MODELLING THE MAXIMUM REVENUE FROM THE REPAIRS OF CARS OF AN AUTO WORKSHOP IN GHANA

By

JOHN ESSUMAN (Bsc. Mechanical Engineering)

A Thesis Submitted to the Department of Mathematics, Kwame Nkrumah University of Science and Technology, Kumasi, in partial fulfillment of the requirement for the degree of

Master of Science

Industrial Maths

Institute of Distance Learning

May, 2012

DECLARATION

I hereby declare that this submission is my own work towards the Msc and that, to the best of my knowledge; it contains no material previously published by another person or materials, which have been accepted for the award of any other degree of the University, except where the acknowledgement has been made in the text.

John Essuman (PG3012609) ....

Student Name & ID No. Signature Date

Certified by:

Mr. K.F. Darkwah ....

Supervisor's Name. Signature Date

Certified by:

Mr. K.F. Darkwah ....

Head of Department's Name. Signature Date

ABSTRACT

Manual selection approaches in the repairs of cars are usually inadequate and cannot provide the best solution for a company to maximize revenue; hence a tremendous effort has been spent for improving operational productivity through effective and efficient means of selections.

This research develops a procedure of using a simple knapsack model to solve the problem of selecting and scheduling cars due for repairs in an auto workshop environment. This is a direct application of a Knapsack problem to an industrial problem of selection and scheduling.

This thesis considers the application of the classical 0-1 knapsack problem with a single constraint to the selection of some cars which has to be repaired within a given time. Our objective of selecting these cars is to maximize income from the associated labour charges earned from the repairs of the cars.

Our focus is to use a simple scientific approach of Dynamic Programming that can solve the classical 0-1 knapsack problem above.

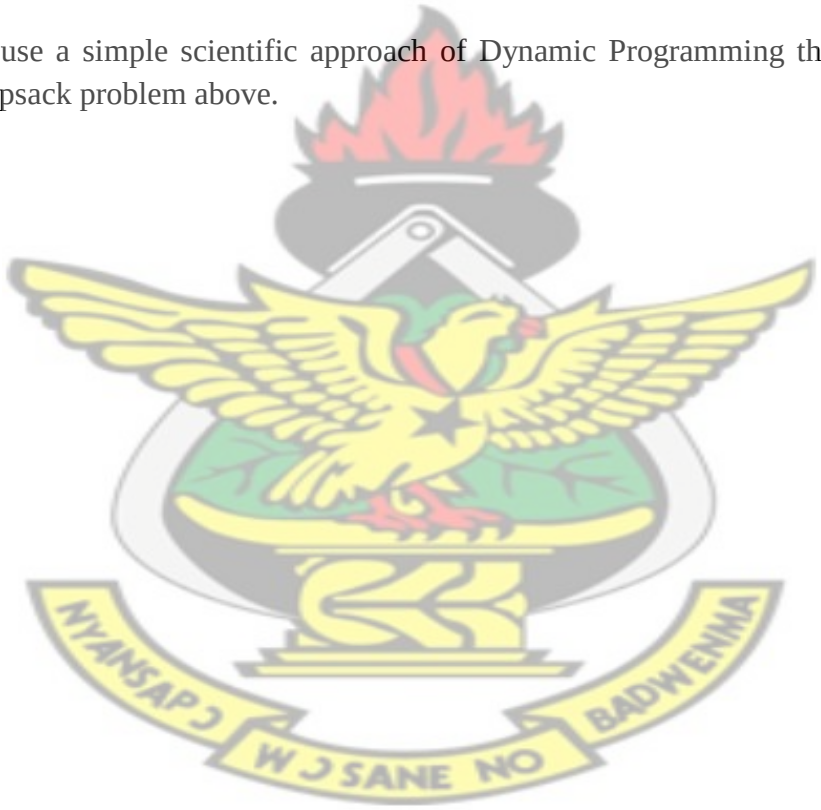


TABLE OF CONTENTS

List of Tables

List of Figures

DEDICATION

I dedicate this thesis to God Almighty my creator and my God. He has guided me and given me strength throughout this work. May His name which is above all names be praised forever.

I also dedicate this thesis to my entire family for their prayers and encouragement.

ACKNOWLEDGEMENT

I give thanks to the Almighty God for giving me the opportunity, wisdom and guidance for undertaking this programme and completing it with this thesis.

I would like to express my sincere appreciation to my supervisor Mr. K.F. Darkwah of the Department of Mathematics, K.N.U.S.T, Kumasi for his step by step guidance and support throughout this study. It was indeed a privilege for me to receive his tremendous inputs.

Special thanks to my wife Mrs. Naa Adei Essuman for her patience and support, and also sincere thanks to my children, Esther and David for their understanding and encouragement.

God Bless you all.

CHAPTER 1

1.0 INTRODUCTION

There is a maxim in the automobile industry which says that: 'the first car is sold by the salesman but all subsequent ones are sold by good after-sales service.' This shows an absolute necessity to recognize the Auto workshop as an important unit in the Auto industry because it can be used to promote sales of cars and also customer loyalty. Customers who become delighted as a result of good after-sale service have the highest probability of remaining loyal to the company that served them.

Motor vehicles are now regarded as necessities in most developed nations. The number of cars, vans, trucks, and buses in the world now averages at least one for every 12 human beings.

The situation is not too different from what pertains in Ghana. Cars and bikes no longer fall under the category of luxurious items in Ghana; rather they have now become the basic necessities of life. Ghanaians have realized that they all require safe vehicles to commute and make their day-to-day travelling easier. We all have our dreams and to own a personal vehicle is everyone's dream. To buy a vehicle is a dream of several people and the ones who have been successful in achieving these dreams ensure that they maintain their vehicle in the best possible way.

1.1 BACKGROUND OF STUDY

The Automobile industry in Ghana continues to experience an unforeseen boom. Thanks to the buoyant economic condition, today almost every decently earning Ghanaian dreams of owning a car. Again, due to the investor-friendly climate in the country and the adequate legislation to

promote and protect businesses to thrive in the country, many investors, both foreign and local, have taken advantage of the situation to open businesses and the Automobile sector is not an exception.

Between the period 1980 and 1990, one could hardly see automobile companies scattered around the country. There were only a few like Toyota Ghana, Japan Motors, Auto Parts and probably CFAO among others. The situation is now different. Today, the industry has seen tremendous growth resulting in individuals importing vehicles both brand new and second hand from Europe for sale in Ghana. Almost along, every highway and major road is a garage displaying different models of vehicles for sale. The fascinating thing about the industry is that some of the players have gone the extra mile to liaise with the banks to provide credit facility for their prospective buyers.

