AN IMPROVED GENETIC ALGORITHM FOR KNAPSACK PROBLEMS

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Engineering

By

GAMZE KILINCLI TASKIRAN

B.S., Uludag University, 2003
M.S., Dokuz Eylul University, 2007

2010

Wright State University

____________________________________
Xinhui Zhang, Ph.D.
Thesis Director

____________________________________
S. Narayanan, Ph.D., P.E.
Department Chair

Committee on
Final Examination

____________________________________
Xinhui Zhang, Ph.D.

____________________________________
Yan Liu, Ph.D.

____________________________________
Pratik Parikh, Ph.D.

____________________________________
John A. Bantle, Ph.D.
Vice President for Research and
Graduate Studies and Interim Dean
of Graduate Studies
In this study, an improved genetic algorithm (GA) is presented to solve the multidimensional 0-1 knapsack problem (MKP). The MKP is a well-known combinatorial optimization problem and has received wide attention from the operations research community for decades. Although recent advances in computing and optimization technologies have made the solution of small and medium size instances possible, this NP-hard problem, in general, still remains one of the challenging problems yet to be solved.

Of the various algorithms developed to solve the MKP, GA seems to be one of the best methods pointed out in the literature. A GA is an iterative search procedure that simulates the evolution process of a population of individuals based on natural selection and genetics. A GA typically starts with a random initial population and uses genetic operators such as crossover and mutation to yield new offspring to replace individuals of current population. GAs, though have been successful in solving MKPs, could be slow in converging to an optimal or near optimal solution.

An improved GA is proposed in this study that aims at exploring the use of greedy heuristics and methods to generate multiple diverse solutions to speed GA convergence. Path re-linking (PR), a method to generate new solutions by exploring trajectories that connect high quality solutions, is used to combine elite solutions to
further improve the quality of solutions. The combination of uniform crossover and PR allows the integration of randomization and elite solutions analysis to achieve a balance of intensification and diversification to further improve the quality of solutions.

Computational studies of benchmark problems suggest that the proposed algorithm was able to quickly achieve good solutions while avoiding being trapped in premature convergence and is on par with some of the state-of-the-art algorithms in the literature. This study demonstrates a systematic method to explore heuristics to generate population generation with diversity, which could significantly influence the convergence of a GA to best solutions. Nevertheless, as our computational results suggest, randomization in crossover is critical for a GA in its overall performance to achieve better quality solutions.
# TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION TO THE KNAPSACK PROBLEM .......................................................... 1
  1.1 Introduction ......................................................................................................................... 1
  1.2 Solution Approaches to the Knapsack Problem ................................................................. 3
    1.2.1 Exact Methods .................................................................................................................. 4
    1.2.2 Heuristics ....................................................................................................................... 8
    1.2.3 Metaheuristics ................................................................................................................ 12

CHAPTER 2: GENETIC ALGORITHMS ..................................................................................... 16
  2.1 Genetic Algorithms ............................................................................................................. 16
  2.2 Genetic Algorithms Applied to Knapsack Problems ......................................................... 19
    2.2.1 Problem Representation ................................................................................................. 19
    2.2.2 Population ...................................................................................................................... 20
    2.2.3 Parent Selection ............................................................................................................. 22
    2.2.4 Crossover and Mutation ................................................................................................. 23
    2.2.5 Replacement ................................................................................................................... 25
  2.3 Scatter Search and Path Re-linking .................................................................................... 26

CHAPTER 3: AN IMPROVED GENETIC ALGORITHM ............................................................. 29
  3.1 A Systematic Approach to Generate Alternatives for Initial Population of a GA ........ 30
  3.2 Parent Selection, Crossover, Mutation and Path Re-linking for the Improved GA. ... 35

CHAPTER 4: COMPUTATIONAL RESULTS ........................................................................... 38
  4.1 Experimental Data and Evaluation Criteria ....................................................................... 38
4.2 MGA-GA vs. Original-GA Comparisons................................................................. 39

4.3 Path Re-linking vs. Uniform Crossover Comparisons ............................................ 44

CHAPTER 5: CONCLUSIONS ......................................................................................... 51

APPENDIX A: MGA-GA and Original-GA Comparisons............................................... 53

APPENDIX B: Figures of Selective Comparison Results for MGA-GA and Original-GA ..... 54

APPENDIX C: MGA-GA-PR and MGA-GA-Uniform Comparisons ............................. 58

REFERENCES ............................................................................................................. 60
LIST OF FIGURES

Figure 1: Path Re-linking .......................................................... 36
Figure 2: MGA-GA and Original-GA comparison for m=5, n=100, α=0.50 ................. 44
Figure 3: MGA-GA and Original-GA comparison for m=5, n=100, α=0.50 ................. 54
Figure 4: MGA-GA and Original-GA comparison for m=5, n=250, α=0.50 ................. 54
Figure 5: MGA-GA and Original-GA comparison for m=5, n=250, α=0.75 ................. 55
Figure 6: MGA-GA and Original-GA comparison for m=5, n=500, α=0.25 ................. 55
Figure 7: MGA-GA and Original-GA comparison for m=10, n=250, α=0.25 ............... 55
Figure 8: MGA-GA and Original-GA comparison for m=10, n=250, α=0.50 ............... 56
Figure 9: MGA-GA and Original-GA comparison for m=10, n=500, α=0.25 ............... 56
Figure 10: MGA-GA and Original-GA comparison for m=10, n=500, α=0.75 .............. 56
Figure 11: MGA-GA and Original-GA comparison for m=30, n=250, α=0.25 .............. 57
Figure 12: MGA-GA and Original-GA comparison for m=30, n=500, α=0.50 .............. 57
LIST OF TABLES

Table 1: Complete set of optimal solutions to the sample problem ......................... 33
Table 2: MGA-GA and Original-GA comparison ......................................................... 40
Table 3: MGA-GA results for $10^6$ steps ................................................................. 42
Table 4: Computational times for Original-GA and MGA-GA .................................... 43
Table 5: Original-GA-PR and Original-GA-Uniform comparison ................................. 45
Table 6: MGA-GA-PR and MGA-GA-Uniform comparison .......................................... 45
Table 7: MGA-GA-PR/Uniform and MGA-GA-Uniform comparison for $10^5$ steps ....... 47
Table 8: MGA-GA-PR/Uniform and MGA-GA-Uniform comparison for $10^6$ steps ........ 48
Table 9: MGA-GA-PR/Uniform and Original-GA comparison for $10^6$ steps ............... 48
Table 10: MGA-GA-PR/Uniform and MGA-GA-Uniform and Original-GA comparisons ... 49
Table 11: MGA-GA-PR/Uniform, MGA-GA-Uniform, and Original-GA solution times .... 50
Table 12: Original-GA and MGA-GA results for each case and different steps ............... 53
Table 13: MGA-GA-PR/Uniform (10% elite ratio) and MGA-GA-Uniform comparison ....... 58
Table 14: MGA-GA-PR/Uniform (10% elite ratio) and Original-GA comparison .............. 59
ACKNOWLEDGEMENT

I would like to express my sincere appreciation and gratitude to Prof. Dr. Xinhui Zhang for his academic guidance and enthusiastic encouragement throughout the research. I would also like to express my special thanks again to him for giving me the opportunity to share his experience and insights during this process that inspired creativity and motivation. I also would like to thank my thesis committee members Dr. Yan Liu and Dr. Pratik Parikh for their time and comments.

I wish to thank my husband, my parents and my sister who have always been an important source of support, encouragement and endless love.
Dedicated to:

My Unborn Baby Girl
CHAPTER 1: INTRODUCTION TO THE KNAPSACK PROBLEM

1.1 Introduction

The multidimensional 0–1 knapsack problem (MKP) is an NP-hard combinatorial optimization problem (Garey and Johnson, 1979). The problem is an extension of the standard 0-1 knapsack problem with many constraints while the standard 0-1 knapsack problem has only one constraint.

The objective of a MKP is to maximize the sum of the values of the items to be selected from a given set by taking into account multiple resource constraints. The problem has been extensively studied in the literature for decades both because of its theoretical interest and its wide applications in operations research, computer science, management science, and various engineering fields.

Basically, the MKP can be formulated as follows:

Maximize  \[ \sum_{j=1}^{n} c_j x_j \]  \hspace{1cm} (1)

Subject to  \[ \sum_{j=1}^{n} a_{ij} x_j \leq b_i \hspace{1cm} \forall i = 1,..,m \]  \hspace{1cm} (2)

\[ x_j \in \{0,1\}, \hspace{1cm} 1 \leq j \leq n \]  \hspace{1cm} (3)

where \( n \) is the number of items and \( m \) is the number of knapsack constraints with capacity \( b_i \) (\( i = 1,..,m \)), and \( x_j \) are decision variables where \( x_j = 1 \) if the item \( j \) is selected, 0 otherwise. Each item \( j \) (\( j = 1,..,n \)) requires \( a_{ij} \) units of resource consumption in the \( i \)th
constraints and yields $c_j$ units of profit upon inclusion. Equation (1) calculates the total profit of selecting item $j$ and equation (2) ensures each knapsack constraint is satisfied. Equation (3) states the binary selection requirement on decision variable $x_j$. The goal here is to find a subset of items that yields the maximum profit without exceeding various resource capacities such as volume and weight of the knapsack. By its nature, all entries are nonnegative. It is further assumed that $a_{ij} > b_i$ and $\sum_{j=1}^{n} a_{ij} > b_i$ for all $i$ ($i = 1,..,m$) since otherwise some or all the variables can be fixed to 0 or 1.

There are different types of knapsack problems in the literature such as bounded, multiple-choice, multi-dimensional, multi-objective, etc. The classical knapsack problem tries to select the subset from a finite set of items, which maximizes a linear function of the items chosen subject to a single inequality constraint. The 0-1 knapsack problem, in which the variables are restricted to be binary, is a specific case ($m=1$) of the MKP, and can be solved in a pseudo-polynomial time. A MKP extends the classical knapsack problem to $m$ constraints; if $m=2$, it becomes a bi-dimensional knapsack problem. In the multi-choice 0-1 knapsack problem, the item set is partitioned into subsets and a solution must include exactly one item in each subset. In the bounded multiple-choice 0-1 knapsack problem, additional constraints are included to restrict the number of items that can be selected in each subset.

Many practical problems can be modeled as a knapsack problem, such as the capital budgeting, resource allocation (McMillan and Plaine, 1973), vehicle loading problems (Shih, 1979), cutting stock problem (Gilmore and Gomory 1966, Martello and
Toth 1990), combinatorial auctions (Rothkopf et al., 1998), collapsing problem (Fayard and Plateau, 1994), economic planning (Pisinger, 1995) to name a few.

