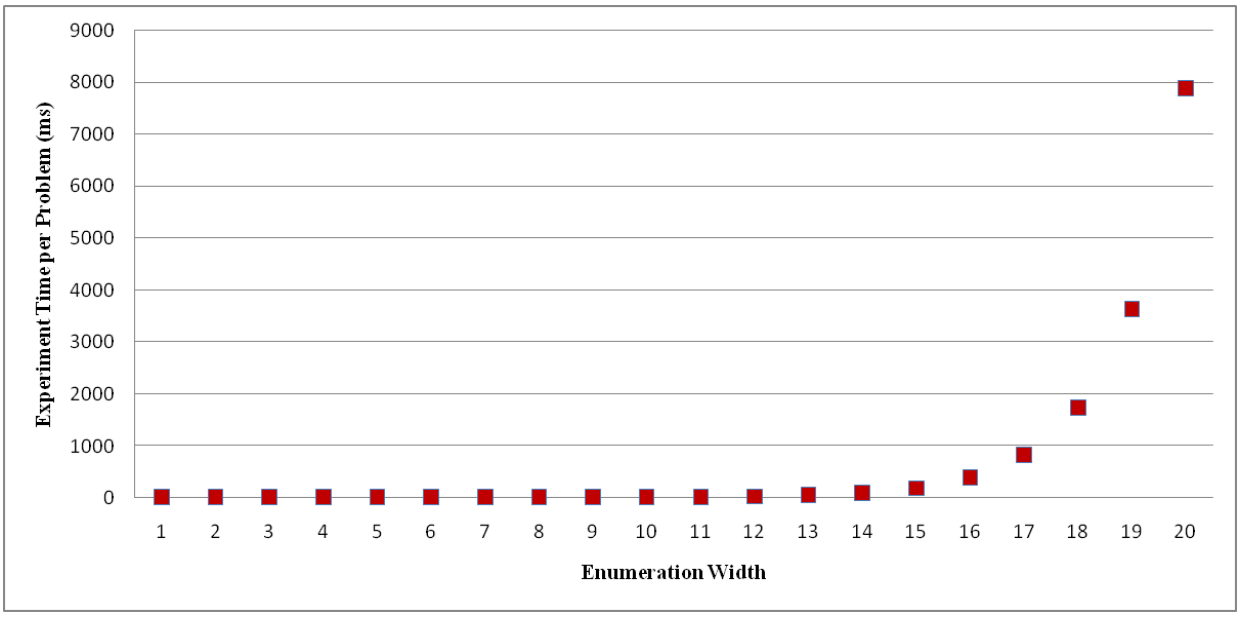

Figure 20 Experiment times with DVI ordering strategy

Figure 21 shows the average % deviation from the optimum. In this figure our objective is to compare the performance of RCBO vs DVIM. Up until a width of 4, DVI performs better than RCBO, beyond that RCBO dominates DVI. We have noted this previously as well but it is worth noting again that RCBO's main advantage over DVI is its elegance and simplicity. DVI requires a good understanding and highly complex coding. However, RCBO is trivial to understand and to code. So with the following results at hand RCBO is definitely the better choice. We should note that even if the DVI results were "slightly" better, we would still pick RCBO over DVI due to its simplicity.

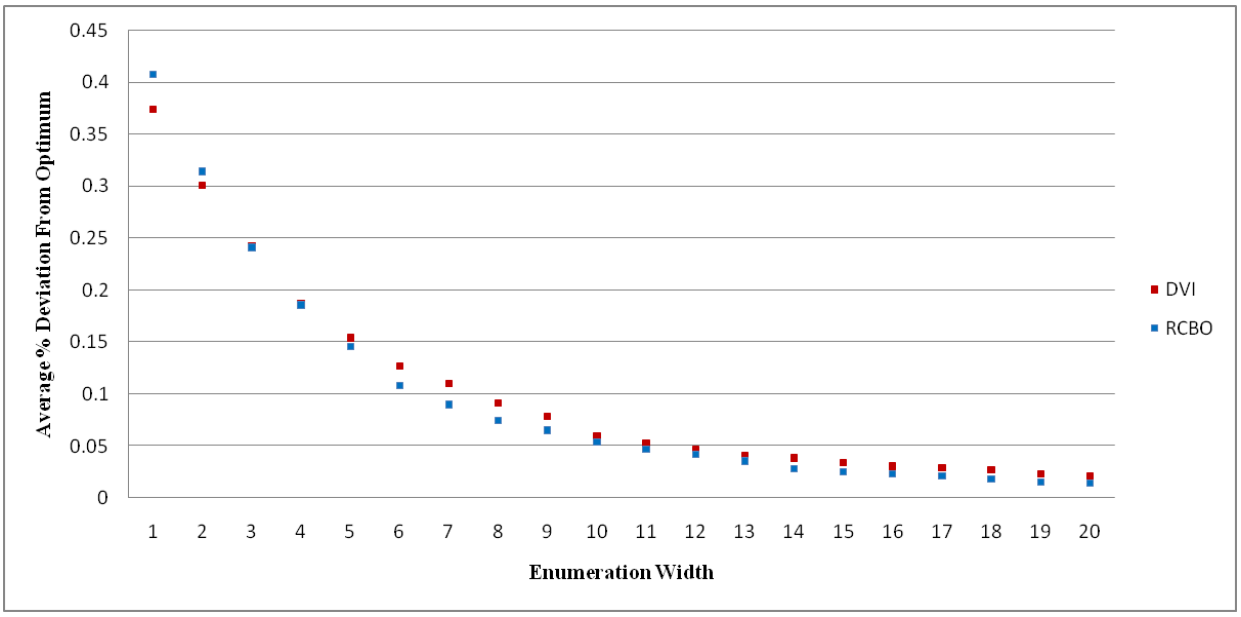

Figure 21 Average % deviation from optimum

Figure 22 and Figure 23 show the performance of sliding enumeration using RCBO. The % deviation from optimum decreases as the width increases. Percent of optimums show that we are very successful in solving three of the problem sets, the only one we are not successful is the

Chu and Beasley problem set. It is known to be the hardest problem set. We should note here that even in the literature only 180 of the optimums out of the 270 problems in Chu and Beasley problem set are available.