Currently, Ghana can boast of about 20 automobile companies that sell brand new cars and over 1000 garages that sell slightly used cars (home second-hand cars) in the country. Those engaged in the sale of the second-hand vehicles continue to increase day in day out churning out moderate prices to enable customers to purchase from them. Competition in the industry has become keener and keener, while dealers continue to devise means of attracting customers to buy their vehicles. One of the means that auto companies can use to attract customers is reducing throughput times in their after-sale service departments. *(Automobile Industry in Ghana, Daily Graphic, Page 18, Wednesday, November 24, 2010)*

Normally the associated decisions involve the assignment of tasks to resources and finding the processing order of jobs on each resource in order to achieve in advance, a specific goal(s) of a company.

In general after sale service activity, managers analyze the capacity of their outfit and customer demand, and then develop a servicing schedule to assign spare parts and tools to jobs to be carried out.

The after-sales service activities in recent times are acknowledged as a significant source of revenue, profit and competitive advantage in most Auto workshops in Ghana. Managers of Auto workshop therefore have the fundamental obligation to ensure the most efficient deployment of the monetary investment allocated to the running of their workshops. If this basic framework is adhered to, then managers of Auto workshops can be able to maximize the total net present value of all revenues accrued from all the jobs (i.e. repairs and servicing) that are carried out in their workshops.

One of the processes that can be used to achieve the above goal is an effective allocation model based on monetary issues. Therefore an efficient and effective allocation process can enable Auto workshops to allocate their limited resources (repair time) to jobs (repairs and servicing) over a specified period of time.

Premier Technik Motors Ltd is an auto workshop located at South Industrial Area, Accra in Ghana. It was incorporated on 25th March 2009, starting business in the same month. The workshop services Vw, Audi, Skoda and Volvo brands of vehicles. It has Auto-robot jig, Spraying Booth and Oven, Hydraulic and Electric workshop lifts, a spare parts store and a workshop floor. It has a team of technicians for the four different brands of Vehicles they repair. Some of these technicians have had training with Volkswagen and Skoda companies in Germany and Czech Republic respectively. This team has thorough knowledge of repairs and hands on experience with respect to these types of vehicles (i.e. Vw, Audi, Skoda and Volvo). The

customer profile of this workshop includes Ssnit,Cocoa Research Institute of Ghana, Cocobod, Seed Production Unit Of Cocobod, Trust Bank, Canadian High Commission, Netherlands Embassy, Valco Trust Fund, InterContinental Bank, Labadi Beach Resort and Elmina Beach Resort.

1.2 PROBLEM STATEMENT

In Ghana, most auto workshops of recognized car dealers such as Toyota Ghana Ltd, Mechanical Lloyd, Honda Place, Stallion motors and other car garages, have a general time resource allocation problem in their workshops. In these workshops, a single resource (repair time for cars) is assigned to a number of options (cars to be repaired) with the objective of maximizing the total returns. Most of the time, total returns from this single resource (repair time for cars) is not maximized because managers of these workshops use their own discretion instead of a scientific approach to do the resource allocation.

Premier Technik Motors carries out 10 different types of repair jobs in their workshop. The car types they repair are: Volvo, Audi, Volkswagen and Skoda. The four technician teams that work in the workshop are Volvo team, Audi team, VW team and Skoda team. Each of these technician teams of the workshop has a maximum available labour time of 176 hours (22 working days) as a team to work in their respective department. All the different type of cars have their repair time and associated labour charges respectively. On the average, the job scheduler receives a certain number of cars due for repairs per month from corporate organizations and individuals.

Management of this Auto workshop has been using the manual selection approach to assign resources (repair time) to jobs (repair of cars) which in these modern times is inadequate and not scientific. Our research work is to find a selection approach that can determine how many cars

the workshop can repair (i.e., Vw, Skoda, Audi and Volvo) in order to maximize total returns every month .

The problem above conforms to a knapsack-type problem in which a set of entities are given, each having an associated value and size, and it is desired to select one or more disjoint subset so that the sum of the sizes in each subset does not equals or exceed a given bound (total labour time for the month) and the sum of the selected values (labour charges) is maximized.

1.3 OBJECTIVES

The objectives are:

(1) To model the maximum revenue accrued from the repairs of cars as a knapsack problem.

(2) To solve the knapsack problem using Dynamic Programming.

1.4 METHODOLOGY

In order to achieve the objective above, various types of knapsack problems and some of the well known algorithms for solving them would be discussed. Finally we will apply the Dynamic Programming algorithm for knapsack to solve our problem. Data on car repair time schedule with its associated labour charges for the period 2010/2011 year which is used in this study was collected from the workshop of Premier Technik Motors. Literature from the KNUST, Science College Library and the Internet were used to enhance this thesis.

1.5 JUSTIFICATION

Fast delivery of a good after-sale service of cars is very crucial to customers in Ghana because it enables them to use their repaired cars to meet other production requirements in their day to day

activities. Installing a quality service or repairs to a faulty car by a workshop generates customer loyalty which also translates into profit making. Delays in car repairs can be minimized if the selection of which car to repair first or last can be sorted out by a good selection approach. Most workshops do not have a well structured system that selects cars that are due for repairs based on the best possible use of resources which also results in maximum total returns. Hence, this is the reason for solving this problem as a knapsack problem.

1.6 ORGANIZATION OF THESIS

Chapter 1 consists of the problem of scheduling the different repair types of cars in the workshop of Premier Technik motors workshop, methodology and justification for the use of knapsack problems application to solve the car repairs selection problem.

Chapter 2 consists of literature review on knapsack problems applications and its proposed solution methods.

Chapter 3 consists of the methodology for solving our problem. Included in this chapter will be the evaluation of various types of knapsack problems and some of the well known algorithms for solving them.

Chapter 4 consists of the data collection and analysis of the actual data of type of car, repair time and labour charges from Premier Technik Motor's workshop.

Chapter 5 consists of the conclusion and recommendation.

CHAPTER 2

LITERATURE REVIEW

Many practical repair scheduling problems can be represented by a set of entities, each having an associated value, from which, one or more subsets has to be selected in such a way that the sum of the values of the selected entities is maximized with respect to certain conditions. These problems are classified as knapsack problems since they call the situation of a traveler having to fill up his knapsack by selecting from among various possible objects that can give him/her the maximum comfort. This concept has been used to model many industrial applications such as cargo loading and advertisement selection in a broadcasting firm. In this chapter, a literature review on the knapsack problems and its applications is presented.