A MKP is also one of the most well-known optimization problems because numerous complex optimization problems can be transformed or solved through a series of knapsack-type sub-problems by some relaxation methodologies. For example, 1) the binary knapsack problem is used as a subproblem when solving the generalized assignment problem, as well as the vehicle routing problem, 2) the set covering problems, widely used in flight and crew scheduling can be reformulated as MKP through variable complementing. As such, the knapsack problem has attracted much theoretical interest and is often used as a benchmark problem to compare or validate solution approaches in the field of combinatorial optimization. For some surveys on the MKP and its association with other problems, please see Gavish and Pirkul (1985), Freville and Plateau (1986), Martello and Toth (1990), Freville and Plateau (1994), Pisinger (1995), Chu and Beasley (1998), and Freville (2004).

1.2 Solution Approaches to the Knapsack Problem

Computationally, various approaches have been proposed for solving the MKPs. These algorithms can be classified into two categories, 1) exact algorithms, and 2) heuristics or meta-heuristics.
1.2.1 Exact Methods

Exact methods for MKP began several decades ago and include branch-and-bound method, special enumeration techniques and reduction schemes, and Lagrangean methods and surrogate relaxation methods.

Lorie and Savage (1955) investigated Lagrangean multipliers in solving the knapsack problem, and proposed a Lagrangean heuristic for 0–1 integer programming, in which all of the constraints were relaxed in the objective function. Everett (1963) formalized Lorie and Savage (1955)’s approach. The surrogate strategy introduced by Glover (1965) replaces the original constraints by a single surrogate constraint. Greenberg and Pierskalla (1970) proposed the first major treatment of surrogate constraints in the context of general mathematical programming, and was followed by the studies of Glover (1968, 1975), Dyer (1980). Karwan and Rardin (1984) and Karwan et al. (1987) provided search procedures to find surrogate and composite multipliers for general integer programs. Martello and Toth (1990) showed experiments with an algorithm solving uncorrelated and weakly correlated instances with up to more than 100,000 variables. Crama and Mazzola (1994) presented an important result on the improvement in the bound that can be realized with relaxations. Freville (2004) proposed that although more effort is required to calculate the bounds, surrogate relaxation methods are more useful for solving the MKP than those using Lagrangean relaxation, because Lagrangean relaxation framework is not appropriate for tackling the simple and homogeneous structure of the MKP.
Balas (1965), and Geoffrion (1969) developed implicit enumeration techniques to solve 0–1 linear programs, and Lemke and Spielberg (1967), and Breu and Burdet (1974) investigated the computational effectiveness of several optimizing 0–1 codes based on these techniques. The ability of these implicit enumeration based branch-and-bound methods for solving MKP instances remained rather limited, and have not been competitive with other more significant approaches. Gilmore and Gomory (1966) proposed a dynamic programming algorithm and Marsten and Morin (1977, 1978) combined dynamic programming and branch-and-bound approaches to solve the MKP, including the introduction of low time-consuming heuristics and LP bounds.

Shih (1979) proposed the first linear programming-based branch-and-bound method using the special structure of the MKP, and found an upper bound by solving m single constrained knapsack problems. He reported computational experiments with a group of thirty randomly generated and uncorrelated problems with up to five knapsack constraints and ninety variables, and showed that the solution time of the improved Balas (1965) algorithm can be reduced by in this way. Gavish and Pirkul (1985) concluded that the main drawbacks of Shih (1979)’s approach are its excessive space requirements, and its inability to solve problems with tight resource constraints, and they proposed a branch-and-bound procedure for MKP embedding new approximate algorithms for obtaining surrogate bounds and rules for reducing problem size, and showed that their method was significantly faster than Shih (1979)’s method by testing problems with size up to 80 variables and 7 constraints. Using an LP relaxation of the surrogate dual to avoid the solving of 0–1 knapsack problems and to reduce solution
times, they got good results without improving the LP bound. Freville and Plateau (1997) searched the use of integer surrogate relaxations for solving the bi-dimensional case, by designing an efficient preprocessing phase which completed with an enumerative phase if needed. Computational experiments with randomly generated and correlated instances up to 750 variables indicated that the procedure is as good as Gavish and Pirkul (1985) procedure and can provide a competitive alternative to LP-based strategies.

Many other special approaches have tried to solve the special structures of the MKP such as Cabot (1970) suggested an enumeration technique based on the Fourier-Motzkin elimination method, Thesen (1975) presented a recursive branch and bound algorithm, and Soyster et al. (1978) proposed an iterative scheme in which linear programs solved to generate subproblems which were solved using implicit enumeration. Sahni (1975) proposed approximation algorithms for the 0-1 knapsack problem. Plateau and Roucairol’s (1989) used a parallelization of tree search algorithm which includes searching for an initial feasible solution, reducing size based on the additivity of the reduced costs and using a terminal branch-and-bound procedure. Freville and Plateau (1993) designed specific procedures for the bi-dimensional 0–1 knapsack problem, which were able to find the optimal dual solution within a finite number of iterations, practically independent of the number of variables. Gabrel et al. (1999) solved multicommodity network optimization problems with general step cost functions via solving a subproblem at each iteration which can be converted into a MKP coupled with multiple choice constraints. Gabrel and Minoux (2002) developed a
procedure based on the solution of several knapsack subproblems to generate most violated extended cover inequalities.

Several other approaches, including the hybridization of dynamic programming and branch-and-bound (Plateau and Elkihel, 1985), the use of expanding core subproblem (Pisinger, 1995), upper bounds obtained by adding valid inequalities on the cardinality of an optimal solution constraint (Martello and Toth, 1997), the combination of a new dynamic programming recursion and an additional cardinality constraint (Martello et al. 1999, Pisinger 2000) dealt with the case of strongly correlated instances which remained very difficult to solve, and can effectively solve large problems of these type as well as other hard classes. Furthermore, several effective special-purpose codes are also available such as given by Fayard and Plateau (1982).

Schilling (1990) presented an asymptotic analysis of the MKP, and computed the asymptotic objective function value where the resource and profit variables were uniformly distributed over the unit interval and the knapsack capacity was equal to one, and Szkatula (1994) generalized that analysis where the knapsack capacities were not restricted to be one. Fontanari (1995) conducted a statistical analysis of the MKP, and examined the dependence of the objective function on the knapsack capacities and the number of capacity constraints.

Beaujon et al. (2001) proposed a MIP formulation taking the form of a MKP with other generalized constraints designed to select projects for inclusion in a R&D portfolio. For solving mixed-integer programming problems, constraint programming techniques integrated into integer programming are also used.
Although various algorithms were developed to provide good lower and upper bounds; due to NP completeness, these exact methods mostly based on some sort of branch and bound and the commercial solvers such as CPLEX, XPRESS, LINDO, OSL, can only solve small and medium size instances optimally. As such, to solve large size instances of MKPs, various heuristics and meta-heuristic methods, such as genetic algorithms, simulated annealing, tabu search and scatter search to name a few.

1.2.2 Heuristics

Senju and Toyoda (1968) proposed a dual heuristic for the MKP which starts with assigning ones to all variables and setting the variables to zero one at a time according to increasing ratios until feasibility requirements are satisfied. By contrast, Kochenberger et al. (1974), Toyoda (1975), Loulou and Michaelides (1979) formulated some methods for the MKP which started from the origin and set variables to one according to decreasing ratios until no more variables can be added without violating the constraints. Hillier (1969) introduced multistage algorithms and interior paths for the MKP which focused on the simplex composed of the optimal LP solution and its adjacent extreme points as a point of departure for a line search. The first phase identifies a path leading from the optimal LP solution to another nearby solution belonging to the integer feasible region, then the algorithm moves along this path to identify a better feasible integer solution in the second phase, finally, a local search which attempts to improve the current feasible solution by changing one or two variables at a time is done in the last step. Using MKP instances of moderate size,
Zanakis (1977) showed that Hillier (1969)’s algorithm was more accurate than basic primal/dual greedy algorithms. One of the most well-known LP-based procedures for finding approximate solutions to general linear 0–1 programs is Pivot and Complement which was developed by Balas and Martin (1980). They proposed an approximate algorithm for the MKP for solving the core problem, a knapsack problem defined on a small subset of the available items, such that there is a high probability for finding a global optimum within the core, showing that the probability for the heuristic to find an optimal solution increases with the size of the instance. The procedure begins with solving the LP relaxation with a standard bounded variable simplex method and continues by executing a sequence of pivots aimed at putting bounded variables into the basis at a minimal cost, then a complementing phase attempts to improve the 0–1 solution obtained in the pivoting. Furthermore, promising results have been also obtained for pure 0–1 linear programs by hybrids of tabu search with the pivot-and-complement heuristic. Freville and Plateau (1994) proposed an efficient preprocessing algorithm for the MKP, which gives sharp lower and upper bounds on the optimal value by reducing the continuous feasible set and by eliminating constraints and variables.

For solving the MKP, Magazine and Oguz (1984) combined the Senju and Toyoda’s dual algorithm with a Lagrangean relaxation approach, which allows fixing variables to their values assigned in all optimal solutions, then their work has been extended by Volgenant and Zoon (1990). Freville and Plateau (1986) worked on Lagrangean and surrogate relaxations, and proposed three solution methods using surrogate constraints, accelerated fixing (more than one variable fixed at a time),
noising strategy and strongly determined variables. Pirkul (1987) constructed a more straightforward generic approach to solve the MKP embedding a descent procedure to determine the surrogate constraints, and proved that this greedy procedure was generally faster than pivot-and-complement heuristic and generated solutions were similar in terms of solution quality with instances up to 200 variables and 20 constraints. Lee and Guignard (1988) proposed a multistage procedure to solve the MKP tuned with a few parameters which control the tradeoff between solution quality and computation times, whose values are set by the users, and they reported solution quality and computation time improvements via numerical results for 48 test problems with 5–20 constraints and 6-500 variables. Hanafi et al. (1996) established a simple multistage algorithm for the MKP which incorporates different heuristic principles, such as greedy, simulated annealing, threshold accepting, noising, in a flexible fashion. Starting from a set of random feasible solutions, the first stage performed different local searches, and then an additional stage based on repeated greedy steps tries to improve the current feasible solution. Balas et al. (2001) showed a sophisticated local search in the integer neighborhood of the fractional LP-solution for solving pure 0–1 programs.