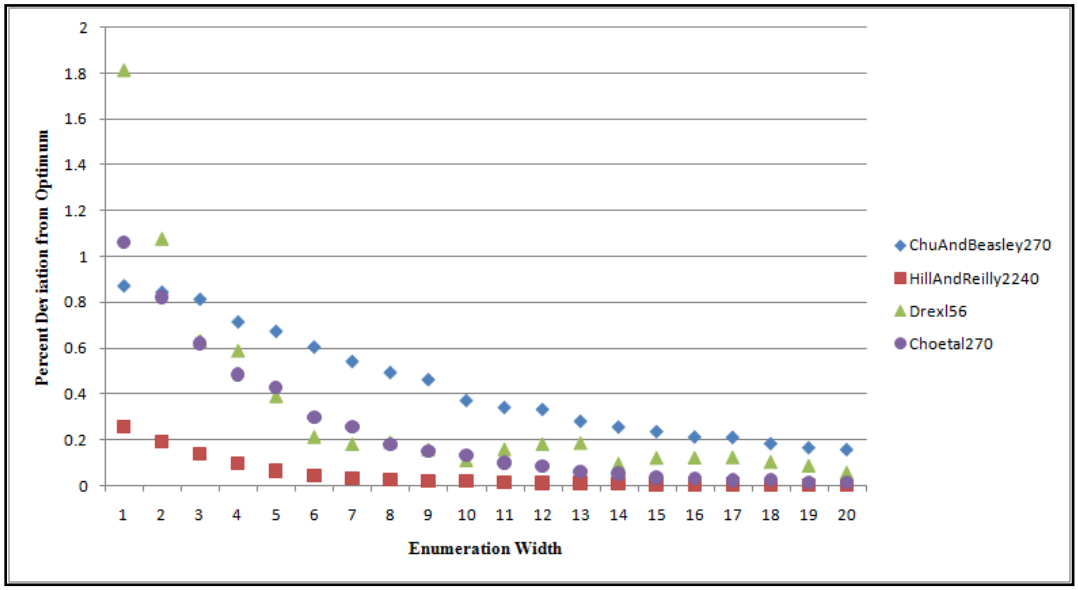

Figure 22 Average % deviation from optimum using RCBO

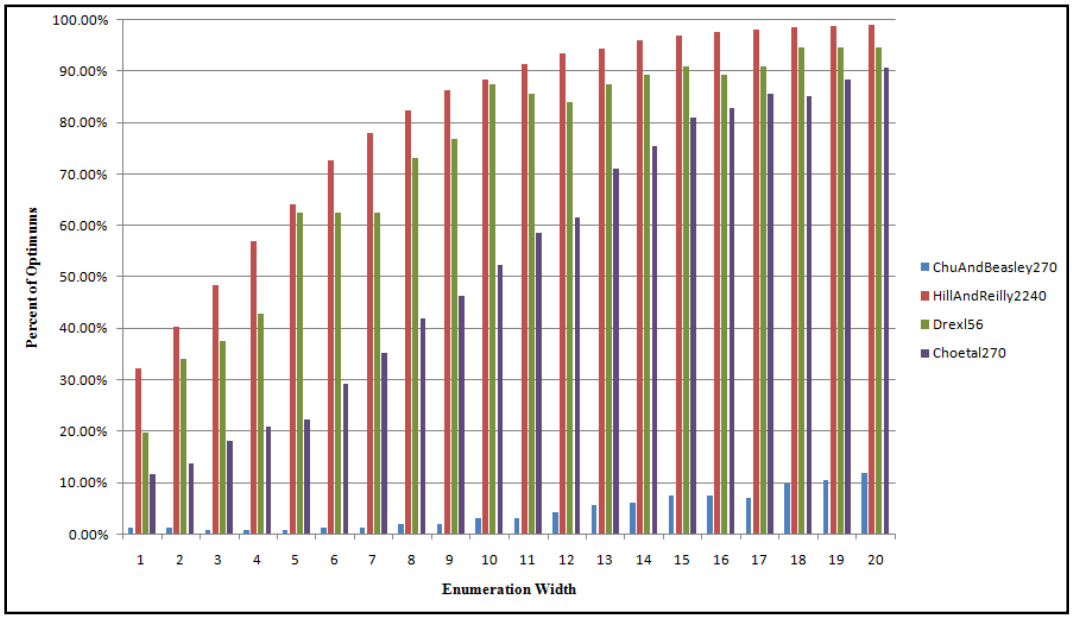

Figure 23 Percent of optimums using RCBO

Figure 24 and Figure 25 are sliding enumeration results using DVI. The results are similar in the sense that they improve as the enumeration width increases. However, we see that DVI does not perform as well as RCBO. Particularly on the Cho et al. and Hill and Reilly problem sets the percent of optima is less compared to RCBO.

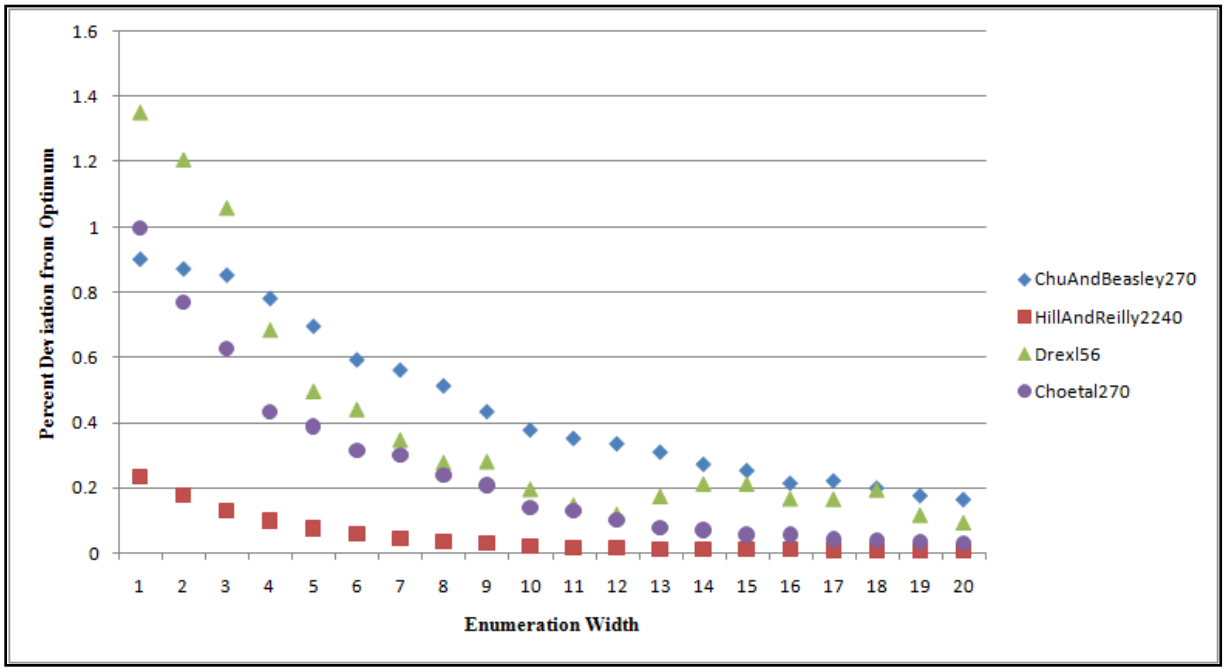

Figure 24 Average % deviation from optimum using DVI

For the Chu and Beasley problem set the difference between DVI and RCBO seems to be minimal. RCBO also slightly performs slightly better than DVI over the Drexl problems. However we should remember that the Drexl problem set is a small problem set with only 56 problems therefore it is hard to make complete conclusions solely on this problem set.

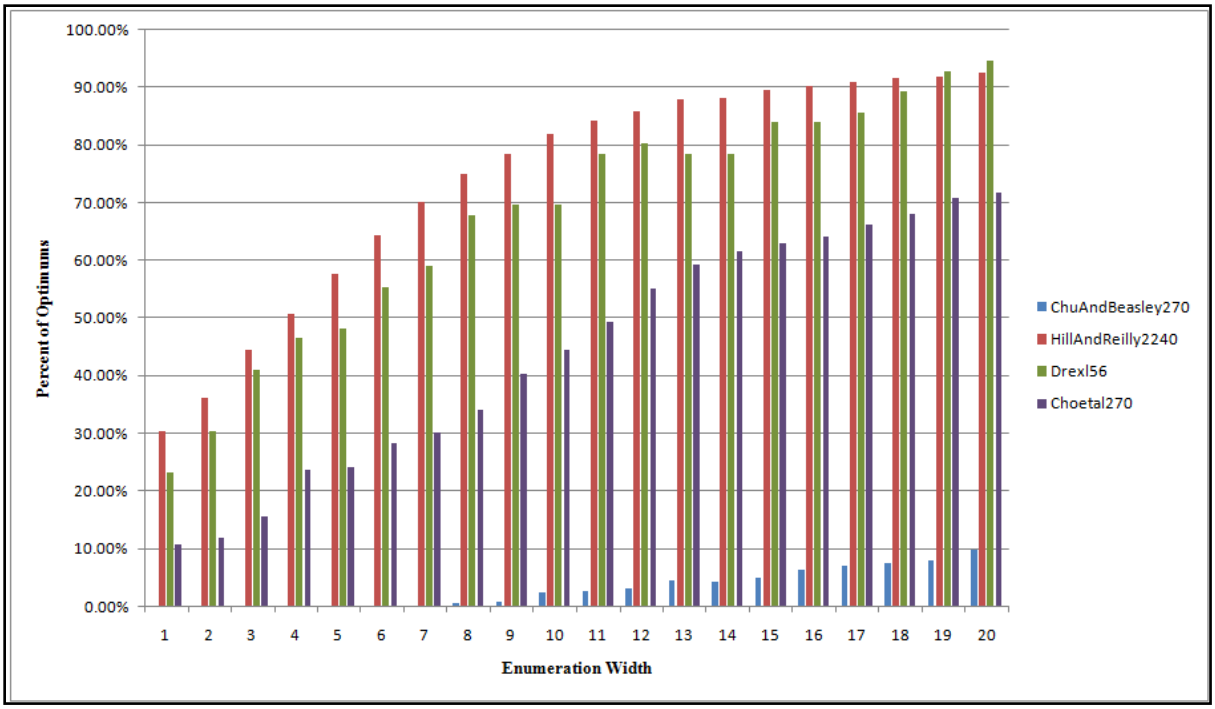

Figure 25 Percent of optimums using DVI

In conclusion, RCBO performs better than DVI. Hence RCBO is our pick of choice for sliding enumeration. As the choice of width, 16 or 17 would be the logical choice if want to make sure experiment times are at the very minimum. However, a width of "20" is good for us as well since 8 seconds per problem is not that long. That is why our very final experiment will be "Sliding enumeration of 20 with RCBO and local search".

Improved Drexl Simulated Annealing Experiments

One of the side products of this dissertation work was the use of discrete right triangular distribution. In this experiment our objective is to re-implement Drexl's (1988) simulated annealing heuristic and try to enhance it using a right triangular distribution. Drexl's simulated

annealing heuristic chooses entering and leaving variables from a given solution at certain points of the heuristic. The heuristic chooses these variables randomly (discrete uniform distribution). We will experiment to see if choosing these variables according to a discrete right triangular distribution will improve the solutions.

Ordering Strategy

As we have mentioned before, the important part when using a right triangular distribution is to decide how the variables will be ordered. Previously we have explained several ordering strategies in the literature. We will use the ordering strategy we suggested previously, reduced cost based ordering (RCBO).

A Review of Drexl's Pseudo-Code

Drexl presents his pseudo-code in detail so the interested reader can refer to the original paper for full details. We are however providing a short review of his pseudo-code in the next paragraph.

Drexl's heuristic starts from a random solution and randomly picks an entering solution. If it is feasible to enter that variable to the solution it keeps the new solution, and goes back to picking a new variable. However, if the new solution is not feasible then it randomly tries to remove a variable. If that is not possible then depending on a parameter called the "temperature," it can choose to move to a solution which is worse than the "best solution known so far". The main idea is as it is laid out in this paragraph.