Knapsack problems have been intensively studied because they arise as sub problems in various integer programming problems and may represent many practical scenarios. The most typical applications are in capital budgeting and industrial production. Various capital budgeting models have been studied by Weingartner (1963, 1968), Weingartner and Ness (1967), Cord (1964) and Kaplan (1966). Among industrial applications, the classical studies a cargo loading problem (Bellman and Dreyfus, 1962) and on cutting stock problems, Gillmore and Gomory (1963; 1965; and 1966) is worth mentioning. More detailed reviews of applications can be found in salkin (1975) and Martello and Toth (1987).

The family of Knapsack Problems all requires a subset of some given items to be chosen such that the corresponding profits sum is maximized without exceeding the capacity of the knapsack(s). Different types of Knapsack Problems occur, depending on the distribution of the items and knapsacks: In the 0-1Knapsack Problem each item may be chosen at most once, while

in the Bounded Knapsack Problem we have a bounded amount of each item type. The Multiple-choice Knapsack Problem occurs when the items should be chosen from disjoint classes and, if several knapsacks are to be filled simultaneously, we get the Multiple Knapsack Problem. The most general form is the Multi-constrained Knapsack Problem, which basically is a general Integer Programming (IP) Problem with positive coefficients.

Pisinger (1999) presented an algorithm for knapsack problem where the enumerated core size is minimal and the computational effort for sorting and reduction were also limited according to hierarchy. The algorithm is based on a dynamic programming approach, where the core size is extended by need and the sorting and reduction is performed in a similar "lazy" way. Computational experiments are presented for several commonly occurring types of data instances. Experience from these tests indicates that the presented approach outperforms any known algorithm for knapsack problem, having very stable solution times.

Martello and Toth (1988) presented a new algorithm for the optimal solution of the 0-1 knapsack problems, which is particularly effective for large-size problems. T he algorithm is based on determination of an appropriate small subset of items and the solution of the corresponding "core problem". From this, they derived a heuristic solution for the original problem which, with high probability, can be proved to be optimal. The algorithm incorporates a new method of computation of upper bounds and efficient implementations of reduction procedures. They also reported computational experiments on small-size and large-size random problems, comparing the proposed code with all those available in the literature.

Munapo (2008) presented an approach that enhances the performance of the branch and bound algorithm for the knapsack model. This is achieved by generating and adding new objective

function and constraint to knapsack model, which is single constrained. The branch and bound algorithm is then applied and the total numbers of sub-problems are reduced

Bazgan et al. (2007) presented an approach, based on dynamic programming, for solving the 0-1 multi-objective knapsack problem. The main idea of the approach relies on the use of several complementary dominance relations to discard partial solutions that cannot lead to new non-dominated criterion vectors. This way, they obtained an efficient method that outperforms the existing methods both in terms of CPU time and size of solved instances. Extensive numerical experiments on the various types of instances were reported. A comparison with other exact methods was also performed.

Ferreira (1995) presented parallel algorithms for solving a knapsack problem of size n on PRAM and distributed memory machines. The algorithms were efficient in the sense that they achieved optimal speed up with regard to the best known solution to this problem. Morever, they matched the best current time/memory/processors tradeoffs, while requiring less memory and processors. Since the PRAM is considered mainly as a theoretical model and we want to produce practical algorithms for the knapsack problem, its solution in distributed memory machines is also studied.

Glickman and Allison, (1973) considered the problem of choosing among the technologies available for irrigation by tubewell to obtain an investment plan which maximizes the net agricultural benefits from the proposed project in a developing country. Cost and benefit relationships were derived and incorporated into a mathematical model which is solved using a modification of the dynamic programming procedure for solving the knapsack problem. The optimal schedule was seen to favour small capacity wells, drilled by indigenous methods, with supplementary water distribution systems.

Huttler and Mastrolilli (2006) addressed the classical knapsack problem and a variant in which an upper bound is imposed on the number of items that can be selected. They showed that appropriate combinations of rounding techniques yielded novel and more powerful ways of rounding. Morever, they presented a linear-storage polynomial time approximation scheme (PTAS) and a fully polynomial time approximation scheme (FPTAS) that compute an approximate solution of any fixed accuracy, in linear time. These linear complexity bounds give a substantial improvement of the best previously known polynomial bounds.

Gomes da Silva et al. (2007) dealt with the problem of inaccuracy of the solutions generated by meta-heuristic approaches for combinatorial optimization bi-criteria (0-1)-knapsack problems. A hybrid approach which combines systematic and heuristic searches was proposed to reduce that inaccuracy in the context of a scatter search method. The components of this method were used to determine regions in the decision space to be systematically searched. Comparisons with small and medium size instances solved by exact methods were presented. Large size instances were also considered and the quality of the approximation was evaluated by taking into account the proximity to the upper frontier, devised by the linear relaxation and the diversity of the solutions. Comparisons with other two well known meta-heuristics were also performed. The results showed the effectiveness of the proposed approach for both small/medium and large size instances.

Rinnooy et al. (1993) proposed a class of generalized greedy algorithms for the solution of multi-knapsack problem. Items are selected according to decreasing ratios of their profit and a weighted sum of their requirement coefficients. The solution obtained depended on the choice of the weights. A geometrical representation of the method was given and the relation to the dual of the linear programming relaxation of multi-knapsack is exploited. They investigated the

complexity of computing a set of weights that gives the maximum greedy solution value. Finally, the heuristics were subjected to both a worst case and a probabilistic performance analysis.

Figuera et al. (2009) presented a generic labeling algorithm for finding all non-dominated outcomes of the multiple objective integer knapsack problem (MOIKP). The algorithm is based on solving the multiple objective shortest path problem on an underlying network. Algorithms for constructing four network models, all representing the MOIKP, were also presented. Each network is composed of layers and each network algorithm, working forward layer by layer identifies the set of all permanent non-dominated labels for each layer. The effectiveness of the algorithms is supported with numerical results obtained for randomly generated problems for up to seven objectives while exact algorithms reported in the literature solve the multiple objective binary knapsack problem with up to three objectives. Extensions of the approach to other classes of problems including binary variables, bounded variables, multiple constraints and time-dependent objective functions are possible.

Majority of algorithms for solving knapsack problems typically use implicit enumeration approaches. Different bounds based on the remaining capacity of the knapsack and items not yet included at certain iteration have been proposed for use in these algorithms. Similar methods may be used for a nested knapsack problem as long as there is an established procedure for testing whether an item inserted into a knapsack at one stage can also be inserted at the following stages. Given *n* different items and a knapsack of capacity, Caceres and Nishibe (2005) algorithm solves the 0-1 knapsack problem using O (nWp) local computation time with O(p) communication rounds. Using dynamic programming, their algorithm solves locally pieces of the knapsack problem in each processor and uses a wave front approach in order to solve the whole

problem. The algorithm was implemented in a Beowulf and the obtained times showed good speed-up and scalability.