Martello and Toth (1988) developed an effective algorithm for large-size problems, which is based on the use of a greedy algorithm for solving large knapsack problems and solves the core problem to optimality through branch-and-bound thus obtaining a better lower bound. Horowitz et al. (1994) showed a simple algorithm solving the knapsack problem using a greedy method.
Plateau et al. (2002) examined a multistage method using metaheuristics and interior point methods, where the first phase includes a hybrid search that uses an interior point method to generate fractional germ points, a local search to bring back feasibility, and a cut generator to diversify the population of initial feasible solutions, and the second phase does a fixed number of path re-linking runs between a set of solution pairs selected from the initial population. They solved MKP and made comparisons with Chu and Beasley (1998)'s genetic algorithm which showed promising prospects for using interior point methods as a guide to enhanced local searches, path re-linking or scatter search approaches. Balev et al. (2008) proposed a heuristic that uses dynamic programming in a suitable way to get a feasible solution by successive improvements of the LP-rounding solution, and tested it on all standard sets of the literature. Their heuristic is shown to be robust and very fast compared with the best tabu search approaches.

Frieze and Clarke (1984) proposed a polynomial approximation scheme based on the use of the dual simplex algorithm for linear programming, and examined the asymptotic properties of a particular random model. Rinnooy Kan et al. (1993) presented a class of generalized greedy algorithms in which items are chosen according to decreasing ratios of their profit and a weighted sum of their resource coefficients. Averbakh (1994) examined the properties of several dual characteristics of the MKP for different probabilistic models.

Fox and Scudder (1985) proposed a heuristic based on starting from setting all variables to zero (one) and successively choosing variables to set to one (zero), and
presented computational results for randomly generated test problems up to 100 variables and 100 constraints for the MKP.

1.2.3 Metaheuristics


Drexel (1988) developed a simulated annealing approach, and proposed a special 2-exchange random move which maintains the feasibility of all solutions generated during the process. Dueck and Scheuer (1990) showed deterministic version of SA, called threshold accepting, with slightly better results than Drexel (1988)'s approach for the MKP.

Dammeyer and Voss (1993) introduced a tabu search for solving the MKP by through a dynamic version of TS, called Reverse Elimination Method, in which feasibility is maintained along the process by using a multivariate DROP/ADD move. Battiti and Tecchiolli (1994) examined a tabu list dynamic management, called Reactive Tabu Search, and got satisfactory performances for the MKP. To solve the MKP, Lokketangen and Glover (1996) established a direct approach by making TS rely on a standard bounded variable simplex method as a subroutine. Glover and Kochenberger (1996)
worked on the use of tunneling effect, and presented a strategic oscillation scheme which alternates between constructive and destructive phases of TS and drives the search to variable depths on both sides of the feasibility boundary, and had high quality computational results over several large MKP test problems with up to 500 variables and 25 constraints. Hanafi and Freville (1998) established a TS approach which combines strategic oscillation with generalized greedy algorithms guided by surrogate constraints information and the state of the search, and got competitive results with those of Glover and Kochenberger (1996) for the MKP. Hanafi and Freville (2001) gave some extensions concerning the link between new dynamic rules and diversification and intensification strategies.

Of the size of the problems solved, Vasquez and Vimont (2005) had best quality solutions for benchmarks from the literature, but the computational times are rather high for solving very large instances. Dammeyer and Voss (1993) proposed a tabu search heuristic based on reverse elimination, and presented computational results for 57 standard MKP test problems from the literature where they found optimal solution for 41 of them. Aboudi and Jornsten (1994) combined tabu search with the pivot and complement heuristic of Balas and Martin (1980) and presented computational results for 57 standard MKP test problems from the literature where they found optimal solution for 49 of them. Lokketangen and Glover (1998) proposed a tabu search heuristic designed to solve general zero-one mixed integer programming problems, and tested their approach on 57 standard MKP problems from the literature and had optimal solutions for 54 of these problems.
Evolutionary algorithms are one of the important streams of metaheuristics. Michalewicz (1996) proposed a GA which represents the penalty function approach to the knapsack problem.

Chu and Beasley (1998) provided the most complete coverage of the GA for the MKP. They proposed the first successful implementation of GA’s by restricting the GAs to search only the feasible search space. They provided a description and review of the MKP and the GA, as well as a set of 270 test problems which were then made available via the web. Their algorithm includes a heuristic operator which guarantees that the child solutions can be made feasible. They compared GA performance to a branch-and-bound algorithm and to other heuristic methods and found that the GA performance was quite good. Numerical comparisons demonstrated the robust behavior of the Chu and Beasley method for obtaining high quality solutions within a reasonable amount of computational time. On a large set of randomly generated problems, they showed that the GA heuristic was capable of obtaining high-quality solutions for problems of various characteristics in a modest amount of computational time. Raidl (1998) developed an improved GA for the MKP by introducing a pre-optimized initial population, a randomized repair operator based on the values of the primal variables of the relaxed linear programming solution, and a local improvement operator which are based on the solution of the LP-relaxed MKP. They tested the improved GA against the Chu and Beasley test set and showed that, most of the time, it converged much faster to slightly better solutions. Haul and Voss (1998) improved the performance of GA’s by using surrogate constraints. Gotlieb (2000) presented new initialization routines and
compared several repair and optimization methods. There are also other evolutionary algorithms proposed for solving MKPs in the literature.

The parallelization of metaheuristics has also drawn much attention for tackling very large-scale instances. Davis (1991) proposed an introduction to hybrid GAs which are GAs that are integrated with competing algorithms or heuristics. Thiel and Voss (1994) investigated GAs to solve MKPs, using a direct search in the complete search space. They showed that a GA with standard operators is not able to obtain good solutions for large problems. Then they integrated simple heuristic operators based on improvement ideas from local search to improve GA. They also combined GA with tabu search, Hybrid-GA, and got promising results for moderate size test problems. Raidl (1999) investigates weight coding in a GA, combined with several relaxation-based decoding heuristics.

Ohlssen et al. (1993) initiated the use of neural networks to solve the MKP. The numerical experiments showed that due to the strategic choice of a penalty function which transforms the MKP into an unconstrained problem, neural network tends to produce final solutions that violate constraints. Hembecker et al. (2007) applied particle swarm optimization for solving MKPs.
CHAPTER 2: GENETIC ALGORITHMS

Of all the algorithms for the solution of the multidimensional 0-1 knapsack problem (MKP), the genetic algorithm (GA) seems to be one of the best in terms of solution quality and computation time, and is the focus of this study. In this chapter, an introduction of GAs is given.

2.1 Genetic Algorithms

Stochastic optimization techniques like evolutionary algorithms, simulated annealing etc., which rely heavily on computational power, have been developed and used for optimization. Among these, evolutionary algorithms, which are randomized search techniques aimed at simulating the natural evolution of asexual species, are found to be very promising global optimizers. GAs are perhaps the most popular evolutionary algorithms (Back and Schwefel, 1993).

GAs were developed by John Holland (1975) and his colleagues at the University of Michigan in the 1970s as a stochastic search technique based on the mechanism of natural selection and recombination. After the publication of Goldberg’s book (1989) that proposed the answer to why the application of GAs to a special problem can lead to good solutions, GAs have attracted large attention in several fields as a methodology for optimization, and been applied in a variety of continuous, discrete and combinatorial

A GA is a population-based search-and-optimization technique that is based on the evolutionary process of biological organisms in nature. It mimics natural evolution, in particular Darwin’s idea of the survival of the fittest. During the evolution process, natural populations evolve according to the principles of natural selection and “survival of the fittest”. In natural selection process, stronger members of a population pass genes corresponding to more desirable characteristics to subsequent generations through a reproduction process. In a population, individuals that are more successful in adapting to their environment will have a higher chance of surviving and reproducing, while individuals that are less fit will be eliminated. In this way, species evolve to become increasingly better in adapting to their environment.

In the solution of optimization problems, GAs imitate the evolutionary process by processing a population of solutions simultaneously. In a GA, problem solutions are presented as chromosomes, in which the genes show if a characteristic exists or not, and the set of solutions under consideration form the finite population, on which GA operates. GAs manipulate bit strings or chromosomes encoding useful information about the problem, and use the evaluation of a chromosome, as returned by the fitness
function, to guide the search. The objective function value is used as the fitness of each member of the population. Parents are selected from existing chromosomes for reproduction. Genes from each parent are combined according to some predefined strategy to produce offspring which create subsequent populations. These new offspring replace the nonselected chromosomes in the population. Basically, the search mechanism consists of three different stages: evaluation of the fitness of each chromosome, selection of the parent chromosomes, and application of the recombination and mutation operators to the parent chromosomes. The new chromosomes resulting from these operations create the next generation, and the process is repeated for some predefined number of iterations or until a predetermined stopping criterion is reached.

The basic steps of a simple GA are shown below (Chu and Beasley, 1998):

*Generate an initial population;*

*Evaluate fitness of individuals in the population;*

*repeat*

*Select parents from the population;*

*Recombine (mate) parents to produce children;*

*Evaluate fitness of the children;*

*Replace some or all of the population by the children;*

*until a satisfactory solution has been found;*
In other words, a GA starts with randomly generated solutions and better solutions are picked for recombination with others in order to create new solutions. Fitter solutions are more likely to pass their information to future generations of solutions, just like the nature picks the healthiest and fittest offspring. New solutions inherit good parts from old solutions; as a result, this repetitive process over many generations yields a population containing the best or the optimal solution.

2.2 Genetic Algorithms Applied to Knapsack Problems

The performance of a GA depends on the design of various features, the most important of which are population size, number of generations, parent selection method, reproduction strategy, and mutation rate.

2.2.1 Problem Representation

One of the most important aspects for a successful implementation of a GA is the representation of an underlying problem by a suitable scheme. Problem representation must contain useful substrings that allow for recombination in a meaningful way. There are different methods to represent solutions as a string of digits of fixed length, and the traditional representation scheme is the binary encoding, where a gene in a chromosome receives a value of 0 or 1. In addition, some other representations are also possible like strings of integers or rules. For the MKP, a solution can be simply represented by a vector of 0 and 1s, where 0 represents that an item is not being selected, 1 represents that an item is being selected.
2.2.2 Population

In a GA, the search space of the problem at hand is represented as a collection of individuals, which are often referred to as chromosomes. A GA begins with a randomly or heuristically generated population of chromosomes representing possible solutions to the problem. Each element of the population is called a chromosome, which is a combination of symbols known as genes.

There are a variety of approaches to generating initial populations for a GA. The most common method of population generation is random generation that is easy to implement and seems to provide a diverse population covering the feasible region. (Thiel and Voss, 1994). In random generation, each gene of a chromosome gets a value of one with probability $p$ and a value of zero with probability $1-p$, usually $p = 1-p = 0.5$. However this approach could have two potential drawbacks: a) there is no guarantee that the initial population contains feasible solutions and b) the initial population could be of inferior quality. If starting either with an infeasible population or a poor quality population, many subsequent generations may be required for a GA to develop feasible solutions, resulting in a slow convergence.