Implementation Remarks

Re-implementing someone else's heuristic is not always a trivial task. Sometimes, one needs to make assumptions where the logic is not clear, or sometimes it is possible to run into problems which we do not know how the original author would have resolved. Hence in this subsection we explain several implementation differences with the original author of the heuristic.

Our first difference with Drexl's pseudo-code happens when a potential entering variable's and a potential exiting variable's objective coefficients are "equal". Drexl elects to swap these variables. However, this leads to infinite loops in some small problems. We avoid this loop by treating this swap as a negative benefit which is equal to "-0.1". This way there is a good possibility that the swap will occur but as the heuristic moves on the possibility of the swap not occurring increases. Another obvious solution could have been keeping a tabu list of the past swaps to avoid cycling. However this would have been extensive modification hence we did not use the tabu approach.

Our second difference from Drexl is that we do not use totally new random numbers in each experiment. We used Java API's java.util.Random class for random number generation. We initialized the random number generator using a seed. The "seed" value is always set to the current number of the iteration. For instance, if we are at iteration 2, we set the seed as "2". This makes our experiments repeatable, since we get the same random number stream at the same iterations. So whenever we re-run the experiment we get the same final results as long as we do not change the parameters or the logic.

Our third difference is, Drexl starts each heuristic run with a "random feasible solution". Considering that he only makes 10 runs per problem this makes it very hard to replicate his results. Thus instead of using the "random feasible solution" to start the heuristic we offer two alternative starting solutions. The first alternative is the origin. The second alternative solution is where all the variables with a positive reduced cost value are set to 1, we call this the "reduced cost based" solution.

Our final change is experimenting with a second iteration number parameter. Drexl experimented with 10 iterations which is quite normal for 1988's computational limits. However today we have more CPU power so we also experiment with 100 iterations to see how much benefit is received from increased number of iterations.

Parameters

We have three main parameters:

- Random variable's cdf
- Initial solution type
- Number of iterations

Random variable's cdf:

- Uniform: This is the method used by Drexl. In this method, each variable is treated equally whenever we pick a variable to enter or leave a solution.
- Triangular: This the method suggested by us. A discrete right triangular distribution using RCBO.

Initial solution:

- Origin: The initial solution is the origin.
- Reduced cost based: The decision variables which have a positive reduced cost value are set to “1”, the rest are set to “0”.

Number of iterations:

- 10: Drexl used 10 iterations. Hence we repeat his choice.
- 100: We are using 100 iterations to see the effect of increased number of iterations on simulated annealing.

Table 15 lists the 8 experiments and their associated parameters. The reader will notice this is a full factorial design since we are experimenting with all the levels of all the factors.

Table 15 List of our Drexl experiments and their parameters

Experiment #	Random variable	Initial solution	Number of Iterations
1	Triangular	Origin	10
2	Triangular	Origin	100
3	Triangular	Reduced cost based	10
4	Triangular	Reduced cost based	100
5	Uniform	Origin	10
6	Uniform	Origin	100
7	Uniform	Reduced cost based	10
8	Uniform	Reduced cost based	100

Validation

First we will validate if our implementation is valid. Table 16 shows results from three experiments. The third and fourth columns are Drexl’s own results. The fifth and sixth columns

are our implementation with the origin as the starting solution. The seventh and eighth columns are again our implementation, but this time the starting solution is reduced cost based.

We used two tailed paired-t test to see if our re-implementation was successful to replicate Drexl's pseudo-code. The experiment with the initial solution origin was found to be significantly different at p=0.0149 (t=2.5150). However the experiment starting from a "reduced cost based" solution was not significantly different from Drexl's results (t=0.6809, p=0.4988). Besides the t-test another point of validation is a visual inspection of the results. Drexl solves 23 of the problems to optimality. We solve 18 of these 23 problems to optimality as well.

Table 16 Comparison of our results to Drexl's original results

Problem	n	m	Drexl original Deviation From Optimum (%)	Time (ms)	Initial solution: Origin Deviation From Optimum (%)	Time (ms)	Initial solution: Reduced cost based Deviation From Optimum (%)	Time (ms)
Wei Shih 1	30	5	0	11.0	0	0.9	0	0.6
Wei Shih 2	30	5	0	6.9	0.46	0.7	0	0.7
Wei Shih 3	30	5	0	8.7	0	0.9	0	0.8
Wei Shih 4	30	5	0	5.6	0	0.6	0	0.4
Wei Shih 5	30	5	0	6.2	0	0.5	0	0.6
Wei Shih 6	40	5	0	14.4	0.43	1.2	0.61	1.1
Wei Shih 7	40	5	0	11.4	0	1.4	0	1.4
Wei Shih 8	40	5	0.04	12.4	0.04	1.2	0	1.2
Wei Shih 9	40	5	0	15.5	0	1.1	0	1.1
Wei Shih 10	50	5	0.58	14.9	0	1.8	0	1.4
Wei Shih 11	50	5	0	14.5	0.07	1.4	0	1.6
Wei Shih 12	50	5	0.02	15.3	0.50	1.5	0	1.5
Wei Shih 13	50	5	0	15.9	0	1.6	0.31	1.4
Wei Shih 14	60	5	0	24.8	0	2.1	0	2.5
Wei Shih 15	60	5	0	18.1	0	2.1	0	2.4
Wei Shih 16	60	5	0	28.5	0.40	2.6	0.03	3
Wei Shih 17	60	5	0	26.9	0.16	2.5	0	2.7
Wei Shih 18	70	5	0	33.0	0.07	3.7	0.49	3.9
Wei Shih 19	70	5	0.17	25.3	0.58	3.1	0.91	2.8
Wei Shih 20	70	5	0.10	28.1	0.38	3.8	0.40	4.2
Wei Shih 21	70	5	0.43	28.0	0.55	2.8	0.26	3.2
Wei Shih 22	80	5	0.62	31.5	1.44	3.9	0.79	4.2
Wei Shih 23	80	5	0.59	38.9	1.13	3.8	1.19	4.4
Wei Shih 24	80	5	0.05	36.0	0.42	4.6	0.49	4.5
Wei Shih 25	80	5	0.40	29.6	0.69	5.3	0.36	5
Wei Shih 26	90	5	1.00	65.3	0.66	5.9	1.27	6
Wei Shih 27	90	5	0	68.2	2.34	4	0.41	5.9
Wei Shih 28	90	5	0.26	64.6	1.54	4.5	0.32	5.6
Wei Shih 29	90	5	0.98	33.7	1.75	5.6	1.64	5.2
Wei Shih 30	90	5	0.48	35.7	0.23	6.2	0.15	6.4
Petersen 1	6	10	0	0.7	0	0.1	0	0.1
Petersen 2	10	10	0	5.4	0	0.4	0	0.5
Petersen 3	15	10	0	4.1	0.25	0.3	0	0.5
Petersen 4	20	10	0.16	1.0	3.59	0.6	0.16	0.7
Petersen 5	28	10	0.08	10.5	3.47	1.1	0.56	1.3
Petersen 6	39	5	0.32	31.3	0.13	3.6	0.80	4.3
Petersen 7	50	5	0.56	36.8	0.73	6.5	0.16	5.3
Hansen, Plateau 1	28	4	0.37	9.3	1.40	1.2	2.19	1.1
Hansen, Plateau 2	35	4	0.42	19.0	2.20	1.2	0.41	1.4
Weingartner 1	28	2	0	17.8	0.57	1.6	0.52	1.7
Weingartner 2	28	2	0.63	22.8	3.24	1.6	0	1.9
Weingartner 3	28	2	0.27	17.6	0.80	1.8	0.68	2.1
Weingartner 4	28	2	0	12.9	1.86	1.1	2.28	1.2
Weingartner 5	28	2	0.49	25.0	3.82	1.9	0.79	1.9
Weingartner 6	28	2	0.13	15.6	4.64	1.6	0	1.5
Weingartner 7	105	2	0.10	127.8	0.41	35.6	0.08	34.7
Weingartner 8	105	2	0.68	71.1	0.68	12.3	0.14	12.1
Senju, Toyoda 1	60	30	0	31.4	0.14	5.7	1.16	5.8
Senju, Toyoda 2	60	30	0.25	49.4	0.10	9.5	0.13	8
Fleisher	20	10	0.78	3.4	2.52	1.1	2.52	1
Pb 1	27	4	0.79	9.2	3.17	1.1	1.68	1.2
Pb 2	34	4	0.41	15.5	2.57	1	1.98	1.2
Pb 3	19	2	0.17	10.1	N/A	N/A	N/A	N/A
Pb 4	29	2	0.41	10.1	0.00	1	4.48	0.9
Pb 5	20	10	0.79	3.0	2.01	0.5	0.79	0.4
Pb 6	40	30	0	12.2	1.80	1.7	0	1.7
Pb 7	37	30	0.10	10.5	1.84	1.8	0.10	1.3
Average			0.24	23.7	1.00	3.16	0.56	3.20

Considering that Drexl's logic extensively uses random numbers we believe we have successfully re-implemented Drexl's method by using the reduced cost based starting solutions on the basis of paired t-test and the matching of the problems for which the optimums are found. Hence, in our analysis we disregard the experiments starting from the origin.

Results

Figure 26 shows the percent of optima we found in four of the problem sets we have. We should note here the last column is our closest solution to Drexl's results. Hence it should be seen as Drexl's re-implementation without any changes.