The binary knapsack problem is a combinatorial optimization problem in which a subset of a given set of elements needs to be chosen in order to maximize profit, given a budget constraint. Das (2003) used a stochastic version of the problem in which the budget is random. They proposed two different formulations of this problem, based on different ways of handling infeasibility and proposed an exact algorithm and a local search-based heuristic to solve the problems represented by these formulations. The results were presented from some computational experiments.

The knapsack problem is believed to be one of the "easier"-hard problems. Not only can it be solved in pseudo-polynomial time but also decades of algorithmic improvements have made it possible to solve nearly all standard instances from the literature. Pisinger (2005) gave an overview of all recent exact solution approaches and to show that the knapsack problem is still hard to solve for these algorithms for a variety of new test problems. These problems are constructed either by using standard benchmark instances with larger coefficients or by introducing new classes of instances for which most upper bounds perform badly. The first group of problems challenges the dynamic programming algorithms while the other groups of problems are focused towards branch and bound algorithms. Numerous computational experiments with all recent state-of-the-art codes are used to show that knapsack problem (KP) is still difficult to solve for a wide number of problems. One could say that the previous benchmark tests were limited to a few highly structured instances, which do not show the full characteristics of knapsack problems.

The 0-1 knapsack problem is well known and it appears in many real domains with practical importance. The problem is NP-complete. The multi-objective 0-1 knapsack problem is a generalization of the 0-1 knapsack problem in which many knapsacks are considered. Many algorithms have been proposed in the past four decades for both single and multi-objective knapsack problem. A new evolutionary algorithm for solving multi-objective 0-1 knapsack problem was proposed by Groan (2003). This algorithm used a ξ-dominance relation for direct comparison of two solutions. Some numerical experiments are realized using the best and recent algorithms proposed for this problem. Experimental results showed that the new proposed algorithm outperforms the existing evolutionary approaches for this problem.

Puchinger (2006) presented a newly developed core concept for the multidimensional knapsack problem (MKP) which is an extension of the classical concept for the one-dimensional case. The core for the multidimensional problem is defined in dependence of a chosen efficiency function of the items, since no single obvious efficiency measure is available for MKP. An empirical study on the cores of widely-used benchmark instances is presented, as well as experiments with different approximate core sizes. Furthermore, they described a memetic algorithm and a relaxation guided variable neighborhood search for the MKP, which are applied the original and to the core problems. The experimental results show that given a fixed run-time, the different meta-heuristics as well as a general purpose integer linear programming solver yield better solution when applied to approximate core problems of fixed size

Fontanari (1995) investigated the dependence of the multi-knapsack objective function on the knapsack capacities and on the number of capacity constraints P, in the case when all N objects are assigned the same profit value and the weights are uniformly distributed over the unit interval. A rigorous upper bound to the optimal profit is obtained, employing the annealed

approximation and then, compared with the exact value obtained through the Langrangian relaxation method. The analysis is restricted to the regime where N goes to infinity and P remains finite.

Benisch et al. (2005) examined the problem of choosing discriminatory prices for customers with probabilistic valuations and a seller with indistinguishable copies of goods. They showed that under certain assumptions this problem can be reduced to the continuous knapsack problem (CKP). They presented a new fast epsilon-optimal algorithm for solving CKP instances with asymmetric concave reward functions. They also showed that their algorithm can be extended beyond the CKP setting to handle pricing problems with overlapping goods (e.g goods with common components or common resource requirements), rather than indistinguishable goods. They provided a framework for learning distributions over customer valuations from historical data that are accurate and compatible with their CKP algorithm. They validated their techniques with experiments on pricing instances derived from the Trading Agent Competition in Supply Chain Management (TAC SCM). Their results confirmed that their algorithm converges to an epsilon-optimal solution more quickly in practice than an adaptation of a previously proposed greedy heuristic.

Pendharkar et al. (2005) described an information technology capital budgeting (ITCB) problem and showed that the ITCB problem can be modeled as a 0-1 knapsack optimization problem and proposed two different simulated annealing (SA) heuristic solution procedures to solve the ITCB problem. Using several simulations, they empirically compared the performance of two SA heuristic procedures with the performance of two well-known ranking methods for capital budgeting. Their results indicated that the information technology (IT) investments selected

using the SA heuristics have higher after-tax profits than the IT investments selected using the two ranking methods.

The bounded Knapsack Problem (BKP) is a generalization of the 0-1 Knapsack Problem where a bounded amount of each item type is available. Currently, the most efficient algorithm for BKP transforms the data instance to an equivalent 0-1 Knapsack Problem, which is solved efficiently through a specialized algorithm. Pisinger (1996) proposed a specialized algorithm that solves an expanding core problem through dynamic programming such that the number of enumerated item types is minimal. Sorting and reduction is done by need, resulting in very little effort for the preprocessing. Compared to other algorithms for BKP, the presented algorithm uses tighter reductions and enumerates considerably less item types. Computational experiments are presented, showing that the presented algorithm outperforms all previously published algorithms for BKP.

The multidimensional 0-1 knapsack problem, defined as a knapsack with multiple resource constraints, is well known to be much more difficult than the single constraint version. Freville and Plateau (2004) designed an efficient preprocessing procedure for large-scale instances. The algorithm provides sharp lower and upper bounds on the optimal value and also a tighter equivalent representation by reducing the continuos feasible set and by eliminating constraints and variables. This scheme was shown to be very effective through a lot of computational experiments with test problems of the literature and large-scale randomly generated instances.

The knapsack sharing problem (KSP) is formulated as an extension to the ordinary knapsack problem. The KSP is NP-hard. Yamada et al. (1998) presented a branch-and-bound algorithm and a binary search algorithm to solve this problem to optimality. These algorithms are

implemented and computational experiments are carried out to analyse the behavior of the developed algorithms. As a result, they found that the binary search algorithm solves KSPs with up to 20 000 variables in less than a minute in their computing environment.

The objective of the multi-dimensional knapsack problem (MKP) is to find a subset of items with maximum value that satisfies a number of knapsack constraints. Solution methods for MKP, both heuristic and exact, have been researched for several decades. Fleszar and Hindi (2009) introduced several fast and effective heuristics for MKP that are based on solving the LP relaxation of the problem. Improving procedures were proposed to strengthen the results of these heuristics. Additionally, the heuristics were run with appropriate deterministic or randomly generated constraints imposed on the linear relaxation that allow generating a number of good solutions. All algorithms were tested experimentally on a widely used set of benchmark problem instances to show that they compared favourably with the best-performing heuristics available in the literature.