To prevent infeasibility, it is common to use a fitness function that penalizes infeasibility, where infeasible strings are penalized depending on their distance to feasibility to help quickly converge to feasible solutions. Yet another approach is to use repair operations, which are some systematic approaches to return an infeasible solution to feasible, such as in a MKP problem, removing items from a knapsack via changing the genes’ values from one to zero until feasibility is obtained. Following repair
operations, reoptimization operators can be used to improve these repaired solutions ensuring the solution is feasible.

In an effort to improve solution quality of a population, heuristics can also be used. Different from random generation, the initial population of a GA may consist of feasible strings generated by a simple heuristic. However, there is a potential danger that heuristic procedures would be biased and would not provide a comprehensive representation of all possible genes; this could lead to premature convergence (Reeves, 1993). Therefore, it is crucial that such a heuristic should create several different solutions to maintain diversity in a GA. A population which has both good and diverse solutions provide quicker convergence of the GA, and it reduces the computational efforts significantly particularly for large problems.

As will be seen in this study, instead of random generation, using the problem specific knowledge, a systematic approach that uses a method to generate alternatives is proposed. This approach generates an initial population with diversity and high quality, and has significantly improves the performance of the GA.

In most GA implementations, the population size seems to be arbitrarily chosen, though some implementations set the size to be dependent in some way on the nature and size of the problem. Small populations have the risk of failing to cover the solution space adequately, whereas large populations may incur a heavy computational load without making enough progress towards a high quality solution in a reasonable amount of time. Related to this tradeoff, early studies suggested that optimal solution sizes that grew exponentially with the length of the string, whereas later studies have concluded
that populations of this size are not needed. For example, many implementations seem
to produce satisfactory results with populations having as few as 30 strings, although
values of 50 or 100 are more common, and small populations are adequate at least for
binary coding (Reeves, 1993).

2.2.3 Parent Selection

In a GA, each chromosome in the population is associated with a fitness value to
evaluate the quality of the solution, and to determine which chromosomes are used to
form new ones in the competition process called selection, and whether a chromosome
survives for the next generation or not. Selection is achieved by favoring fitter
chromosomes so as to ensure that good properties are carried forward to the next
generation. The fitness of an individual is evaluated by the fitness function with respect
to a given objective function.

Various selection schemes can be used, such as roulette-wheel and tournament
selection. The roulette-wheel method simply generates a probability distribution, via
allocating weights to each solution, in which the selection probability of a solution is
proportional to its fitness. The roulette principle is basically a process by which a good
parent with a higher fitness value is assigned a higher selection probability than a bad
parent. There is a section on the roulette-wheel for each parent, and the size of the
section is proportional to the ratio of the parent's fitness to the total fitness of the
population. However, this selection has also some drawbacks. For example, if there is a
chromosome with a very high fitness value, it will be selected at almost each trial and
will quickly dominate the population. Therefore, the population does not evolve further, because all its members are similar, and premature convergence is occurred. The other approach is tournament selection, in which a set of $T$ chromosomes is chosen and compared, and the best one is selected. When $T=2$, this approach is called a binary tournament selection.

### 2.2.4 Crossover and Mutation

In a GA, three genetic operators known as reproduction, crossover and mutation are applied to the parents in the population to generate new individuals, called offspring.

Crossover combines two chromosomes (parents) to produce new chromosomes (children), which shares some characteristics taken from both parents. The offspring are created by exchanging information among strings of the parents. In other words, crossover is a matter of replacing some of the genes in one parent by the corresponding genes of the other. In general, the crossover operator can be described as follows. Randomly choose a position $k$ of the two strings of length $l$, where $k \in \{1,\ldots,l-1\}$, and create two new strings by exchanging all bits from position $k+1$ up to position $l$.

One-point crossover and uniform crossover operators are the most common operators used in GAs (Holland 1975, Goldberg 1989). One-point crossover exchanges the bit strings at a randomly selected crossover point in two parent chromosomes and generates two offspring. There are various extensions to the one-point crossover such as two-point crossover, multipoint crossover, and uniform crossover. The two-point crossover randomly selects two cut points on both parent chromosomes, and exchanges
the substring located between these two cut points. In multipoint crossover \( m \) crossover points are chosen randomly. It can be further generalized by making \( m \) a random variable, or simply copying a given gene from the first parent with probability \( p \) and from the second parent with probability \( 1-p \). Uniform crossover exchanges each bit in two given parents with a probability of 0.5 and generates two offspring. More general extensions for crossover are also possible with increasing difficulties for maintaining feasibility in a general search space.

The other genetic operator is mutation in which a gene or a subset of genes in a chromosome is chosen randomly and the bit value of the chosen genes are changed; in the case of binary strings, mutation causes the change from 0 to 1 or from 1 to 0. In other words, mutation is used for achieving a local change on the current solution. The goal of the mutation operator is to introduce random perturbations into the search process. By altering the genes, a GA is able to maintain adequate diversity, which is necessary for effective search, in the population of chromosomes to avoid premature convergence. Mutation is useful to introduce diversity in homogeneous populations, and to restore gene values that cannot be got back through crossover; for example, when the gene value at a given position is the same for every chromosome in the population. The mutation operator is necessary in a GA to protect it against premature loss of information through crossover.

The traditional mutation scheme consecutively processes each gene in an offspring starting from the first bit and switches its value with a given mutation probability. The probability with which each gene is to be mutated is called the
mutation rate, which can be changed for different genes in the string or for different generations. In general, the mutation rate is usually fairly low, because with a high mutation rate, a GA becomes an unguided random search; however, in order to maintain an acceptable level of diversity in the population, the mutation probability could be increased as the search progresses.

It is known that the success of a GA depends on the ability to set a proper balance between exploration and exploitation. To achieve such a balance and, as a consequence, effective search performance, a set of GA parameters such as population size, crossover probability, and mutation rate should be tuned properly.

2.2.5 Replacement
Following the crossover and mutation operators, the chromosomes are selected from the current population to survive to the next generation. The search is guided by the results of evaluating the objective function for each string in the population, and the strings that have higher fitness values replace some or all of the current population, depending on the selection scheme, and construct a new population.

In the most general approach, the whole population is replaced by the offspring, a new population, at each generation. In steady-state approach, in which only one new chromosome (or sometimes a pair) is generated at each iteration, the offspring replace less fit individuals. Basically, in replacement phase, some of the best solutions of each generation are kept while the others are replaced by the newly formed solutions. The
evolutionary process is repeated until some stopping criteria are met, e.g. until a satisfactory and acceptable solution is found or until the optimal solution is found.

2.3 Scatter Search and Path Re-linking

Recently, there has been another evolutionary approach called Scatter Search, and its generalized form called Path Re-linking (PR), that have proved unusually effective for solving a diverse array of optimization problems from both classical and real world settings.

PR was originally proposed by Glover and Laguna (1993, 1997) as an evolutionary method to combine elite solutions for integration of intensification and diversification. Instead of directly producing a new solution when combining two or more solutions as it is performed in GAs, PR generates paths between and beyond the selected solutions in the neighborhood space by adding, dropping or modifying solution attributes by the moves executed. This approach generates new solutions by exploring trajectories that connect high-quality solutions starting from one of the solutions, called an initiating solution, and generating a path in the neighborhood space that leads toward the other solution, called guiding solution. This process is achieved by selecting moves that introduce attributes contained in the guiding solutions.

PR usually starts from a given set of elite solutions and considers a combination of solutions in the solution neighborhood space via generating paths between and beyond the elite solutions. The simplest PR approach includes an initiating solution and a guiding solution, which are the endpoints of a path. The path starts from the initiating
solution and then the moves progressively introduces attributes from the guiding solution to reduce the distance between attributes of the initiating and guiding solutions. The roles of the initiating and guiding solutions are interchangeable, and also each solution can be caused to move simultaneously toward the other. Choosing best, worst or average moves provides options that generate different sequences. In general, it appears reasonable to select best moves at each step, and then to reinitiate the process in the opposite direction by interchanging initiating and guiding solutions. There are many research topics associated with the PR strategy, such as the selection of elite solutions, how to guide the initiating solution to guiding solution, which path is the best to find a better solution among the possible alternatives, using multiple guiding solutions, allowing for intermediate infeasible solutions.

Several variations of PR, such as simultaneous re-linking, extrapolated re-linking, tunneling, and multiple guiding solutions exist (Marti et al., 2006).

Simultaneous Re-linking: This approach starts with both endpoints, initiating and guiding solutions, simultaneously producing two sequences which finally converge to a single point.

Extrapolated Re-linking: The PR approach goes beyond consideration of points between initiating and guiding solutions. The ability to go beyond the endpoints, initiating and guiding solutions, creates a form of diversification.

Tunneling: Strategic oscillation is a mechanism used in tabu search to allow the search process to visit solutions around the feasibility boundary, by crossing the boundary from the feasible side to the infeasible side and also from the infeasible side
to the feasible side. PR also allows the search to cross the feasibility boundary by way of a tunneling strategy, which permits infeasible solutions to be visited while re-linking initiating and guiding solutions. The tunneling effect therefore gives a chance to reach solutions that might otherwise be bypassed. As a result, intermediate solutions generated by PR do not need to be feasible in order to be relevant as a starting solution for an improvement procedure.

Multiple Guiding Solutions: New points can be generated from multiple guiding solutions. Instead of moving from an initiating solution to a guiding solution, guiding solution is replaced by a collection of guiding solutions.

Laguna et al. (1999) used PR for search intensification in elite areas for the linear ordering problem. Glover (1999) proposed that selecting an unattractive move to generate the path at each step tends to produce high quality improving final moves, whereas choosing attractive moves at each step tends to produce low quality final moves. Zhang and Lai (2006) proposed two-solution based PR and its combination with GA. They investigated two integration approaches; parallel connection of GA and PR, which uses PR during GA evolution, and series connection of GA and PR, which uses PR between two generations of GA. When the GA solution is trapped in a local solution and does not improve for a specific number of generations, PR is executed to escape from the local optimization. They investigated probability-based selection method; in the parallel connection, both guiding and initiating solutions are randomly selected based on the possibility distribution of fitness, in the series connection the initiating solution is randomly selected whereas the guiding solution is one of the best solutions.
CHAPTER 3: AN IMPROVED GENETIC ALGORITHM

In this study, an improved genetic algorithm (GA) is proposed for the solution of the multidimensional 0-1 knapsack problem (MKP). The algorithm differs from other GA approaches in that 1) it incorporates a systematic approach to generate high quality diverse populations for the GA, and 2) it explores the use of path re-linking (PR) as crossover operators in the GA to explore trajectories that connect high quality solutions.