There is a clear difference between the use of triangular and uniform distributions. Triangular distribution performs dramatically better. There is also a noticeable difference of results due to the starting solution types particularly when the uniform distribution is used.

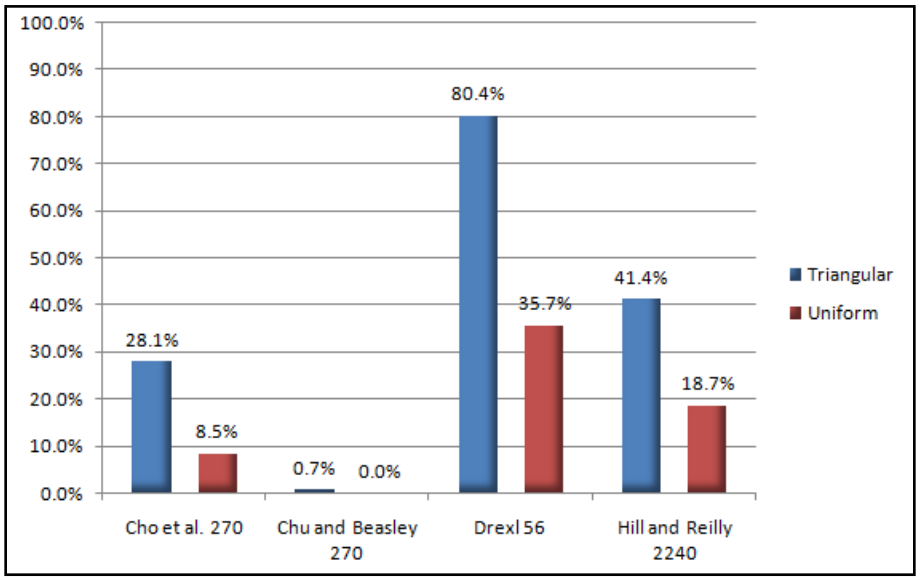

Figure 26 Percentage of optimums for 10 iterations

Figure 27 shows that the number of optimums can increase with the increase of iterations.

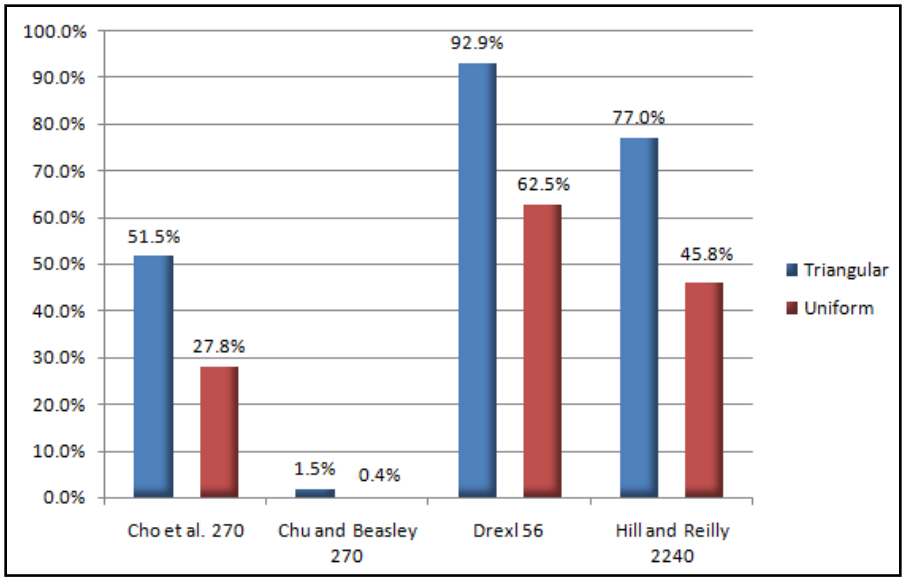

Figure 27 Percentage of optimums for 100 iterations

Table 17 and Table 18 show the % deviation from the optima. "*% deviation from optimum*" is calculated as follows:

$$\%\ Deviation\ from\ optimum = \frac{Optimum\ -Experiment\ \ Best\ Objective\ \ Value}{Optimum} x100 \qquad (17)$$

The results are very similar to the previous results. The triangular distribution dramatically improves results against the uniform distribution. The only difference from the previous results is that starting from the origin works better on the Drexl problem set which has smaller number of variables compared to the other problem sets.

Table 17 Average percentage deviation from optimum for 10 iterations

Problem Set	No of Problems	Triangular	Uniform
Cho et al.	270	0.3008*	0.9509
Drexl	56	0.1443	0.5579
Hill and Reilly	2,240	0.0772*	0.2616
Average	2,566	0.1022*	0.3406

*: Best result for the problem set

Table 18 Average percentage deviation from optimum for 100 iterations

Problem Set	No of Problems	Triangular	Uniform
Cho et al.	270	0.1038*	0.3683
Drexl	56	0.0179	0.1884
Hill and Reilly	2,240	0.0153*	0.0852
Average	2,566	0.0247*	0.1172

*: Best result for the problem set

The Chu and Beasley problem set was not shown in Table 17 and Table 18 because we do not know all the optimal solutions to this problem set.

Table 19 and Table 20, where we use percentage deviation from the LP optimum as a measure of performance, "*% deviation from LP optimum*" is calculated as follows:

$$\%\, Deviation\ from\ LP\ optimum = \frac{LP\ Optimum\ -Experiment\ \ Best\ Objective\ \ Value}{LP\ Optimum} x100 \qquad (18)$$

Deviation from LP optimum results shown in Table 19 and Table 20 are also in line with the previous results. Triangular distribution out performs the uniform distribution. Triangular distribution out performs the uniform distribution.

Table 19 Average percentage deviation from LP optimum for 10 iterations

Problem Set	No of Problems	Triangular	Uniform
Cho et al.	270	0.9033*	1.5475
Chu and Beasley	270	1.2311*	1.8059
Drexl	56	1.6175	2.0245
Hill and Reilly	2,240	0.1859*	0.3701
Average	2,835	0.3820*	0.6516

*: Best result for the problem set

Table 20 Average percentage deviation from LP optimum for 100 iterations

Problem Set	No of Problems	Triangular	Uniform
Cho et al.	270	0.7075*	0.9701
Chu and Beasley	270	0.9990*	1.3856
Drexl	56	1.4952	1.6608
Hill and Reilly	2,240	0.1241*	0.1939
Average	2,835	0.2900*	0.4102

*: Best result for the problem set

For information purposes we are providing the run times (ms) per iteration in Table 21 and Table 22.

Table 21 Average experiment time (ms) per iteration for 10 iterations

Problem Group	No of Problems	Triangular	Uniform
Cho et al.	270	27	30
Chu and Beasley	270	131	260
Drexl	56	5	3
Hill and Reilly	2,240	12	8
Average	2,835	25	34

Table 22 Average run time (ms) per iteration for 100 iterations

Problem Group	No of Problems	Triangular	Uniform
Cho et al.	270	26	29
Chu and Beasley	270	127	257
Drexl	56	5	3
Hill and Reilly	2,240	12	8
Average	2,835	24	34

As a conclusion we see that, in meta-heuristics like Drexl's, where the method heavily depends on the random numbers, the right triangular distribution helps improve the results dramatically. The probabilistic approach we have provided in this paper ensures that all variables have a chance to enter the solution. Hence it makes it more possible to move from one solution to another solution. The right triangular distribution is very simple and it shows that random number use is a very important part of heuristic development. Frequently, random numbers are not the best choice and more complicated distributions needs to be chosen to better represent the likelihood of entering or exiting variables. Further research is necessary to examine distributions which can potentially better fit the 0-1 MKP problems.

Improved Pirkul Heuristic Experiments

In this section we will show how we can improve Pirkul's greedy heuristic MKHEUR. Again we will use the 0-1 MKP for our experiments. Pirkul's MKHEUR was previously used as a benchmark by several authors, Chu and Beasley (1998), Moraga, DePuy and Whitehouse (2005), Akcay, Li and Xu (2007) can be mentioned as some of the important works.

An Analysis of Pirkul's Ordering Strategy

Pirkul's main idea is to order decision variables according to the equation defined in (9).

$$\frac{c_j}{\sum_{i=1}^{m} \lambda_i a_{ij}} \tag{9}$$

The problem with Pirkul's method is that it starts to decay as the problem size increases, particularly with the increase of constraints. Akcay et al. (2007) notes this decay and claims that their method, PECH, would catch up with Pirkul's method and surpass it on large problems. We have previously claimed that the particular decay of Pirkul's heuristic method MKHEUR is because it cannot distinguish in between partial variables since the value it calculates is always equal to "1" for partial variables. Hence, as the number of partial variables in the LP optimum solution increases MKHEUR performs worse. So actually it is not the high number of constraints which makes a problem harder to solve for MKHEUR but the number of partial variables. A problem can have a high number of constraints but a small number of partial variables. The Cho et al. problem set is a good example such problems.

Another problem with MKHEUR, due to the same reasoning, is that it is open to different implementations. Therefore it is hard to repeat previous experiments unless their exact code is available.

Table 23 shows the number of fractional values in the Chu and Beasley problem set. The reader should notice the high ratio of the fractional variables to the number of constraints.