Transportation programming, a process of selecting projects for funding given budget and other constraints, is becoming more complex. Zhong and Young (2009) described the use of an integer programming tool, Multiple Choice Knapsack Problem (MCKP) to provide optimal solutions to transportation programming problems in cases where alternative versions of projects are under consideration. Optimization methods for use in the transportation programming process were compared and then the process of building and solving the optimization problems discussed. The concepts about the use of MCKP were presented and a real world transportation programming example at various budget levels were provided. They illustrated how the use of MCKP addresses the modern complexities and provides timely solutions in transportation programming practice.

Jurait et al. (2006) focused on ways to find proportions of the mixture of heuristics which would lead to better performance of the algorithm. New results were compared with earlier research and some other constructive heuristics. The performance of the corresponding algoritms was experimentally compared for homogeneous and heterogeneous instances. Proposed improvements allow achieving better filling ratio without increasing the computational complexity of the algorithm.

Lin and Yao (2001) investigated knapsack problems in which all of the weight coefficients are fuzzy numbers. The work was based on the assumption that each weight coefficient is imprecise due to the use of decimal truncation or rough estimation of the coefficients by the decision maker. To deal with this kind of imprecise data, fuzzy sets provide a powerful tool to model and solve this problem. Their work was intended to extend the original knapsack problem into a more generalized problem that would be useful in practical situations. As a result, their study showed that the fuzzy knapsack problem is an extension of the crisp knapsack problem and that the crisp knapsack problem is a special case of the fuzzy knapsack problem.

Zhang and Ong (2004) proposed a simple and useful method, the core of which is an efficient LP-based heuristic for solving bi-objective 0-1 knapsack problems. Extensive computational experiments showed that the proposed method is able to generate a good approximation to the non dominated set very efficiently. They also suggested three qualitative criteria to evaluate such an approximation. In addition, the method can be extended to other problems having properties similar to the knapsack problem.

A promising solution approach called Meta-RaPS was presented by Moraga et al. (2005) for the 0-1 Multidimensional Knapsack Problem (0-1 MKP). Meta-RaPS construct feasible solutions

at each iteration through the utilization of a priority rule used in a randomized fashion. Four different greedy priority rules are implemented within Meta-RaPS and compared. These rules differ in the way the corresponding pseudo-utility ratios for ranking variables are computed. In addition, two simple local search techniques within Meta-RaPS' improvement stage are implemented. The Meta-RaPS approach is tested on several established test sets and the solution values are compared to both the optimal values and the results of other 0-1 MKP solution techniques. The Meta-RaPS approach outperformed many other solution methodologies in terms of differences from the optimal value and the number of optimal solutions obtained. The advantage of the Meta-RaPS approach is that it is easy to understand and easy to implement and it achieved good results.

Florios et al. (2009) solved instances of the multi-objective multi-constraint (or multidimensional) knapsack problem (MOMCKP) from the literature with three objective functions and three constraints. They used exact as well as approximate algorithms. The exact algorithm is a properly modified version of the multi-criteria branch and bound (MCBB) algorithm which is further customized by suitable heuristics. Three branching heuristics and a more general purpose composite branching and construction heuristics were devised. Furthermore, the same problems are solved using standard multi-objective evolutionary algorithms (MOEA) namely the SPEA2 and the NSGAII. The results from the exact case show that the branching heuristics greatly improve the performance of the MCBB algorithm, which becomes faster than the adaptive ξ-constraint. Regarding the performance of the MOEA algorithms in the specific problems, SPEA2 outperforms NSGAII in the degree of approximation of the Pareto front as measured by the coverage metric (especially for the largest instance).

Abboud et al. (1997) presented an interactive procedure for the multi-objective multidimensional 0-1 knapsack problem that takes into consideration the incorporation of fuzzy goals of the decision maker, that is easy to use since it requires from the decision maker to handle only one parameter, namely, the aspiration level of each objective and that is fast and can treat our problem as a usual 0-1 knapsack problem using already available software called the primal effective gradient method, it is used primarily to solve the large scale cases. To get some statistics on the behavior of the algorithm, a number of randomly generated simulations of problems was solved. From their numerical experience, it is possible to conclude that their proposed method is worthwhile alternative to existing methods from a practical point of view.

Akinc (2006) addressed the formulation and solution of a variation of the classical binary knapsack problem. The variation that was addressed is termed the "Fixed-Charge Knapsack Problem" in which sub-sets of variables (activities) are associated with fixed costs. These costs represented certain set-ups and preparations required for the associated sub-set of activities to be scheduled. Several potential real world applications as well as problem generalizations were discussed. The efficient solution of the problem is facilitated by a standard branch and bound algorithm based on a non-iterative polynomial algorithm to solve the LP relaxation, various heuristic procedures to obtain good candidate solutions by adjusting the LP solution and powerful rules to peg the variables. Computational experience shows that the suggested branch and bound algorithm shows excellent potential in the solution of a wide variety of large fixed charge knapsack problems.

Aissi et al. (2007) investigated for the first time in literature, the approximation of min-max (regret) versions of classical problems like shortest path, minimum spanning tree and knapsack. For a constant number of scenarios, they established fully polynomial-time approximation

schemes for the min-max versions of these problems using relationships between multi-objective and min-max optimization. Using dynamic programming and classical trimming techniques, they constructed a fully polynomial-time approximation scheme for min-max regret shortest path. They also established a fully polynomial-time approximation scheme for min-max regret spanning tree and proved that min-max regret knapsack was not at all approximable. For a non-constant number of scenarios case, min-max regret versions of polynomial-time problems become strongly NP-hard, non-approximability results were provided for min-max (regret) versions of shortest path and spanning tree.

Jan et al. (2006) considered web content adaptation with a bandwidth constraint for server-based adaptive web systems. The problem can be stated as follows: Given a web page P consisting of n components items d_1 , $d_{2\ .\ .\ .}$, d_n and each of the component items d_i having J_i versions d_{i1} , d_{i2} , . . . , d_{ij} for each component item d_i select one of its versions to compose the web page such that the fidelity function is maximized subject to the bandwidth constraint. They formulated this problem as a linear multi-choice knapsack problem (LMCKP) and transformed the LMCKP into a knapsack problem (KP) and then presented a dynamic programming method to solve the KP. A numerical example illustrated the method and showed its effectiveness.

Devyaterikova et al. (2009) presented discrete production planning problem which may be formulated as the multidimensional knapsack problem is considered, while resource quantities of the problem are supposed to be given as intervals. An approach for solving this problem based on using its relaxation set is suggested. Some L-class enumeration algorithms for the problem are described. Results of computational experiments were presented.