Before presenting the proposed algorithm, let us first look the state-of-the-art GA developed by Beasley and Chu (1998). The algorithm starts with a randomly generated population, select parents through a tournament selection with T=2, evolves through a uniform crossover operator, uses a 2% mutation rate, employs a repair operator, and a steady state replacement strategy. The algorithm stops when a total of $10^6$ non-duplicate individuals have been generated. Though there have been several algorithms proposed for the solution of knapsack problems, this algorithm still remains one of the most competitive algorithms and thus is used as the baseline for this study and comparisons. Our algorithm differs from Beasley and Chu’s algorithm mainly in two aspects; a) population generation, and b) crossover operators design.
3.1 A Systematic Approach to Generate Alternatives for Initial Population of a GA

The initial population of a GA is critical to the overall convergence and overall quality of solutions. In the literature, two approaches are generally employed; the first one is random generation while the second one is heuristics.

A random generation method, for example, could randomly select an unselected item. If the inclusion of the item does not violate the capacity of the constraints, include it in the knapsack; otherwise, continue until there is no item that can be included in the knapsack. Random generation usually suffers from initial population could be of inferior quality; thus requires many subsequent generations to converge to high quality solutions. To improve solution quality of a population, a heuristic can also be used.

A heuristic based approach, for example, could select an item to be included based on some myopic greedy measures and iteratively select the items, rather than purely randomly. However, randomization is still important to create diversity among populations which permits search across the solution space; hence, there is a potential danger that heuristic procedures would be biased and would not provide a comprehensive representation of all possible genes, this could lead to premature convergence (Reeves, 1993). Therefore, it is crucial that a systematic procedure must be designed that is able to create several different solutions to maintain diversity in a GA, yet retain high quality solutions. A population which has both good and diverse solutions could provide quicker convergence of the GA, and reduce the computational effort significantly, particularly for large problems.
In an effort to achieve this goal, a systematic approach has to be developed that provides an efficient algorithm to obtain high-quality diverse solutions for the MKP. The generation of multiple alternative solutions, however, is not an easy task. Typical approaches using linear programming techniques to search for alternative solutions in the final simplex tableau, or changing coefficients in the objective functions, are computationally prohibitive and do not guarantee sufficiently diverse solutions.

The following procedure generates not only alternative solutions, but also multiple alternative solutions of maximal differences (Brill et al. 1990, Gu et al. 2008). In a nut shell, for an integer program defined by following Equations (1) – (3):

Maximize  \( z = cx \)  \hspace{1cm} (1)
Subject to  \( Ax = b \)  \hspace{1cm} (2)
where  \( x_i \) is integer  \hspace{1cm} (3)

The procedure can be summarized as follows:

**Step 1**: Obtain an initial solution using a linear programming optimization; denote the optimal solution value as \( \psi^* \).

**Step 2**: To generate alternative solutions, solve the following MGA problem:

Minimize  \( \sum_{k \in K} x_k \)
Subject to  Eqs. (1) – (3)
\[ cx \geq \psi^* - \theta \psi^* \]  \hspace{1cm} (4)

where \( \theta \) is a small relaxation ratio. \( K \) is the set of indices of the variables that are nonzero in the initial set of solution or generated during the process thus far.
**Step 3:** Stop until no new variables appear in the optimization problem or the number of solutions generated exceeds the size of the population.

In the above procedure, Step 1 solves the original linear program and sets the target goal for following solutions. Step 2 produces an alternative solution that is sufficiently different from previous solutions by minimizing the sum of decision variables that are nonzero in the previous solutions. Constraints $cx \geq y^* - \theta y^*$ ensure that the alternative solution is close with respect to the modeled objective. If $\theta$ is set to 0, the solutions will have the same objective as in Step 1. The above procedure involves iteratively solving a series of linear programs until no new variables appear or a fixed number of iterations have reached as show in Step 3.

An example of the above procedure for a small optimization problem is shown below to illustrate how the algorithm works, and its quality of the solutions and the coverage of the solution space. The small optimization problem whose optimal solutions can be enumerated is used and these optimal solutions are compared with those produced from the procedure to generate alternative solutions.

*A Sample Optimization Problem:* The problem is to find the minimum number of workers to satisfy the demand in a day. Here, a day is divided into six four-hour periods and a worker starts at the beginning of a time period and works two periods. For a demand of 4, 8, 10, 7, 12, 4 for the six periods, the problem can be stated as follows:

Minimize $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$

Subject to $x_6 + x_1 \geq 4; x_1 + x_2 \geq 8; x_2 + x_3 \geq 10; x_3 + x_4 \geq 7; x_4 + x_5 \geq 12; x_5 + x_6 \geq 4$

where $x_i$ integers
$x_i$ are the decision variables representing the number of workers that start their shifts at the beginning of time period $i$. The following table lists the complete set of optimal solutions to the sample problem and the alternative solutions, marked in bold, generated from our procedure defined above. As can be seen, there are 10 optimal solutions for this integer program (because of the cyclic structure, the linear programming solutions to this problem are all integers); these solutions can be divided into clusters (the solutions in each cluster only differ slightly). The MGA procedure generates solutions in most clusters and provides a comprehensive sample of the solution space.

Table 1: Complete set of optimal solutions to the sample problem

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Solution</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>2</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>11</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>26</td>
</tr>
</tbody>
</table>

As it can be seen, upon completion, the above procedure provides a set of linear programming solutions that are of “maximal distance” from each other and are of high quality that could be used for a GA population.
For multiple knapsack problems, these linear programming solutions provide a good starting point for the knapsack problem due to the following theorems.

**Theorem 1:** For a MKP with \( m \) constraints and \( n \) variables, there are at most \( m \) variables that will appear fractional in the linear programming solutions.

To show this, first the \( m \) inequality constraints are transformed into \( m \) equality constraints by adding \( m \) 0-1 slack variables. From linear programming theory, the optimal basic feasible solution will have at most \( m \) basic variables, which could be fractional. As such, there are at most \( m \) fractional solution components in the linear programming optimal solution if a standard simplex method is used. This completes our proof.

**Theorem 2:** For the optimization problem to find alternative solutions of maximal difference, there are at most \( m+1 \) fractional components in the final solutions. The optimal problem to find alternative solutions of maximal difference has one more constraint, the objective cut, \( cx \geq \psi^* - \theta \psi^* \) and has \( m+1 \) constraints. The proof is similar to that of Theorem 1 and is not presented.

From Theorem 1 and 2, it is known that the linear programming solutions contain at most \( m+1 \) fractional components. Therefore, the resulting integer program after fixing the variable contains at most \( m \) by \( m \) constraints. As \( n \) is typically much larger than \( m \), this has dramatically reduced the size of the problem.
These variables that appear either as 1 or 0 are then fixed at their corresponding values and the remaining variables are randomly selected until no variables can be selected without exceeding the capacity. This forms one of the individuals of the population. Then two parents are randomly selected and a tournament with $T=2$ is utilized to choose the two individuals as parents, and crossover operations are performed.

3.2 Parent Selection, Crossover, Mutation and Path Re-linking for the Improved GA

While one-point crossover and uniform crossover are widely used in the literature, PR is an alternative to crossover, providing another way of combining two solutions to generate new solutions by searching trajectories that link two elite solutions.

PR starts from one of these solutions, called the initiating solution, and generates a path in the neighborhood space that leads toward the other solution, called the guiding solution. In this path, the moves are selected that introduce attributes of the guiding solution. Therefore, PR evaluates more solutions on the path between two selected solutions, whereas uniform crossover just generates only one solution using these two solutions. While used in elite solutions, PR serves both as an intensification (between elite solutions) and diversification (diversify to other solutions from the path). Figure 1 shows an example, which creates a path between two selected solutions $x_1$ (initiating solution) and $x_2$ (guiding solution).
In PR, the initiating and guiding solutions are chosen via binary tournament selection. Then, the selected two solutions are compared according to their fitness value, and a path is linked from the solution with lower fitness to the other solution. There are, however, many paths from initiating solution to guiding solution. In this implementation, a randomized path is selected as follows.

At each step, an item, which is in the guiding solution but not in the initiating solution, is randomly selected to be in the solution. If the solution is feasible, another item is randomly selected and added to the solution; otherwise, an item which is not in the guiding solution but is in the initiating solution is dropped from the solution until reaching the feasible region. The process continues until the solution is the same as the guiding solution.

In this process, the solutions between two selected solutions are searched by moving between feasible and infeasible regions – similar to the strategic oscillation used in Glover and Laguna (1993) in their tabu search procedure for MKP. Because the optimum lie on the boundary between feasible and infeasible region of the search
space, strategic oscillations of small depths allow the search to cross boundaries and to concentrate on promising regions.

An improvement operation (repair operation) is further applied to the best solution, which has the highest fitness value generated during the path trace. Since two consecutive solutions obtained by a re-linking step are generally very similar, it is not necessary to apply an improvement method at every step of the re-linking process.

The best solution from the PR process then serves as the child generated from the two parents. If the child is not identical in both fitness value and genes to other individuals from the population, it is then selected to replace the worse solution (the least fit individual from the population) in the steady-state replacement. For comparison purpose, the algorithm stops when a total of $10^6$ non-duplicate individuals are obtained.

In summary, the proposed GA can be summarized as follows: a) first, a procedure to generate alternatives is used to systematically generate initial populations; b) a tournament selection with $T=2$ is used to select parents; c) the algorithm employs both randomly uniform crossover and PR as the crossover operators. (The decision to use PR or uniform crossover as the crossover operator of the MGA-GA is discussed in Chapter 4 where the computational results and insights on these two crossover operator are presented); d) a dynamic mutation rate is employed where at the beginning a mutation rate of 2% is used; otherwise in the later stage, a mutation rate randomly selected between 5% or 10% is used; e) a steady-state non-duplicate replacement method is used; and f) finally, the algorithm terminates when a total of $10^6$ non-duplicate individuals are generated.
CHAPTER 4: COMPUTATIONAL RESULTS

In this chapter, the computational results for the proposed genetic algorithm (GA) are presented.

4.1 Experimental Data and Evaluation Criteria

A benchmark data set of 270 multidimensional knapsack problems (MKPs) was proposed in Chu and Beasley (1998) and was widely used in the literature for the testing of MKP algorithms. The problems are generated with \( n = 100, 250, 500 \) variables, \( m = 5, 10, 30 \) constraints, and tightness ratios \( \alpha = 0.25, 0.50, 0.75 \). This set of problems contains 27 different problem sets, each having 10 randomly generated instances, thus a total of 270 problems.