Table 23 . Number of fractional variables in a Chu and Beasley dataset

Constraints	Variables	No of Problems	Avg. No of Fractional Variables	St. Dev. of No of Fractional Variables	Min No of Fractional Variables	Max No of Fractional Variables
5	100	30	5.00	0	5	5
5	250	30	5.00	0	5	5
5	500	30	5.00	0	5	5
10	100	30	9.83	0.38	9	10
10	250	30	10.00	0	10	10
10	500	30	10.00	0	10	10
30	100	30	20.97	2.13	16	27
30	250	30	24.70	1.97	21	30
30	500	30	27.30	1.68	24	30

Table 24 shows the same analysis on a more recent data set by Cho et al. The reader would notice that the number of fractional problems is less compared to the Chu and Beasley problem set. Thus, it is going to be a good control data set to see if the decay is due to the number of fractional variables or not.

Table 24 . Number of fractional variables in Cho et al. dataset

Constraints	Variables	No of Problems	Avg. No of Fractional Variables	St. Dev. of No of Fractional Variables	Min No of Fractional Variables	Max No of Fractional Variables
5	50	30	2.33	0.66	1	3
5	100	30	1.80	0.55	1	3
5	250	30	2.17	0.87	1	5
10	50	30	3.00	0.87	2	5
10	100	30	2.83	1.02	1	5
10	250	30	2.63	0.72	1	4
25	50	30	3.73	1.23	2	7
25	100	30	3.73	1.26	1	6
25	250	30	4.17	1.20	2	6

Tie breakers

Now that we have shown the potential reason behind MKHEUR's inefficiency for large problems we will propose potential improvements. To fix the problem we propose to use a tie-breakers where Pirkul's value is equal to "one". The tie-breaker we propose to use is the tie-breaking method we have used within the RCBO technique. We will use the values, x_j^*, as tie-breakers. Higher priority will be given to decision variables with larger values. In case there are still decision variables which we cannot differentiate in between then we will use the objective coefficient of that decision variable as secondary tie breaker.

RCBO

Since we have already shown the relationship between reduced costs and Pirkul's values it is also necessary to see what would happen if we were to replace Pirkul's values with reduced cost values in MKHEUR. The reader should notice even though Pirkul's values and reduced costs will be highly correlated they will not be in the exact same order.

A Note on Implementing RCBO vs. Pirkul Values

Our results show that, in the context of MKHEUR, there's no significant difference between using Pirkul's values or reduced costs for ordering strategy.

From a theoretical stand point, since reduced costs are calculated using the shadow prices, Pirkul values could be calculated prior to reduced costs. Therefore it should be faster to calculate them. However in practice, today, researchers use software packages to solve linear programming problems. Both shadow prices and reduced costs are available to the user in times

far less than a single millisecond. Therefore instead of using Pirkul's values, using reduced costs is faster and less open to errors.

Results

In Table 25 each row represents 30 problems. The third and fourth columns present Chu and Beasley's and our MKHEUR implementations respectively. As we have discussed previously, MKHEUR, as it is outlined by Pirkul, is open to different interpretations between implementations. This experiment also validates this claim. The reader should notice that as the number of constraints, hence the number partial variables, increases a gap between the Chu and Beasley results and our results starts to open up. That is why we provided our results in the 4th column, it would not have been fair to compare our results to Chu and Beasley's MKHEUR implementation considering the potential implementation differences. We also made a paired t-test and found that our implementation results were not significantly different from Chu and Beasley's results (t=1.9862, p=0.0823).

In Table 25, the fifth column represents the MKHEUR implementation with Pirkul's ordering strategy but with our new tie-breakers, so it is the advanced version of Pirkul. The sixth column represents the MKHEUR implementation with RCBO strategy. The difference between a Pirkul ordering and RCBO is clearly insignificant. However both of these methods are superior to the regular implementations both by us and by Chu and Beasley.

The last column in Table 25 represents Akcay et al. (2007)'s greedy heuristic PECH's results. In their research, they have also shown that the gap between their and Chu and Beasley's

MKHEUR implementation closes up as the problem size increases. However we see here that the difference between the advanced Pirkul implementation and PECH does not close up fast.

Table 25 . Percentage deviation from the LP optimum for the Chu and Beasley data set

Constraints	Variables	MKHEUR (Chu and Beasley implementation)	MKHEUR (Our implementation)	MKHEUR w/ advanced Pirkul	MKHEUR w/ RCBO	PECH
5	100	0.95	0.96*	1.03	1.04	4.28
5	250	0.31	0.32	0.31*	0.31*	4.03
5	500	0.12	0.14	0.13*	0.13*	3.85
10	100	2.11	1.75	1.50*	1.50*	4.58
10	250	0.66	0.67	0.56*	0.56*	3.31
10	500	0.29	0.30	0.24*	0.25	2.92
30	100	4.85	4.16	2.63*	2.63*	3.98
30	250	2.02	1.59	1.07*	1.07*	2.69
30	500	1.03	0.90	0.54*	0.54*	2.09
Average		1.37	1.20	0.89*	0.89*	3.46

*: Best result for that constraint, variable combination

Table 26 shows a similar analysis but this time over the Cho et al. dataset. As we have expressed before, the Cho et al. dataset is important to us because their problems have less fractional variables at the LP optimum solution. Since the decay we have presented in Table 25 is not present in Table 26 that is another clear sign that the decay of Pirkul's MKHEUR is a result of the number of fractional variables and Pirkul's indifference within these variables.

We conclude that the main reason MKHEUR performs worse on large problems in the Chu and Beasley dataset is the high ratio of fractional variables to constraints and it is possible to fix this decay using tie-breaking strategies. Additionally we have shown that it is possible to use the RCBO method within MKHEUR instead of Pirkul values and it would be equally successful.

Implementing RCBO also simplifies the calculations required by MKHEUR, hence makes the implementation relatively easier.

Table 26 . Percentage deviation from the LP optimum for the Cho et al. data set

Constraints	Variables	MKHEUR (Our implementation)	MKHEUR w/ advanced Pirkul	MKHEUR w/ RCBO
5	50	1.17*	1.22	1.26
5	100	0.43*	0.43*	0.52
5	250	0.11*	0.11*	0.13
10	50	2.10	2.03*	2.16
10	100	0.67*	0.70	0.73
10	250	0.22	0.21*	0.22
25	50	3.00	2.77*	3.05
25	100	1.08*	1.12	1.16
25	250	0.40*	0.40*	0.42
	Average	1.02	1.00*	1.07

*: Best result for that constraint, variable combination

Sliding enumeration using RCBO plus local search

Previously in the parameter setting section for sliding enumeration we showed results of sliding enumeration with different enumeration widths. In that section an enumeration width of 20, and an RCBO ordering strategy performed the best. In this section we will take those experiments one step further and conduct a local search around our sliding enumeration solution. First show results over the Chu and Beasley's 270 problems. The results in Table 27 show Chu and Beasley's (1998) (C&B) and Vimont et al.'s (2008) (VBV) experiments perform better than our method. Our main difference from their methods is that both of them employ complex meta-heuristics to obtain their results. Chu and Beasley for instance, employs a detailed genetic

algorithm where there are several choices such as how the initial population should be generated, what the mutation rate should be, how the crossover operator should work and also how the repair operator should work.

Table 27. Deviation comparison of the RCBO method for the Chu and Beasley problems

		Average % deviation from LP optimum				
m	n	Sliding Enumeration &Local Search	MR	C&B	VBV	Sliding Enum. Avg. Exp Time (s)
5	100	0.61	0.60	0.59	N/A	46
5	250	0.18	0.17	0.14	0.14	270
5	500	0.07	0.09	0.05	0.04	300
10	100	1.02	1.17	0.94	N/A	112
10	250	0.38	0.45	0.30	0.28	290
10	500	0.19	0.20	0.14	0.10	300
30	100	1.83	2.23	1.69	N/A	154
30	250	0.76	1.38	0.68	0.62	300
30	500	0.42	0.82	0.35	0.28	300
Overall		0.61	0.77	0.53	N/A	230

Vimont et al.'s experiment is even more complicated than the Chu and Beasley experiment. They use a tabu search to calculate lower bounds and then they create cuts on the problem space using reduced costs, they also fix variables and finally they start a branch and bound algorithm. All of these methods we just listed have several internal parameters as well. They report results at the end of 4 hours which is substantially longer than our 5 minute experiment time limit.

In summary, our experiment starts with a sliding enumeration using RCBO as outlined in Figure 13. The enumeration width is 20. Once the sliding enumeration concludes we make a local search around the sliding enumeration's resulting solution. We have presented the local search's pseudo-code in Figure 15 and Figure 16. We employ a 5 minute limit on our heuristic. The heuristic terminates itself either when its 5 minutes runs out or it exhausts all the local solutions. The results in the previous table show that even though sliding enumeration using RCBO with local search is less complex than the other methods, it still performs very competitively.

In our experiments we also kept track of the exact time(ms) when we found "our" best result for each problem. Figure 28 shows the time distribution of those best solution times for the Chu and Beasley problem set. The graph shows that in 180 of the problems we reached the best solution in the first 10 seconds and we did not find a better solution for the rest of the five minutes. We found the best solution in the first minute for 219 problems which is 81% of the 270 problems. So for future experiments we can conclude that instead of a 5 minute threshold we can use a one or maybe at most two minute threshold without losing much performance.