CHAPTER 3

METHODOLOGY

The knapsack problem is a general resource allocation problem in which a single resource is assigned to a number of alternatives with the objective of maximizing the total return. The knapsack problem seeks to optimize a set of yes/no decisions subject to a single non-negative constraint. The families of Knapsack Problems all require a subset of some given items to be chosen such that the corresponding profit sum is maximized without exceeding the capacity of the knapsack.

3.1 TYPES OF KNAPSACK PROBLEMS

Different types of Knapsack Problems occur, depending on the distribution of the items and knapsacks: In the 0-1Knapsack Problem each item may be chosen at most once, while in the Bounded Knapsack Problem we have a bounded amount of each item type. The Multiple-choice Knapsack Problem occurs when the items should be chosen from disjoint classes and, if several knapsacks are to be filled simultaneously, we get the Multiple Knapsack Problem. The most general form is the Multi-constrained Knapsack Problem, which basically is a general Integer Programming (IP) Problem with positive coefficients.

3.2 SINGLE KNAPSACK PROBLEMS

There is one container (or knapsack) that must be filled with optimal subset of items. The capacity of such a container will be denoted by c.

Some of the problems under single knapsack are:

(i) 0-1 knapsack problem

(ii) Bounded knapsack problem

3.2.1 THE SINGLE 0-1 KNAPSACK PROBLEM

Consider the classical 0-1 knapsack problem (KP) where a subset of n given items has to be packed in a knapsack of capacity c. Each item has a profit p_j and a weight w_j and the problem is to select a subset of the items whose total weight does not exceed c and whose total profit is a maximum.

We assume, without loss of generality, that all input data are positive integers. Introducing the binary decision variables x_j, with $x_j = 1$ if item j is selected and $x_j = 0$ otherwise, we get the integer linear programming (ILP) model:

Maximize $z = \sum_{j=1}^{n} p_j x_j$

Subject to $\sum_{j=1}^{n} w_j x_j \leq c$

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

Where all data are positive integers

3.2.2 THE BOUNDED KNAPSACK PROBLEM

The bounded knapsack problem (BKP) is:

Given n item types and a knapsack, with

p_j = profit of an item of type j;

w_j = weight of an item of type j;

b_j = upper bound on the availability of an items of type j;

c = capacity of the knapsack,

Select a number x_j (j = 1,…,n) of items of each type so as to

We assume, without loss of generality, that all input data are positive integers. Introducing the binary decision variables x_j, with x_j = 1 if item j is selected and x_j =0 otherwise, we get the integer linear programming (ILP) model:

$$\text{Maximize} \quad z = \sum_{j=1}^{n} p_j x_j$$

$$\text{Subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c$$

$$0 \leq x_j \leq b_j \text{ and integer}, \quad j \in N = 1,\ldots,n$$

Where x_j is bounded non negative number.

3.3 MULTIPLE KNAPSACK PROBLEMS

In this kind of problems, there are more than one container available which must be filled with optimal subset of items. We will give consideration to the 0-1 Multiple Knapsack problem

3.3.1 0-1 MULTIPLE KNAPSACK PROBLEMS

The 0-1 Multiple Knapsack Problem (MKP) is defined as follows:

Given a set of n items and a set of m knapsacks ($m \leq n$), with

p_j = profit of item j;

w_j = weight of item j;

c_i= capacity of knapsack i,

Select m disjoint subsets of items so that the total profit of the selected items is a maximum and each subset can be assigned to a different knapsack whose capacity is no less than the total weight of items in the subset. Thus,

Maximize $z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij}$

Subject to $\sum_{j=1}^{n} w_j x_{ij} \leq c_i$

$\sum_{j=1}^{n} x_{ij} \leq 1$ where $i \in M = \{1, \dots, m\}, j \in N = \{1, \dots, n\}$,

$x_{ij} \in \{0,1\}$ Where $x_{ij} = \begin{cases} 1, & \text{if item j is assigned to knapsack i;} \\ 0, & \text{if otherwise} \end{cases}$

3.4 METHODS FOR SOLVING KNAPSACK PROBLEMS

The 0-1 knapsack problems can be solved by two basic exact methods known as branch-and-bound and dynamic programming methods. However, large scale problems could be solved by the use of meta-heuristics such as Simulated annealing, Genetic algorithm, Variable neighborhood search and Tabu search.

3.4.1 BRANCH AND BOUND METHOD

Branch and bound is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. It consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded by using upper and lower estimated bounds of the quantity being optimized.

BRANCH-AND-BOUND ALGORITHMS FOR KNAPSACK

The first branch-and-bound approach to the exact solution of KP was presented by Kolesar (1967). His algorithm consists of a highest-first binary branching scheme.

The large computer memory and time requirements of the Kolesar algorithm were greatly reduced by the Greenberg and Hegerich (1970) approach, differing in two main respects:

(a) at each mode, the continuous relaxation of the induced sub problem is solved and the corresponding critical items ŝ is selected to generate the two descendent nodes (by imposing $X_{\hat{s}} = 0$ on $(X_{\hat{s}} = 1)$;

(b) the search continues from the node associated with the exclusion of item ŝ (condition $X_{\hat{s}} = 0$).

When the continuous relaxation has an all-integer solution, the search is resumed from the last node generated by imposing $X_{\hat{s}} = 1$, i.e. the algorithm is of depth – first type.

Horowitz and Sahni (1997) (and independently, Ahrens and Finke (1975)) derived from the previous scheme a depth-first algorithm in which;

(a) selection of the branching variable X_j is the same as in Kolesar;

(b) the search continues from the node associated with the insertion of item j (condition $X_j = 1$), i.e. following a greedy strategy.

The Horowitz – Sahni algorithm is the most effective, structured and easy to implement and has constituted the basis for several improvements, including that of Martello – Toth algorithm (Martello and Toth, 1977), which is generally considered highly effective.