In these problem set, \( a_{ij} \) was drawn from discrete uniform generator \( U(0,1000) \) and the right hand side coefficients \( b_i, i \in \{1,...,m\} \), were set using \( b_i = \alpha \sum_{j=1}^{n} a_{ij} \) where \( \alpha \) is the tightness ratio, and \( \alpha = 0.25 \) for the first ten problems, \( \alpha = 0.5 \) for the next ten problems, and \( \alpha = 0.75 \) for the last ten problems. The objective function coefficients \( c_j, j \in \{1,...,n\} \), were correlated to \( a_{ij} \) and are generated as \( c_j = \sum_{i=1}^{m} a_{ij} / m + 500q_j \) where \( q_j \) is a real number drawn from the continuous uniform generator \( U(0,1) \). The XPRESS-MP is used as the solver. Our experiment shows that for large size problems, MKP continues to be a challenging problem for commercial ILP solvers.
Several experiments were conducted to test the performance of the proposed GA. All the code was written in MOSEL, the modeling language of XPRESS optimization, and the linear programming relaxation was solved using its embedded XPRESS-MP Solver. The computation was performed on a Pentium Dual Core Processor 3.0 GHz with 4G of RAM, however, no parallel functionality was used and thus only one CPU is used in the experiment.

4.2 MGA-GA vs. Original-GA Comparisons

In the first experiment, the convergence and solution quality of the GA with MGA population generation, referred to as MGA-GA, and Original-GA of Chu and Beasley (1998) with random population generation are compared.

Convergence: Table 2 reports the computational results for the two algorithms at $10^1$, $10^2$, $10^3$, $10^4$, $10^5$, $10^6$ GA iterations or steps. Here, column “Steps” represents the number of iterations, “Better”, “Equal” and “Worse” columns represent the number of cases the first algorithm (MGA-GA) is better than, equal to, or worse than the second algorithm (Original-GA). The “avg. ratio” column represents the average of the results obtained from the first algorithm divided by the results of the second algorithm. Finally, the “avg. gap (%)” represents the average gap between the two solutions.
The results show that MGA-GA converges faster than Original-GA. For $10^1$ and $10^2$ steps, MGA-GA gives better solutions than Original-GA for all 270 instances. MGA-GA solution is on average 15.2% and 9.7% better than that of the Original-GA. For $10^3$, $10^4$, $10^5$ steps, MGA-GA also gives better results than Original-GA in the number of better solutions and the average gap. The numbers of better, equal and worse solutions at these steps are 217, 37, 16 at $10^3$ steps, 132, 81, 57 at $10^4$ steps, and 158, 89, 23 at $10^5$ steps and the average gaps are 0.3%, 0.01%, and 0.03%. These results prove that MGA-GA was able to obtain or converge to high quality solutions much faster than that of the Original-GA. The primary reason is the systematic use of high quality solutions generated using the linear programming heuristic to generate the initial population, rather than a random population generation approach.

### Final Solution Quality

It was natural to ask whether the heuristic based approach with the alternative generation methods would obtain similar solutions as the random generation used in the Original-GA of Chu and Beasley (1998). Generally, heuristic generated population tends to be biased toward certain genes and would lead to
premature convergence. In order to prevent premature convergence, diverse solutions and a good coverage of solution landscape has to be provided in the population. To see whether the MGA provides a good coverage and avoid the premature convergence, in the following table, comparison results for both algorithm was reported when the algorithm terminates at $10^6$ steps. Here the “MGA-GA sol” and “Original-GA sol” column represent the solutions obtained from each algorithm and “MGA-GA times” and “Original-GA times” represent the total time used at $10^6$ steps.
### Table 3: MGA-GA results for $10^6$ steps

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>alpha</th>
<th>MGA-GA-sol</th>
<th>MGA-GA-time (s)</th>
<th>Original-GA-sol</th>
<th>Original-GA-time (s)</th>
<th>&quot;&gt;&quot;</th>
<th>&quot;=&quot;</th>
<th>&quot;&lt;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>21652</td>
<td>1437</td>
<td>21652</td>
<td>1437</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>41432</td>
<td>1373</td>
<td>41431</td>
<td>1373</td>
<td>1</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59199</td>
<td>1342</td>
<td>59199</td>
<td>1340</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>56867</td>
<td>3383</td>
<td>56890</td>
<td>3379</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>106653</td>
<td>3175</td>
<td>106652</td>
<td>3160</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>150440</td>
<td>2980</td>
<td>150442</td>
<td>2982</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>115457</td>
<td>6514</td>
<td>115463</td>
<td>6513</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>216145</td>
<td>6159</td>
<td>216163</td>
<td>6149</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302353</td>
<td>5952</td>
<td>302349</td>
<td>5896</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>summary</td>
<td></td>
<td></td>
<td>120150</td>
<td>2208</td>
<td>120151</td>
<td>2205</td>
<td>56</td>
<td>159</td>
<td>55</td>
</tr>
</tbody>
</table>

As can be seen, both MGA-GA and Original-GA obtain approximately same results, MGA-GA obtain better solution in 56 cases, equal solution in 159 cases, and worse solution in 55 cases. The average solution is 120150 for the MGA-GA, and 120151 for the Original-GA. Considering the random nature of the algorithms, the MGA
procedure provided a set of diverse solutions was on par with the solutions obtained from the Original-GA and does not exhibit any premature convergence. This result seems to suggest the systematic method to generate alternative solutions was able to yield a good coverage of the solution landscape and provides a better alternative to random population generation.

Solution Time: Table 4 provides the solution time averages for both algorithms. The “first” column shows the time when the algorithm first finds the overall best result, and the “total’ column shows the total amount of time to run that total number of GA iterations. The results indicate that, for all number of GA iterations, both Original-GA and MGA-GA requires approximately same amount of time.

<table>
<thead>
<tr>
<th>Time - averages</th>
<th>Original-GA</th>
<th>MGA-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steps</td>
<td>first</td>
<td>total</td>
</tr>
<tr>
<td>10^1</td>
<td>0.3219</td>
<td>0.1409</td>
</tr>
<tr>
<td>10^2</td>
<td>0.3016</td>
<td>0.3418</td>
</tr>
<tr>
<td>10^3</td>
<td>2.2141</td>
<td>2.4734</td>
</tr>
<tr>
<td>10^4</td>
<td>12.0319</td>
<td>23.6773</td>
</tr>
<tr>
<td>10^5</td>
<td>88.3495</td>
<td>226.3414</td>
</tr>
<tr>
<td>10^6</td>
<td>654.3372</td>
<td>2205.2466</td>
</tr>
</tbody>
</table>

Figures on the Solution Process: To vividly show the solution process of the two algorithms, Figure 2 shows comparison of the solution progress for the Original-GA and MGA-GA for a selected case with different number of constraints, variables, and
tightness ratios. Here the horizontal axis represents the number of steps while the vertical axis represents the best solution obtained at these steps. As the figure indicates, MGA-GA converges faster than Original-GA. For additional details, please see the figures in Appendix C.

![Figure 2: MGA-GA and Original-GA comparison for m=5, n=100, α=0.50](image)

4.3 Path Re-linking vs. Uniform Crossover Comparisons

The aim of this experiment is to see whether path re-linking (PR) would serve as a good crossover operator for a GA.

a) Original-GA-PR vs. Original-GA-UNIFORM

Table 5 reports the comparison results of Original-GA-Uniform and Original-GA-PR. The former is the original GA with the uniform crossover as the crossover operator, the later with PR as the crossover operator. The format of table is similar to Table 2 and is not elaborated.
As can be seen, the results indicate that using PR approach instead of uniform crossover and mutation does not seem to provide better results. Though the use of PR was able to find better solutions at the beginning of the search process, 212 better solutions at $10^3$ steps, its performance deteriorates dramatically and was only able to obtain 30 better solutions, 51 equal solutions, but 189 worse solutions at $10^5$ steps.

**b) MGA-GA-PR vs. MGA-GA-UNIFORM**

Table 6 reports the comparison results of MGA-GA-Uniform and MGA-GA-PR. The former uses MGA-GA framework with uniform crossover as the crossover operator; while the later uses MGA-GA framework, but with PR as the crossover operator.
The results indicate, similar to what observed previously, though the use of PR instead of uniform crossover as the crossover operator improves convergence, it does not improve overall result. When the algorithm is complete at for $10^6$ steps, MGA-GA-PR is slightly outperformed by MGA-GA-Uniform.

These comparisons indicated that replacing uniform crossover completely by PR does not improve the solution quality. While PR was able to improve solution at the beginning, randomization seems critical to remain diversity in the search process and to achieve overall best solutions. The comparisons for MGA-GA also indicated that PR gives better results when it is used to combine high quality solutions. In view of this, uniform crossover is still retained as the main crossover operator, but is supplemented with PR to improve convergence. Further, instead of applying PR with ordinary parents that are selected using binary tournament selection, PR is invoked only among elite parents, solutions that are within the top a% of the population.

Convergence: Table 7 provides the results for different elite ratios and repair operation ratios for $10^5$ steps for MGA-GA-PR/Uniform method, and is compared with MGA-GA-Uniform. MGA-GA-PR/Uniform method selects parents randomly as in the Original-GA. However, if both parents are in the top specific percent of the population, it uses PR with random add and drop feature rather than uniform crossover to intensify the search in an effort to find better solutions.
Table 7: MGA-GA-PR/Uniform and MGA-GA-Uniform comparison for $10^5$ steps

<table>
<thead>
<tr>
<th>elite ratio</th>
<th>repair operation</th>
<th>Better</th>
<th>Equal</th>
<th>Worse</th>
<th>avg. ratio</th>
<th>avg. gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>no</td>
<td>72</td>
<td>128</td>
<td>70</td>
<td>0.99997</td>
<td>-0.003</td>
</tr>
<tr>
<td>20%</td>
<td>no</td>
<td>82</td>
<td>116</td>
<td>72</td>
<td>1.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>30%</td>
<td>no</td>
<td>84</td>
<td>124</td>
<td>62</td>
<td>1.00006</td>
<td>0.006</td>
</tr>
<tr>
<td>40%</td>
<td>no</td>
<td>88</td>
<td>113</td>
<td>69</td>
<td>1.00005</td>
<td>0.005</td>
</tr>
<tr>
<td>50%</td>
<td>no</td>
<td>90</td>
<td>112</td>
<td>68</td>
<td>1.00002</td>
<td>0.002</td>
</tr>
<tr>
<td>60%</td>
<td>no</td>
<td>84</td>
<td>109</td>
<td>77</td>
<td>1.00002</td>
<td>0.002</td>
</tr>
<tr>
<td>70%</td>
<td>no</td>
<td>85</td>
<td>112</td>
<td>73</td>
<td>1.00004</td>
<td>0.004</td>
</tr>
<tr>
<td>80%</td>
<td>no</td>
<td>95</td>
<td>102</td>
<td>73</td>
<td>1.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>90%</td>
<td>no</td>
<td>93</td>
<td>112</td>
<td>65</td>
<td>1.00006</td>
<td>0.006</td>
</tr>
<tr>
<td>100%</td>
<td>no</td>
<td>90</td>
<td>106</td>
<td>74</td>
<td>1.00007</td>
<td>0.007</td>
</tr>
<tr>
<td>10%</td>
<td>every Steps</td>
<td>73</td>
<td>127</td>
<td>70</td>
<td>0.99997</td>
<td>-0.003</td>
</tr>
<tr>
<td>20%</td>
<td>every Steps</td>
<td>84</td>
<td>123</td>
<td>63</td>
<td>1.00004</td>
<td>0.004</td>
</tr>
</tbody>
</table>

The results indicate that replacing uniform crossover and mutation with PR for different elite ratios gives better convergence than MGA-GA-Uniform. Here the “elite ratio” column represents the elite percentage the parents were in the population.