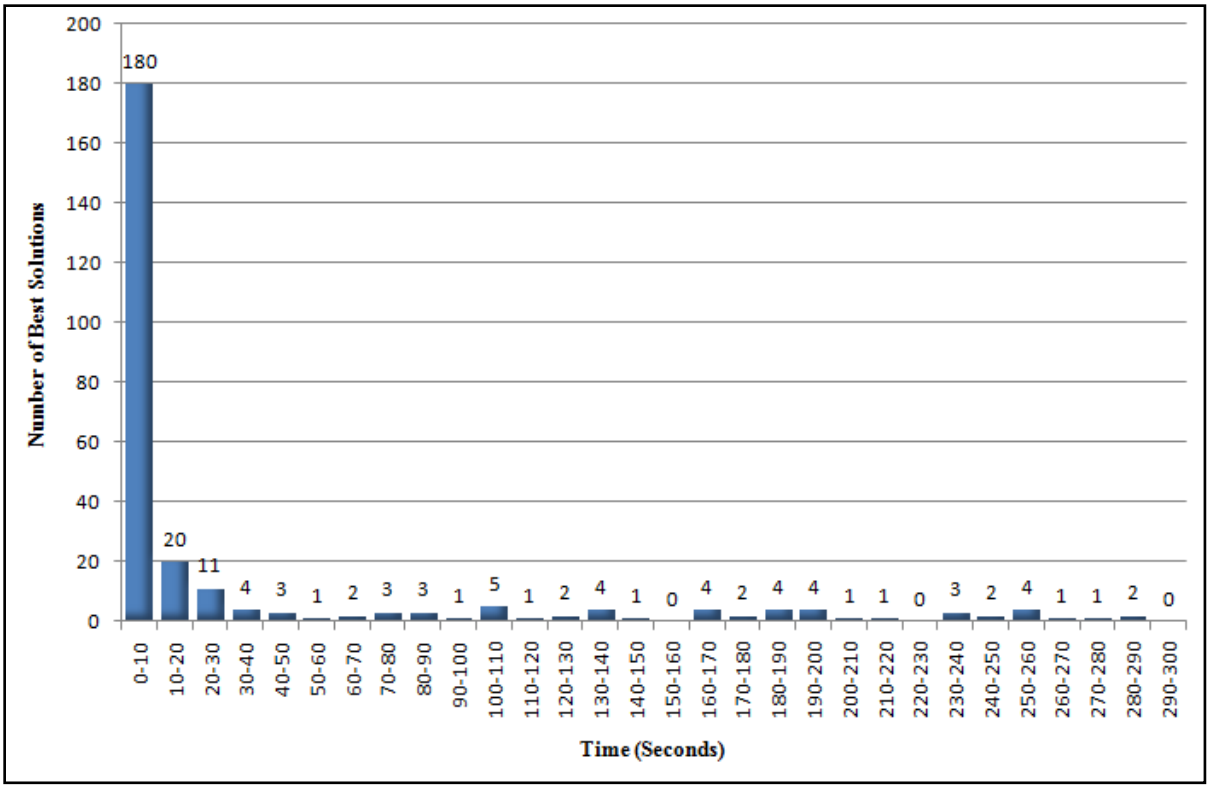

Figure 28 Time to best solution distribution

Table 28 shows sliding enumeration and local search's results over the Cho et al (2008) problem set. Cho et al problem set has 270 problems just like the Chu and Beasley problem set. They created this problem set as an alternative to the Chu and Beasley problem set. This problem set has more controlled parameters compared to the Chu and Beasley problem set and hence our hope is to gain more understanding about the 0-1 MKP. However as the results show our method performs very well over the problem set, which is unfortunate in the sense that it does not allow us to gain much insight on the 0-1 MKP. The table also compares our results with Cho et al.'s results and our results dominate this table. Under the light of these findings we should note that it might be interesting to consider increasing the dimensions Cho et al.'s problem sets. The reader

would notice that Cho et al.'s problems were easier to solve compared to Chu and Beasley's problems. One reason behind this could be the number of partial variables at the LP optimum. Chu and Beasley problems characteristically have many partial variables; Cho et al. problems however, have a small number of partial variables. We believe this is a potential lead for future researchers who will try to detect or maybe build hard 0-1 MKPs.

Table 28 Sliding Enumeration and Local Search Results Over the Cho et al. Problems

		Cho et al.	Sliding Enum. & Local search		
m	n	% Deviation From Optimum	% Deviation From Optimum	No of Optimums	Avg. Time to Best Solution(ms)
5	50	0.991	0	30	2305
5	100	0.406	0	30	3130
5	250	0.218	0.003	27	7728
10	50	1.634	0	30	2597
10	100	0.605	0	30	3457
10	250	0.275	0.003	28	4250
25	50	1.730	0	30	2772
25	100	1.005	0	30	4601
25	250	0.932	0.014	21	16900
	Overall	0.866	0.002	256	5304

In the Drexl problem set we were able to find the optimum for 53 of the 56 problems. In the Hill and Reilly problem set we were able to find the optimum for 2239 of the 2240 problems. For the Hill and Reilly problems it took 2.6 seconds on average to reach the best solution.

In our final problem set, Glover and Kochenberger, we still got good results but we were dominated by the research due to Vasquez and Hao (2001a). Vasquez and Hao's work is a little less advanced version of the Vimont et al. work we discussed previously in this section. Considering they performed better than our work in prior problem sets their results being better was not unexpected. We should note that our results very close to theirs. Another point we

should not is in their paper they discuss it took 3 full days to complete the Glover and Kochenberger problems. We reached the following best results in 10 minutes in total and again we should stress the simplicity of our method.

Table 29 Comparison of Results over the Glover and Kochenberger Problems

		Sliding Enumeration		Sliding Enum. &Local Search		
n	m	Best result	Time to Best Result (s)	Best result	Time to Best Result (s)	Vasquez and Hao's Results
100	15	3759	2	3766	4	3766
100	25	3949	3	3951	4	3958
150	25	5648	4	5649	159	5656
150	50	5756	4	5760	55	5767
200	25	7547	4	7552	5	7560
200	50	7663	5	7665	13	7677
500	25	19201	10	19205	165	19220
500	50	18790	11	18793	191	18806
1500	25	58073	91	58078	98	58087
1500	50	57273	91	57278	99	57295
2500	100	95207	156	95216	164	95237

Next table compares “sliding enumeration” vs. “sliding enumeration & local search”. Our goal is to see how much time burden “local search” adds, and how much our results improve. This will help us see the benefit provided by the local search. In terms of “percentage deviation from LP optimum”, local search does not add much benefit except the Chu and Beasley problem set and the time to best solution is very close as well. Again, as we have mentioned previously, it is evident that the 5 minute local search was redundant. A one or two minute local search at most would have been enough.

Table 30 Comparing Sliding Enumeration vs. Sliding Enum & Local Search

	Sliding Enumeration		Sliding Enum & Local Search	
Problem set	Avg. Time to Best Solution(ms)	Avg. % Deviation From LP Optimum	Avg. Time to Best Solution(ms)	Avg. % Deviation From LP Optimum
Cho et al.	3,092	0.62	5,304	0.61
Chu And Beasley	15,810	0.68	36,629	0.61
Drexl	1,316	1.54	1,526	1.54
Hill And Reilly	1,426	0.11	2,527	0.11

The next figure shows a similar analysis but this time we are calculating the percentage of the optimums we found in each problem set. Again local search does not add that much improvement except the Chu and Beasley problems. Considering that the Chu and Beasley problem set is the hardest problem set in the literature, we deem the 7 percent improvement we gained from the local search very important. One possible conclusion we get from this comparison is a good heuristic should perform better when complemented with a local search. This has already been a practice in the past. Pirkul (1987), Moraga et al. (2005) are two sample works which complete their works with some sort of local search technique. Our results validate that there is merit to this practice. Even though some problems can be solved easily without local search methods, it is a good practice to conduct a local search.