The Horowitz – Sahni Algorithm

Assume that the items are sorted. A forward move consists of inserting the largest set of new consecutive items into the current solution. A backtracking move consists of removing the last inserted item from the current solution. Whenever a forward move is exhausted, the upper bound U_1 corresponding to the current solution is computed and compared with the best solution so far, in order to check whether further forward moves could lead to a better one; if so, a new forward move is performed, otherwise a backtracking follows. When the last item has been considered, the current solution is complete and possible updating of the best solution so far occurs. The algorithm stops when no further backtracking can be performed. In the following description of the algorithm we use these notations.

n = number of items

$\hat{x}_j$ = current solution;

P_j = profit of item j;

w_j = weight of item j;

C = capacity of the knapsack;

$\hat{Z}$ = current solution value ($= \sum_{j=1}^{n} p_j \hat{x}_j$)

$\hat{C}$ = current residual capacity ($= C - \sum_{j=1}^{n} w_j \hat{x}_j$)

x_j = best solution so far;

Z = value of the best solution so far ($=\sum_{j=1}^{n} p_j \hat{x}_j$)

The Algorithm (this is for KP)

Input: n, C, (P_j), (w_j);

Output: Z; (x_j);

Begin

1. [Initialize]

   Z: = 0;

   $\hat{Z}$: = 0;

   $\hat{C}$: = C;

   p_{n+1}: = 0;

   w_{n+1}:= + ∞;

   j: = 1

2. [Compute upper bound U_1]

   find r = min {i: $\sum_{k=j}^{i} w_k > \hat{C}$};

   U: = $\sum_{k=j}^{r-1} p_k + \left[\left(\hat{C} - \sum_{k=j}^{r-1} w_k\right)\frac{p_r}{w_r}\right]$;

   If Z ≥ $\hat{Z}$ + U then go to 5;

3. [Perform a forward step]

   while $w_j \leq \hat{C}$ do

   begin

   $\hat{C}$:= $\hat{C}$ - w_j;

   $\hat{Z}$:= $\hat{Z}$+ P_j

   $\hat{x}_j$: = 1

   j : = j + 2

   end

   if j ≤ n then

   begin

   $\hat{x}_j$ = 0

   j = j + 1

end

if j < n then go to 2;

if j = n then go to 3;

4. [Update the best solution so far]

   if $\hat{Z} > Z$ then

   begin

   Z: = $\hat{Z}$;

   for k: = 1 to n do x_k: = $\widehat{x_k}$

   end

   j: = n;

   if $\widehat{x_n} = 1$ then

   begin

   $\hat{C}$: = $\hat{C}$ + w_n;

   $\hat{Z}$:= $\hat{Z}$ - p_n;

   $\widehat{x_n}$:= 0

   end

5. [Backtracking]

   find i= max {k < j: $\widehat{x_k} = 1$};

   if no such i then return to 4;

   $\hat{C}$: = $\hat{C}$ + w_i;

   $\hat{Z} := \hat{Z} - p_i$;

   $\widehat{x_i} := 0$;

   j: = i + 1;

   go to 2

   end

Example 3.1 Consider the instance of KP defined by n = 7;

(p_j) = (70, 20, 39, 37, 7, 5, 10);

(w_j) = (31, 10, 20, 19, 4, 3, 6);

C = 50

By applying the above algorithm, we would have the decision tree of Figure 3.1 below.

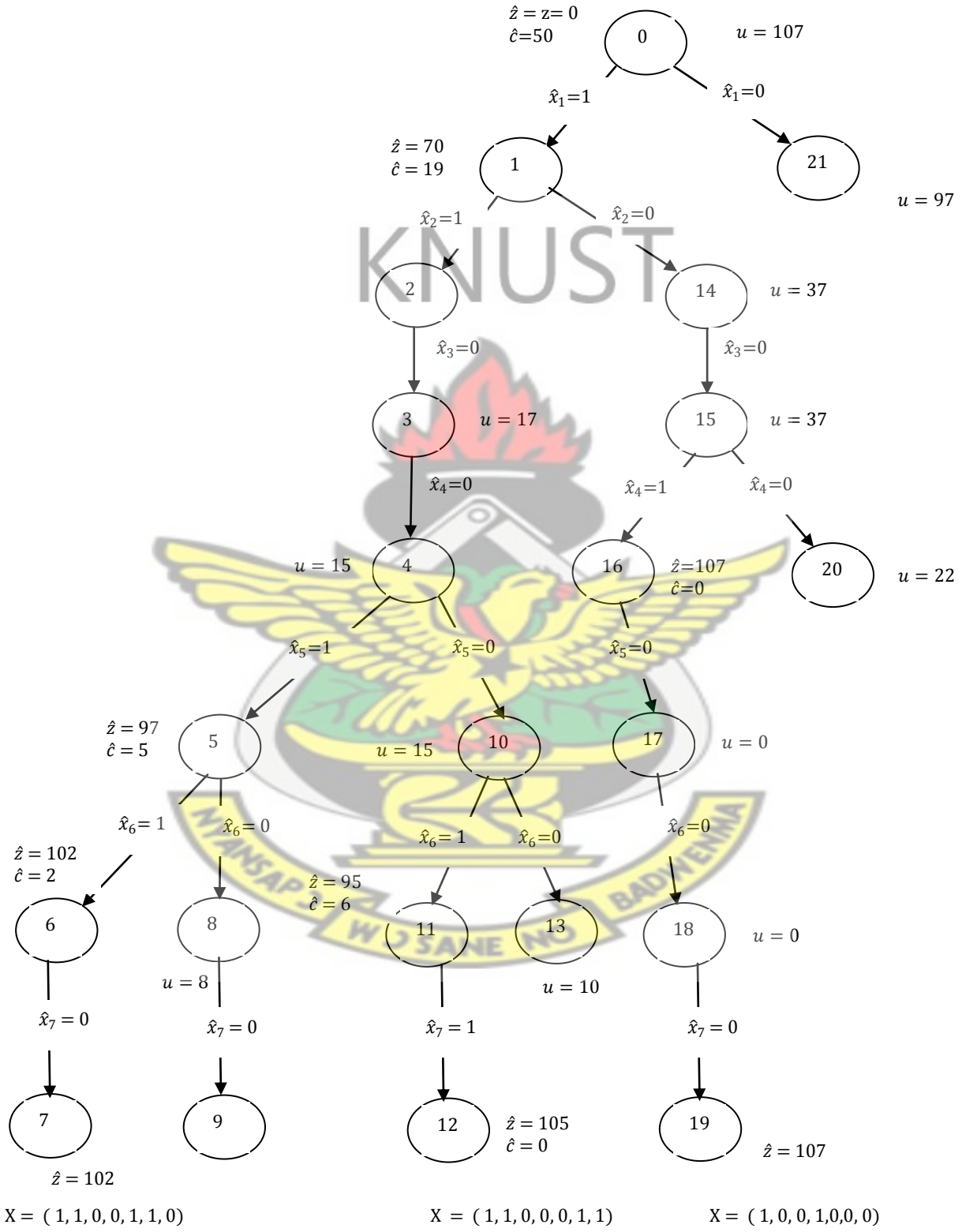


Figure 3.1 : Decision tree of Horowitz-Sahni algorithm for example 3.1.

The optimal solution of this example from the decision tree of Horowitz and Sahni algorithm is X=(1,0,0,1,0,0,0) with a value of 107 .