**Final Solution Quality:** Table 8 presents the results when the algorithm is complete at $10^6$ iterations. The results show that MGA-GA-PR/Uniform gives 65 better, 168 equal, and 37 worse results without repair operation or 68 better, 155 equal, and 47 worse results with repair operation, than MGA-GA-Uniform. The average solution obtained was also slightly better than that of the MGA-GA-Uniform.
Table 8: MGA-GA-PR/Uniform and MGA-GA-Uniform comparison for $10^6$ steps

<table>
<thead>
<tr>
<th>elite ratio</th>
<th>repair operation</th>
<th>Better</th>
<th>Equal</th>
<th>Worse</th>
<th>avg. ratio</th>
<th>avg. gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>no</td>
<td>65</td>
<td>168</td>
<td>37</td>
<td>1.00003</td>
<td>0.003</td>
</tr>
<tr>
<td>10%</td>
<td>every Steps</td>
<td>68</td>
<td>155</td>
<td>47</td>
<td>1.00001</td>
<td>0.001</td>
</tr>
</tbody>
</table>

Table 9 shows the comparison results for MGA-GA-PR/Uniform and Original-GA at $10^6$ steps. The algorithm was able to get 64 better, 160 equal, and 46 worse solutions than the Original-GA.

Table 9: MGA-GA-PR/Uniform and Original-GA comparison for $10^6$ steps

<table>
<thead>
<tr>
<th>elite ratio</th>
<th>repair operation</th>
<th>Better</th>
<th>Equal</th>
<th>Worse</th>
<th>avg. ratio</th>
<th>avg. gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>no</td>
<td>64</td>
<td>160</td>
<td>46</td>
<td>1.00001</td>
<td>0.001</td>
</tr>
<tr>
<td>10%</td>
<td>every Steps</td>
<td>65</td>
<td>159</td>
<td>46</td>
<td>0.99999</td>
<td>-0.001</td>
</tr>
</tbody>
</table>

The detailed results of MGA-GA-PR/Uniform with 10% elite ratio and no repair operation for $10^1$, $10^2$, $10^3$, $10^4$, $10^5$, $10^6$ steps are shown in Table 10. For $10^6$ steps; MGA-GA-PR/Uniform yields 65 better and 37 worse results than MGA-GA-Uniform. The experiment shows that the solution is improved in 24% of cases. Average ratio of MGA-GA-PR/Uniform / MGA-GA-Uniform is 1.00003, which shows, on average, MGA-GA-PR/Uniform gives higher results than MGA-GA-Uniform. For $10^6$ steps; MGA-GA-PR/Uniform gives 64 better and 46 worse results than Original-GA, whereas MGA-GA-Uniform gives 56 better and 55 worse results than Original-GA which is presented in the previous sections. Average ratio of MGA-GA-PR/Uniform / Original-GA is 1.00001, which
shows, on average, MGA-GA-PR/Uniform gives higher results than Original-GA. The average value is greater than the one obtained with the Original-GA developed by Chu and Beasley (1998) within comparable CPU time requirements.

Table 10: MGA-GA-PR/Uniform and MGA-GA-Uniform and Original-GA comparisons

<table>
<thead>
<tr>
<th>Steps</th>
<th>MGA-GA-PR/Uniform (10% elite ratio) vs. MGA-GA-Uniform</th>
<th>MGA-GA-PR/Uniform (10% elite ratio) vs. Original-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Better</td>
<td>Equal</td>
</tr>
<tr>
<td>$10^1$</td>
<td>98</td>
<td>57</td>
</tr>
<tr>
<td>$10^2$</td>
<td>111</td>
<td>53</td>
</tr>
<tr>
<td>$10^3$</td>
<td>103</td>
<td>75</td>
</tr>
<tr>
<td>$10^4$</td>
<td>88</td>
<td>101</td>
</tr>
<tr>
<td>$10^5$</td>
<td>72</td>
<td>128</td>
</tr>
<tr>
<td>$10^6$</td>
<td>65</td>
<td>168</td>
</tr>
</tbody>
</table>

Computation Time: Finally, Table 11 presents average computational times for all three methods for $10^6$ steps. The results indicate that although for case 3 (m=5, n=500), case 6 (m=10, n=500), and case 9 (m=30, n=500) MGA-GA-PR/Uniform requires slightly more time, the difference is not significant. The proposed algorithm is computationally effective; it is able to get better solutions in a reasonable amount of time.
Table 11: MGA-GA-PR/Uniform, MGA-GA-Uniform, and Original-GA solution times

<table>
<thead>
<tr>
<th>Case</th>
<th>MGA-GA-PR/Uniform (10% elite ratio)</th>
<th>MGA-GA-Uniform</th>
<th>Original-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>first</td>
<td>total</td>
<td>first</td>
</tr>
<tr>
<td>1</td>
<td>18.40</td>
<td>580.72</td>
<td>33.74</td>
</tr>
<tr>
<td>2</td>
<td>563.18</td>
<td>2007.78</td>
<td>342.65</td>
</tr>
<tr>
<td>3</td>
<td>1553.68</td>
<td>5118.59</td>
<td>435.04</td>
</tr>
<tr>
<td>4</td>
<td>103.91</td>
<td>646.82</td>
<td>69.48</td>
</tr>
<tr>
<td>5</td>
<td>694.89</td>
<td>2047.57</td>
<td>404.07</td>
</tr>
<tr>
<td>6</td>
<td>2080.90</td>
<td>5385.20</td>
<td>643.14</td>
</tr>
<tr>
<td>7</td>
<td>279.63</td>
<td>1235.33</td>
<td>264.16</td>
</tr>
<tr>
<td>8</td>
<td>1002.66</td>
<td>3339.34</td>
<td>769.42</td>
</tr>
<tr>
<td>9</td>
<td>3007.91</td>
<td>7306.56</td>
<td>2489.02</td>
</tr>
<tr>
<td>avg.</td>
<td>1033.91</td>
<td>3074.21</td>
<td>605.64</td>
</tr>
</tbody>
</table>

Finally, Table 13 and Table 14 in Appendix D present detailed comparisons for MGA-GA-PR/Uniform (10% elite ratio) with MGA-GA-Uniform and Original-GA, respectively.

It is well known that for GA to converge it is necessary to seed the population with divergence to cover the solution space; otherwise, premature convergence would occur. Though a comprehensive test of coverage for large scale optimization problem is not feasible, this experiment suggests that MGA algorithm was able to get diverse solutions that cover the solution landscape.
CHAPTER 5: CONCLUSIONS

In this study, a genetic algorithm (GA) for solving multidimensional 0-1 knapsack problems (MKP) was investigated. This study has presented a novel GA approach to solving the MKP that incorporates a) a systematic approach to generate populations, and b) path re-linking (PR) as crossover operators.

First, a systematic approach to generate alternative solutions is used in GA population generation. The method provides an attractive approach compared to widely used random population generation and was able to provide high-quality solutions with diversity. This has dramatically reduced the computational effort required to obtain high quality solution, and accelerated GA convergence.

Second, to understand the effect of crossover operator on the performance of GA, PR is combined with uniform crossover as the crossover operator of the GA. The efficiency of solution for combination provided by PR and uniform crossover were examined and compared. Originally, GAs were founded on precise notions of crossover, however, combining elite solutions using PR approach improves the solution quality. In general, if properly combined with uniform crossover, this approach allows the integration of diversification and intensification to achieve better results.

Computational studies conducted on benchmark problems suggested that the proposed algorithm was able to quickly get good solutions while avoiding being trapped
in premature convergence and are on par with some of the state-of-the-art algorithms in the literature. The study demonstrated that the systematic method to explore heuristics to generate population generation with diversity could significantly influence the convergence of a GA to best solutions. Nevertheless, as our computational results suggest, randomization in crossover is critical for a GA in its overall performance to achieve better quality solutions.
### APPENDIX A: MGA-GA and Original-GA Comparisons