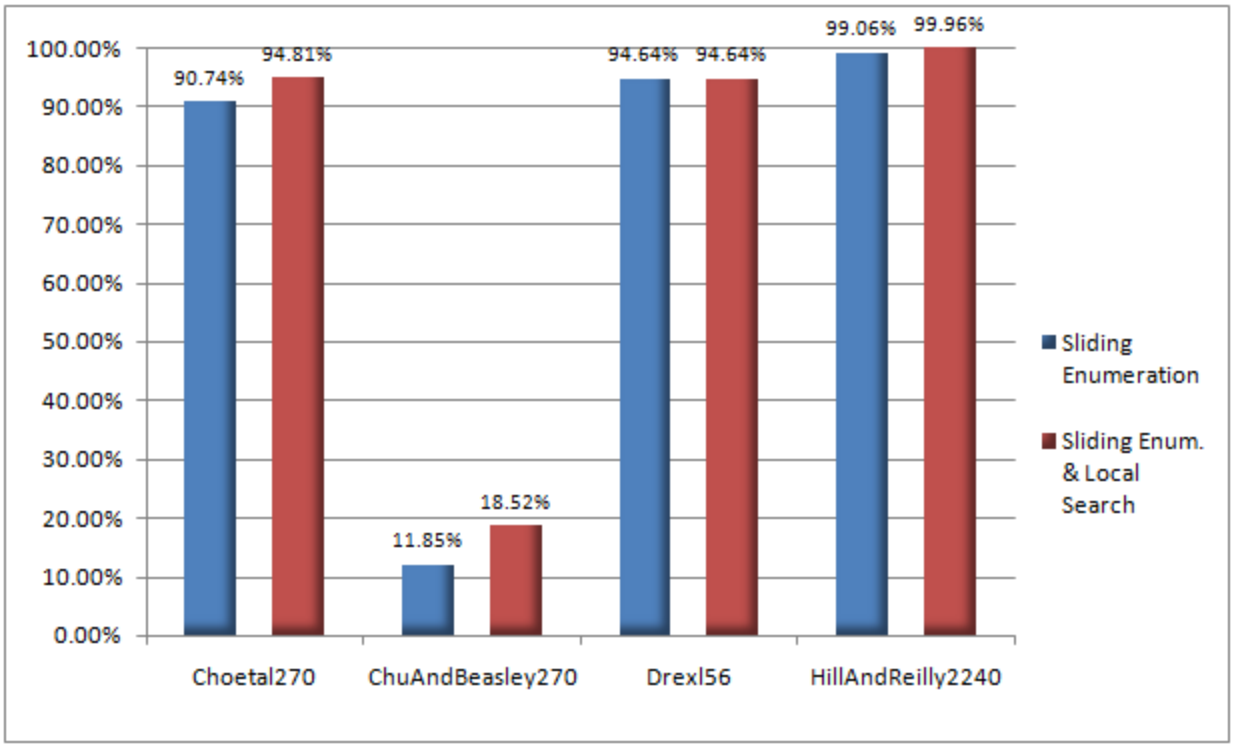

Figure 29 Percent of Optimum Comparison

CHAPTER SEVEN: CONCLUSIONS

As a result of our experiments we have successfully untapped the use of reduced costs. We presented different ways of using reduced costs for the 0-1 MKP. We presented a probabilistic method, a "discrete right triangular distribution" based probabilistic approach, two ordering strategies, "RCBO", and DVI and a concept called "sliding" and under that concept a method called "sliding enumeration". In this chapter we will list our contributions and discuss future research recommendations.

Contributions

RCBO is a brand new ordering strategy which proved useful in several methods such as improving Pirkul's MKHEUR, the triangular distribution based probabilistic approach and most importantly sliding enumeration. Using reduced costs for an ordering strategy is a very trivial task which requires almost no calculation at all by the researchers since LP solvers readily provide reduced cost values once they solve an LP. This ensures minimum deviation between different implementations of the RCBO.

We showed that Pirkul's ordering strategy based on shadow prices fails to order the partial variables. We presented a possible fix to this problem which worked good on our problem sets. This fix improved Pirkul's MKHEUR's performance significantly on problems with a high number of partial variables. There tends to be a high number of partial variables in hard problems therefore this insight will help future researchers solve hard problems with more success.

We showed that in meta-heuristic approaches such as Drexl's simulated annealing where random numbers are abundantly used, it would be better to use special probability distributions instead of random numbers. We showed that it is possible to use a discrete right triangular distribution based on RCBO to model the likelihood of a variable being a "one" in the optimal solution or not. This approach significantly improved Drexl's simulated annealing.

Finally we introduced the sliding concept, which is powerful but simple to understand and easy to implement greedy heuristic. We showcased a sub-branch of it which we called the sliding enumeration. Even though it is a simple method, it found optima in a few seconds for most problems. The sliding concept was inspired from the well known "core" concept in the knapsack literature however it requires less parameters and is easier to implement compared to the core concept. We also showed it is possible to replace enumeration with other exact solution procedures.

Future Research

We have noticed several potential future research areas during our studies. First and foremost, it is necessary to replace the enumeration within the sliding concept with a technique which can solve more variables at the same time. This could be the use of CPLEX to optimize the sliding variables, to use a special enumeration for the 0-1 MKP or to use a branch and bound method designed for the 0-1 MKP.

Second and very importantly, we have mentioned previously the sliding concept gets inspiration from the "core concept". There are a few past researches about the core concept for the 0-1 MKP but none of them are particularly successful. It would be worthwhile to research the

core concept using RCBO. The reader would remember in order to use the core concept the researcher needs to pick an ordering strategy first.

Third, this research, focused on simple ordering strategies. However we should note there is a lot of merit to dynamic ordering strategies like Moraga's DGR (2002) and future research should investigate the use of dynamic ordering strategies for the reduced cost values. It is possible to recalculate the reduced costs after each allocation of a variable. So, one could use the new order of the reduced costs at each new step of sliding enumeration or whichever greedy heuristic they are using.

Fourth, it is also worth studying the relation between the number of partial decision values and the hardness of the 0-1 MKPs. A very much studied hardness factor of the 0-1 MKP problems is the number of constraints and variables. Additionally the correlation structure of the problems and the slackness levels of constraints have also been discussed previously. One non-discussed point is the number of partial variables. An analysis outlining how much the number of partial variables contributes to the hardness of a problem could be an important study. It could also be studied if it would be possible to reduce the number of constraints in a 0-1 MKP to the number of partial variables.

Fifth and finally, it is repeatedly shown in literature that the 0-1 MKPs with tight constraints are hard to solve. However, there is no well known method to measure the tightness of a 0-1 MKP which has different tightness levels for each constraint. It would interesting to see if we can use the number of "one"s and the number of partial variables in the LP solution, as measure of tightness and if this information could be used to predict how hard a problem is.

LIST OF REFERENCES

Aboudi, R., & Jönsten, K. (1994). Tabu search for general zero-one integer programs using the pivot and complement heuristic. *ORSA Journal of Computing, 6*(1), 82-93.

Akcay, Y., Li, H., & Xu, S. H. (2007). Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research, 150,* 17-29.

Alaya, I., Solnon, C., & Ghedira, K. (2004). Ant algorithm for the multi-dimensional knapsack problem. In *Proceedings of International Conference on Bioinspired Optimization Methods and their Applications* (pp. 63-72).

Balas, E., & Martin, C. H. (1980). Pivot and a complement heuristic for 0-1 programming. *Management Science, 26,* 86-96.

Balas, E., & Zemel, E. (1980). An algorithm for large zero-one knapsack problems. *Operations Research, 28*(5), 1130-1154.

Beasley, J. E. (1990). OR library: Distributed test problems by electronic mail. *Journal of the Operational Research Society, 41,* 1069-1072.

Beasley, J. E. (1996). Obtaining test problems via internet. *Journal of Global Optimization, 8,* 429-433.

Beaujon, G. J., Martin, S. P., & McDonald, C. C. (2001). Balancing and optimizing a portfolio of R&D projects. *Naval Research Logistics, 48,* 18-40.

Bertsimas, D., & Demir, R. (2002). An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science, 48*(4), 550-565.

Burke, E. K., Cowling, P. I., & Keuthen, R. (2001). Effective local and guided variable neighborhood search methods for the asymmetric traveling salesman problem. In E.J.W. Boers et al. (Eds.) *Proceedings of EvoWorkshops on Applications of Evolutionary Computing*, (pp. 203-212).

Cherbaka, N. S. & Meller, R. D. (2008). Single-plant sourcing decisions with a multidimensional knapsack model. *Journal of Manufacturing Systems, 27*, 7-18.

Cho, Y. K., Hill, R. R., Moore, J. T., & Reilly, C. H. (2008). Exploiting empirical knowledge for bi-dimensional knapsack problem heuristics. *International Journal of Industrial and Systems Engineering, 4*(5), 530-548.

Chu, P. C., & Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics, 4*, 63-86.

Clements, D. P., Crawford, J. M., Joslin, D. E., Nemhauser, G. L., Puttlitz, M. E., & Savelsbergh, M. W. P. (1997). Heuristic optimization: A hybrid AI/OR approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*.

Cotta, C., & Troya, J. M. (2003). Embedding branch and bound within evolutionary algorithms. *Applied Intelligence, 18*, 137-153.

Dammeyer, F., & Voss, S. (1993). Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research, 41*, 31-46.

Dantzig, G. B. (1957). Discrete variable extremum problems. *Operations Research, 5*, 266-277.

Denzinger, J., & Offermann, T. (1999). On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the 1999 Congress on Evolutionary Computation*, 3, (pp. 2317-2324).

Drexl, A. (1988). A Simulated Annealing Approach to the Multiconstraint Zero-one Knapsack Problem. *Computing*, *40*, 1-8.

Dumitrescu, I., & Stützle, T. (2003). Combinations of local search and exact algorithms. In G. R. Raidl, J. A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, & E. Marchiori (Eds.), *Applications of Evolutionary Computation*, Vol. 2611 of LNCS, (pp. 211-223). Springer.

Eilon, S., Watson-Gabdy, C., & Christofides, N. (1971). *Distribution management: Mathematical modeling and practical analysis*. New York: Hafner Publishing Company.

Feltl, H., & Raidl, G. R. (2004). An improved hybrid genetic algorithm for the generalized assignment problem. In *Proceedings of ACM Symposium on Applied Computing*, (pp. 990-995).

Fidanova, S. (2002). Evolutionary algorithm for the multidimensional knapsack problem. *In Proceedings of PPSN VII – Workshop.*

Fidanova, S. (2003). ACO algorithm for MKP using different heuristic information. In Dimov, I., Lirkov, I., Margenov, S., and Zlatev, Z., (Eds.), *Proceedings of NM&A 2002 - Fifth International Conference on Numerical Methods and Applications, volume 2542 of Lecture Notes in Computer Science*, (pp. 438-444). Springer Verlag, Berlin, Germany.

Filho, G. R., & Lorena, L. A. N. (2000). Constructive genetic algorithm and column generation an application to graph coloring. *In The Fifth Conference Of The Association Of Asian-Pacific Operations Research Societies*.

French, A. B., Robinson, A. C., & Wilson, J. M. (2001). Using a Hybrid Genetic-Algorithm/Branch and Bound Approach to Solve Feasibility and Optimization Integer Programming Problems. *Journal of Heuristics, 7,* 551-564.

Fréville, A. (2004). The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research, 155,* 1-21.

Fréville, A., & Hanafi, S. (2005). The multidimensional 0-1 knapsack problem- Bounds and computational aspects. *Annals of Operations Research, 139,* 195-227.

Fréville, A., & Plateau, G. (1993). An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem. *European Journal of Operational Research, 68,* 418-421.

Fréville, A., & Plateau, G. (1994). An efficient preprocessing procedure for the multidimensional knapsack problem. *Discrete Applied Mathematics, 49,* 189-212.

Fréville, A., & Plateau, G. (1996). The 0-1 bidimensional knapsack problem: Towards an efficient high-level primitive tool. *Journal of Heuristics, 2,* 147-167.

Garey, M. R., & Johnson, D. S. (1979). *A Guide to the Theory of NP-Completeness*. New York: WH Freeman & Co.

Gavish, B., & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming, 31,* 78-105.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer Academic Publishers.

Glover, F., & Kochenberger G. A. (1996). Critical event tabu search for multidimensional knapsack problems. In I.H. Osman & J.P. Kelly (Eds.), *Metaheuristics: The Theory and Applications* (pp. 407-428). Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

Hanafi, S., Fréville, A., & El Abdellaoui, A. (1996). Comparison of heuristics for the 0-1 multidimensional knapsack problem. In J.H. Osman & J.P. Kelly (Eds.), *Meta-heuristics: Theory and Applications* (pp. 449-466). Kluwer Academic Publishers.

Hanafi, S., & Fréville, A. (1998). An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research, 106*, 659-675.

Hill, R. R., & Reilly, C. H. (2000). The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance. *Management Science, 46*(2), 302-317.

Hoare, C. A. R. (1962). Quicksort. *The Computer Journal*, *5*(1), 10-16.

Horowitz, E., & Sahni, S. (1974). Computing partitions with applications to the knapsack problem. *Journal of the ACM, 21*(2), 277-292.

James, R. J. W., & Nakagawa, Y. (2005). Enumeration methods for repeatedly solving multidimensional knapsack sub-problems. *IEICE - Transactions on Information and Systems, E88-D*(10), 2329-2340.

Jensen, T. R., & Toft, B. (1994). *Graph Coloring Problems*. New York: Wiley-Interscience.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica, 4*, 373-395.

Klee, V., & Minty, G. J. (1972). How good is the simplex algorithm?. *In O. Shisha (Ed.), Inequalities III*, (pp. 159-175). New York, NY: Academic Press.

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

Kochenberger, G. A., McCarl, B. A., & Wyman, F. P. (1974). A heuristic for general integer programming. *Decision Sciences, 5*, 36-44.

Kostikas, K., & Fragakis, C. (2004). Genetic programming applied to mixed integer programming. *In M. Keijzer et al. (Eds.), Genetic Programming - EuroGP 2004, Vol. 3003 of LNCS,* (pp. 113-124).

Laarhoven, P. J., & Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. Springer.

Laguna, M., & Marti, R. (2003). *Scatter Search: Methodology and Implementations in C*. Springer.

Leguizamon, G., & Michalewicz, Z. (1999). A new version of ant system for subset problem. In *Proceedings of the 1999 Congress on Evolutionary Computation*, (pp.1459-1464).

Lokketangen, A., & Glover, F. (1998). Solving zero-one mixed integer programming problems using tabu search. *European Journal of Operational Research, 106*, 624-658.

Lokketangen, A., Jornsten, K., & Storoy, S. (1994). Tabu search within a pivot and complement framework. *International Transactions of Operations Research, 1*, 305-316.

Lorie, J., & Savage, L. J. (1955). Three problems in capital rationing. *Journal of Business, 28,* 229-239.

Marino, A., Prugel-Bennett, A., & Glass, C. A. (1999). Improving graph colouring with linear programming and genetic algorithms. In K. Miettinen, M. M. Makela, and J. Toivanen (Eds.), *Proceedings of EUROGEN99,* (pp. 113-118).

Magazine, M. J., & Oguz, O. (1984). A heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research, 16*, 319-326.

Martello, S., & Toth, P. (1977). An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research, 1*, 169-175.

Martello, S., & Toth, P. (1990). *Knapsack Problems Algorithms and Computer Interactions.* Wiley-Interscience.

Moraga, R. J. (2002). *Meta-RaPS: An effective solution approach for combinatorial problems*. PhD thesis. Orlando, FL: University of Central Florida.

Moraga, R. J., DePuy, G. W., & Whitehouse, G. E. (2005). MetaRaPS approach for the 0-1 multidimensional knapsack problem. *Computers & Industrial Engineering, 48,* 83-96.

Moscato, P. (1989). *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*. Technical Report No. 790 Caltech Concurrent Computation program, California Institute of Technology, Pasadena, California, USA.

Nauss, R. M. (1976). An efficient algorithm for the 0-1 knapsack problem. *Management Science, 23*(1), 27-31.

Nash, S. G. & Sofer, A. (1996). *Linear and Nonlinear Programming*. New York: McGraw-Hill.

Osman, I. H., & Kelly, J. P. (1996). Meta-heuristics: An overview. In I.H. Osman & J.P. Kelly (Eds.), *Metaheuristics: The Theory and Applications*, (pp. 407-427). Kluwer Academic Publishers.

Osorio M. A., Glover, F., & Hammer, P. (2000). *Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions.* Technical report, Hearing Center for Enterprise Science. Report HCES-08-00.

Pirkul, H. (1987). A heuristic solution procedure for the multi-constraint zero-one knapsack problem. *Naval Research Logistics, 34*, 161-172.

Plateau, A., Tachat, D., & Tolla, P. (2003). A hybrid search combining interior point methods and metaheuristics for 0-1 programming. *International Transactions in Operational Research, 9*, 731-746.

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A Survey and Classification. In J. Mira & J. Alvarez (Eds.), *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Volume 3563 of LNCS*, (pp. 41-53).

Reilly, C. (IN PRESS). Synthetic Optimization Problem Generation: Show Us the Correlations!. *INFORMS Journal on Computing*.

Reilly, C. H., Gonsalvez, D. J. A. & Mount-Campbell, C. A. (1990). Finding Fixed Satellite Service Orbital Allotments with a k-Permutation Algorithm. *IEEE Transactions on Communications, 38*(8), 1253-1259.

Rider, P. R. (1963). Sampling from a Triangular Population. *Journal of the American Statistical Association*, *58*(302), 509-512.

Sahni, S. (1975). Approximate algorithms for the 0/1 knapsack problem. *Journal of the Association for Computing Machinery, 22*(1), 115-124.

Senju, S., & Toyoda, Y. (1968). An approach to linear programming with 0-1 variables. *Management Science, 15B*, 196-207.

Shih, W. (1979). A branch & bound method for the multiconstraint zero-one knapsack problem." *Journal of the Operational Research Society, 30*, 369-378.

Thiel, J., & Voss, S. (1994). Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms. *INFOR*, 32, 226-242.

Toyoda, Y. (1975). A simplified algorithm for obtaining approximate solutions to zero-one programming problem. *Management Science, 21*(12), 1417-1427.

Vasquez, M., & Hao, J. (2001a). A hybrid approach for the 0-1 multidimensional knapsack problem. *In Proceedings of the International Joint Conference on Artificial Intelligence 2001*, (pp.328-333).

Vasquez, M., & Hao, J. (2001b). A "logic constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications, 20*, 137-157.

Vasquez, M., & Vimont, Y. (2005). Improved results on the 0-1 multidimensional knapsack problem. *European Journal Of Operational Research, 165*, 70-81.

Vimont, Y., Boussier, S. & Vasquez, M. (2008). Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem. *Journal of Combinatorial Optimization, 15*, 165-178.

Volgenant, A., & Zoon, J. A. (1990). An improved heuristic for multidimensional 0-1 knapsack problems. *Journal of Operations Research Society, 41*, 963-970.

Volgenant, A., & Zwiers, I. Y. (2007). Partial enumeration in heuristics for some combinatorial optimization problems. *Journal of Operational Research Society, 58*, 73-79.

Weingartner, H. M., & Ness, D. N. (1967). Methods for the solution of the multi-dimensional 0/1 knapsack problem. *Operations Research , 15*(1), 83-103.