#### Table 12: Original-GA and MGA-GA results for each case and different steps

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>steps</th>
<th>Original-GA</th>
<th>MGA-GA</th>
<th>Original-GA</th>
<th>MGA-GA</th>
<th>Original-GA</th>
<th>MGA-GA</th>
<th>Original-GA</th>
<th>MGA-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>24164</td>
<td>24174</td>
<td>24182</td>
<td>24190</td>
<td>24197</td>
<td>24197</td>
<td>24197</td>
<td>24197</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>43202</td>
<td>43230</td>
<td>43236</td>
<td>43239</td>
<td>43234</td>
<td>43251</td>
<td>43253</td>
<td>43253</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>60432</td>
<td>60452</td>
<td>60455</td>
<td>60463</td>
<td>60467</td>
<td>60471</td>
<td>60471</td>
<td>60471</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>60269</td>
<td>60345</td>
<td>60348</td>
<td>60360</td>
<td>60354</td>
<td>60394</td>
<td>60405</td>
<td>60404</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>109080</td>
<td>109221</td>
<td>109241</td>
<td>109254</td>
<td>109260</td>
<td>109272</td>
<td>109280</td>
<td>109287</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>151427</td>
<td>151506</td>
<td>151519</td>
<td>151542</td>
<td>151530</td>
<td>151551</td>
<td>151558</td>
<td>151558</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>119999</td>
<td>120546</td>
<td>120574</td>
<td>120572</td>
<td>120558</td>
<td>120597</td>
<td>120609</td>
<td>120611</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>217093</td>
<td>219467</td>
<td>219449</td>
<td>219474</td>
<td>219459</td>
<td>219496</td>
<td>219502</td>
<td>219503</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>301302</td>
<td>302300</td>
<td>302314</td>
<td>302341</td>
<td>302315</td>
<td>302345</td>
<td>302349</td>
<td>302350</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0.25</td>
<td>22516</td>
<td>22524</td>
<td>22544</td>
<td>22550</td>
<td>22554</td>
<td>22584</td>
<td>22602</td>
<td>22602</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>42538</td>
<td>42583</td>
<td>42612</td>
<td>42600</td>
<td>42637</td>
<td>42644</td>
<td>42651</td>
<td>42659</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59479</td>
<td>59520</td>
<td>59545</td>
<td>59539</td>
<td>59545</td>
<td>59549</td>
<td>59556</td>
<td>59556</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>58824</td>
<td>58855</td>
<td>58884</td>
<td>58921</td>
<td>58918</td>
<td>58967</td>
<td>58992</td>
<td>58984</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>108428</td>
<td>108599</td>
<td>108647</td>
<td>108647</td>
<td>108649</td>
<td>108678</td>
<td>108716</td>
<td>108711</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>151119</td>
<td>151261</td>
<td>151281</td>
<td>151282</td>
<td>151282</td>
<td>151312</td>
<td>151331</td>
<td>151331</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>117675</td>
<td>118431</td>
<td>118464</td>
<td>118482</td>
<td>118498</td>
<td>118540</td>
<td>118566</td>
<td>118555</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>215004</td>
<td>217128</td>
<td>217159</td>
<td>217215</td>
<td>217202</td>
<td>217241</td>
<td>217262</td>
<td>217262</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>301409</td>
<td>302469</td>
<td>302494</td>
<td>302526</td>
<td>302505</td>
<td>302541</td>
<td>302559</td>
<td>302559</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.25</td>
<td>21457</td>
<td>21533</td>
<td>21586</td>
<td>21598</td>
<td>21635</td>
<td>21634</td>
<td>21652</td>
<td>21652</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>41287</td>
<td>41383</td>
<td>41399</td>
<td>41400</td>
<td>41410</td>
<td>41415</td>
<td>41431</td>
<td>41432</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59099</td>
<td>59139</td>
<td>59165</td>
<td>59167</td>
<td>59186</td>
<td>59181</td>
<td>59199</td>
<td>59199</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>56542</td>
<td>56661</td>
<td>56749</td>
<td>56739</td>
<td>56784</td>
<td>56844</td>
<td>56890</td>
<td>56867</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>106252</td>
<td>106513</td>
<td>106562</td>
<td>106580</td>
<td>106617</td>
<td>106617</td>
<td>106652</td>
<td>106653</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>150175</td>
<td>150294</td>
<td>150359</td>
<td>150366</td>
<td>150391</td>
<td>150423</td>
<td>150442</td>
<td>150440</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>114374</td>
<td>115181</td>
<td>115281</td>
<td>115321</td>
<td>115417</td>
<td>115460</td>
<td>115463</td>
<td>115457</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>213605</td>
<td>215922</td>
<td>216022</td>
<td>216026</td>
<td>216090</td>
<td>216102</td>
<td>216163</td>
<td>216145</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>300912</td>
<td>302190</td>
<td>302257</td>
<td>302259</td>
<td>302292</td>
<td>302309</td>
<td>302349</td>
<td>302353</td>
</tr>
<tr>
<td>summary</td>
<td></td>
<td></td>
<td>119543</td>
<td>120053</td>
<td>120086</td>
<td>120098</td>
<td>120111</td>
<td>120134</td>
<td>120152</td>
<td>120150</td>
</tr>
</tbody>
</table>
APPENDIX B: Figures of Selective Comparison Results for MGA-GA and Original-GA

Figure 3: MGA-GA and Original-GA comparison for m=5, n=100, \( \alpha=0.50 \)

Figure 4: MGA-GA and Original-GA comparison for m=5, n=250, \( \alpha=0.50 \)
Figure 5: MGA-GA and Original-GA comparison for $m=5$, $n=250$, $\alpha=0.75$

Figure 6: MGA-GA and Original-GA comparison for $m=5$, $n=500$, $\alpha=0.25$

Figure 7: MGA-GA and Original-GA comparison for $m=10$, $n=250$, $\alpha=0.25$
Figure 8: MGA-GA and Original-GA comparison for m=10, n=250, α=0.50

Figure 9: MGA-GA and Original-GA comparison for m=10, n=500, α=0.25

Figure 10: MGA-GA and Original-GA comparison for m=10, n=500, α=0.75
Figure 11: MGA-GA and Original-GA comparison for $m=30$, $n=250$, $\alpha=0.25$

Figure 12: MGA-GA and Original-GA comparison for $m=30$, $n=500$, $\alpha=0.50$
APPENDIX C: MGA-GA-PR and MGA-GA-Uniform Comparisons

Table 13: MGA-GA-PR/Uniform (10% elite ratio) and MGA-GA-Uniform comparison

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>alpha</th>
<th>MGA-GA-PR/Uniform-sol</th>
<th>MGA-GA-PR/Uniform-time</th>
<th>MGA-GA-sol</th>
<th>MGA-GA-time</th>
<th>&quot;&gt;&quot;</th>
<th>&quot;=&quot;</th>
<th>&quot;&lt;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>24195</td>
<td>617</td>
<td>24197</td>
<td>573</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>43253</td>
<td>578</td>
<td>43253</td>
<td>520</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>60471</td>
<td>547</td>
<td>60471</td>
<td>496</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>60405</td>
<td>1745</td>
<td>60404</td>
<td>1267</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>109288</td>
<td>2125</td>
<td>109287</td>
<td>1136</td>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>151555</td>
<td>2153</td>
<td>151558</td>
<td>1043</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>120611</td>
<td>5655</td>
<td>120611</td>
<td>2440</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>219504</td>
<td>4435</td>
<td>219503</td>
<td>2189</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302354</td>
<td>5266</td>
<td>302350</td>
<td>1997</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0.25</td>
<td>22594</td>
<td>665</td>
<td>22602</td>
<td>758</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>42661</td>
<td>616</td>
<td>42659</td>
<td>684</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59556</td>
<td>660</td>
<td>59556</td>
<td>645</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>59004</td>
<td>1766</td>
<td>58984</td>
<td>1682</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>108708</td>
<td>2016</td>
<td>108711</td>
<td>1541</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>151328</td>
<td>2360</td>
<td>151331</td>
<td>1433</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>118558</td>
<td>5360</td>
<td>118555</td>
<td>3264</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>217272</td>
<td>5004</td>
<td>217262</td>
<td>2923</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302556</td>
<td>5791</td>
<td>302559</td>
<td>2711</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.25</td>
<td>21654</td>
<td>1124</td>
<td>21652</td>
<td>1437</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>41432</td>
<td>1279</td>
<td>41432</td>
<td>1373</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59196</td>
<td>1303</td>
<td>59199</td>
<td>1342</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td></td>
<td>0.25</td>
<td>56890</td>
<td>2920</td>
<td>56867</td>
<td>3383</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>106671</td>
<td>3268</td>
<td>106653</td>
<td>3175</td>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>150444</td>
<td>3829</td>
<td>150440</td>
<td>2980</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td>0.25</td>
<td>115466</td>
<td>6249</td>
<td>115457</td>
<td>6514</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>216155</td>
<td>7112</td>
<td>216145</td>
<td>6159</td>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302361</td>
<td>8559</td>
<td>302353</td>
<td>5952</td>
<td>6</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>summary</td>
<td></td>
<td></td>
<td>120153</td>
<td>3074</td>
<td>120150</td>
<td>2208</td>
<td>65</td>
<td>168</td>
<td>37</td>
</tr>
</tbody>
</table>
Table 14: MGA-GA-PR/Uniform (10% elite ratio) and Original-GA comparison

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>alpha</th>
<th>MGA-GA-PR/Uniform-sol</th>
<th>MGA-GA-PR/Uniform-time</th>
<th>Original-GA-sol</th>
<th>Original-GA-time</th>
<th>&quot;&quot;&gt;&quot;</th>
<th>&quot;=&quot;</th>
<th>&quot;&lt;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>0.25</td>
<td>24195</td>
<td>617</td>
<td>24197</td>
<td>572</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>43253</td>
<td>578</td>
<td>43253</td>
<td>520</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>60471</td>
<td>547</td>
<td>60471</td>
<td>496</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>0.25</td>
<td>60405</td>
<td>1745</td>
<td>60405</td>
<td>1270</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>109288</td>
<td>2125</td>
<td>109280</td>
<td>1135</td>
<td>6</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>151555</td>
<td>2153</td>
<td>151558</td>
<td>1045</td>
<td>0</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>0.25</td>
<td>120611</td>
<td>5655</td>
<td>120609</td>
<td>2424</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>219504</td>
<td>4435</td>
<td>219502</td>
<td>2188</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302354</td>
<td>5266</td>
<td>302349</td>
<td>2004</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>0.25</td>
<td>22594</td>
<td>665</td>
<td>22602</td>
<td>759</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>42661</td>
<td>616</td>
<td>42651</td>
<td>685</td>
<td>2</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59556</td>
<td>660</td>
<td>59556</td>
<td>645</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>0.25</td>
<td>59004</td>
<td>1766</td>
<td>58992</td>
<td>1677</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>108708</td>
<td>2016</td>
<td>108716</td>
<td>1534</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>151328</td>
<td>2360</td>
<td>151331</td>
<td>1436</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>0.25</td>
<td>118558</td>
<td>5360</td>
<td>118566</td>
<td>3286</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>217272</td>
<td>5004</td>
<td>217262</td>
<td>2929</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302356</td>
<td>5791</td>
<td>302559</td>
<td>2706</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.25</td>
<td>21654</td>
<td>1124</td>
<td>21652</td>
<td>1436</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>41432</td>
<td>1279</td>
<td>41431</td>
<td>1373</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>59196</td>
<td>1303</td>
<td>59199</td>
<td>1340</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td>100</td>
<td>0.25</td>
<td>56890</td>
<td>2920</td>
<td>56890</td>
<td>3379</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>106671</td>
<td>3268</td>
<td>106662</td>
<td>3160</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>150444</td>
<td>3829</td>
<td>150442</td>
<td>2982</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>0.25</td>
<td>115466</td>
<td>6249</td>
<td>115463</td>
<td>6513</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.50</td>
<td>216155</td>
<td>7112</td>
<td>216163</td>
<td>6149</td>
<td>7</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.75</td>
<td>302361</td>
<td>8559</td>
<td>302349</td>
<td>5896</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>summary</td>
<td></td>
<td></td>
<td>120153</td>
<td>3074</td>
<td>120152</td>
<td>2205</td>
<td>64</td>
<td>160</td>
<td>46</td>
</tr>
</tbody>
</table>
REFERENCES