The Martello – Toth algorithm

Their method differs from that of Horowitz and Sahni (1974) in the following main respect (we use the notations introduced in the previous method).

(a) Upper bound U_2 is used instead of U_1

(b) The forward move associated with the selection of the j^{th} item is split into two phases: building of a new current solution and saving of the current solution. In the first phase, the largest set N_j of consecutive items which can be inserted into the current solution starting from the j^{th} is defined, and the upper bound corresponding to the insertion of the jth item is computed. If this bound is less than or equal to the value of the best solution so far, a backtracking move immediately follows. If it is greater, the second phase, that is, insertion of the items of set N_j into the current solution is performed only if the value of such new solution does not represent the maximum which can be obtained by inserting the jth item. Otherwise, the best solution so far is changed, but the current solution is not updated, so that unnecessary backtrackings on the items in N_j are avoided.

(c) A particular forward procedure, based on dominance criteria, is performed whenever, before a backtracking move on the i^{th} item, the residual capacity $\hat{C}$ does not allow insertion into the current solution of any item following the ith. The procedure is based on the following consideration;

The current solution could be improved only if the i^{th} item is replaced by an item having greater profit and a weight small enough to allow its insertion, or by at least two items having global weight not greater than $W_i + \hat{C}$. By this approach it is generally possible to eliminate most of the unnecessary nodes generated at the lowest levels of the decision – tree.

(d) The upper bounds associated with the nodes of the decision-tree are computed through a parametric technique based on the storing of information related to the current solution. Supposing the current solution has been built by inserting all the items from the j^{th} to the r^{th}: then, when performing a backtracking on one of these items (say the i^{th}, $j \leq i < r$), if no insertion occurred for the items preceding the j^{th}, it is possible to insert at least items

i + 1, … r into the new current solution. To this end, we store in $\bar{r}_i$; $\bar{p}_i$ and $\bar{w}_i$ the quantities r+1, $\sum_{k=i}^{r} P_k$ and $\sum_{k=i}^{r} w_k$, respectively, for i = j, …,r, and in $\bar{r}$ the value r – 1 (used for subsequent updating). Below is the detailed description of the algorithm.

THE ALGORITHM

Input: n, C, (P_j), (w_j);

Output: Z; (x_j);

begin

1. [Initialize]

   Z: = 0;

   $\hat{Z}$: = 0;

   $\hat{C}$: = C;

   P_{n+1}: = 0;

   w_{n+1}: = + ∞;

   for k: = 1 to n do $\widehat{x_k}$:= 0;

   compute the upper bound U = U_2 in the optimal solution value;

   $\bar{w}_1 := 0$;

   $\bar{p}_1 := 0$;

   $\bar{r}_1 := 0$;

   $\bar{r} := n$;

   for k: = n to 1 do compute m_k = min $\{w_i: i > k\}$;

   j: = 1;

2. [build a new current solution]

   while $w_j > \hat{C}$ do

   if $Z \geq \hat{Z} + [\hat{C}\ P_{j+1}/\ w_{j+1}]$ then go to 5 else j:=j+1;

   find r = min $\{i : \bar{w}_j + \sum_{k=\bar{r}_j}^{i} w_k > \hat{C}\}$;

   $P' := \bar{p}_j + \sum_{k=\bar{r}_j}^{r-1} p_k$;

   $w' := \bar{w}_j + \sum_{k=\bar{r}_j}^{r-1} w_k$;

   If r ≤ n then U:= max $([\hat{c} - w') \frac{p_{r+1}}{w_{r+1}}], [\quad p_r - \left(w_r - (\hat{c} - w') \frac{p_{r-1}}{w_{r-1}} \right])$,

   else U: = 0;

   if $Z \geq \hat{Z} + P' + U$ then go to 5;

if U = 0, then go to 4;

3. [save the current solution]

$\hat{C} := \hat{C} - w'$;

$Z \geq \hat{Z} + P'$

for k: = j to r – 1 do $\widehat{x_k} := 1$

$\bar{w}_j := w'$

$\bar{p}_j := P'$;

$\bar{r}_j := r$;

for k: = j + 1 to r – 1 do

begin

$\bar{w}_k := \bar{w}_{k-1} - w_{k-1}$;

$\bar{p}_k := \bar{p}_{k-1} - p_{k-1}$

$\bar{r}_k$: = r

end

for k: = r to $\bar{r}$ do

begin

$\bar{w}_k := 0$;

$\bar{p}_k := 0$

$\bar{r}_k := k$

end

$\bar{r} := r - 1$;

j: = r + 1;

if $\hat{C} \geq m_{j-1}$ then go to 2;

if $Z \geq \hat{Z}$ then go to 5;

$P' := 0$

4. [Update the best solution so far]

$Z \geq \hat{Z} + P'$;

for k: = 1 to j – 1 do $x_k := \widehat{x_k}$

for k: = j to r - 1 do $x_k := 1$

for k: = r to n do $x_k := 0$;

if $Z = U$ then return;

5. [Backtracking]

find i = max {k < j: $\widehat{x_k}$=1};

if no such i then return to 4;

$\hat{C}$: = $\hat{C}$ + w_i;

$\hat{Z}$ = $\hat{Z}$– p_i ;

$\widehat{x_i}$:=0

j: = i + 1;

if $\hat{C} - w_i \geq m_i$ then go to 2;

j: = i;

h: = i;

6. [try to replace item i with item h]

h: = h + 1

if $Z \geq \hat{Z} + [\hat{C}\frac{p_h}{w_h}]$ then go to 5;

if $w_h = w_i$ then go to 6

if $w_h > w_i$ then

begin

if $w_h > \hat{C}$ or $Z \geq \hat{Z} + p_h$ then go to 6;

Z: = $\hat{Z} + p_h$;

for k: = 1 to n do $x_k := \widehat{x_k}$;

x_h = 1;

if $Z = U$ then return;

i: = h;

go to 6

end

else

begin

if $\hat{C} - w_h < m_h$ then go to 6;

$\hat{C} := \hat{C} - w_h$;

$\hat{Z}$: = $\hat{Z} + p_h$;

$\widehat{x_h} := 1;$

$j := h + 1;$

$\bar{w}_h := w_h;$

$\bar{p}_h := p_h;$

$\bar{r}_h := h + 1;$

for $k := h + 1$ to $\bar{r}$ do

begin

$\bar{w}_k := 0;$

$\bar{p}_k := 0;$

$\bar{r}_k := k;$

end

$\bar{r} := h;$

go to 2

end

end

Applying the above example 3.1 to this algorithm gives the decision – tree of Figure 3.2.

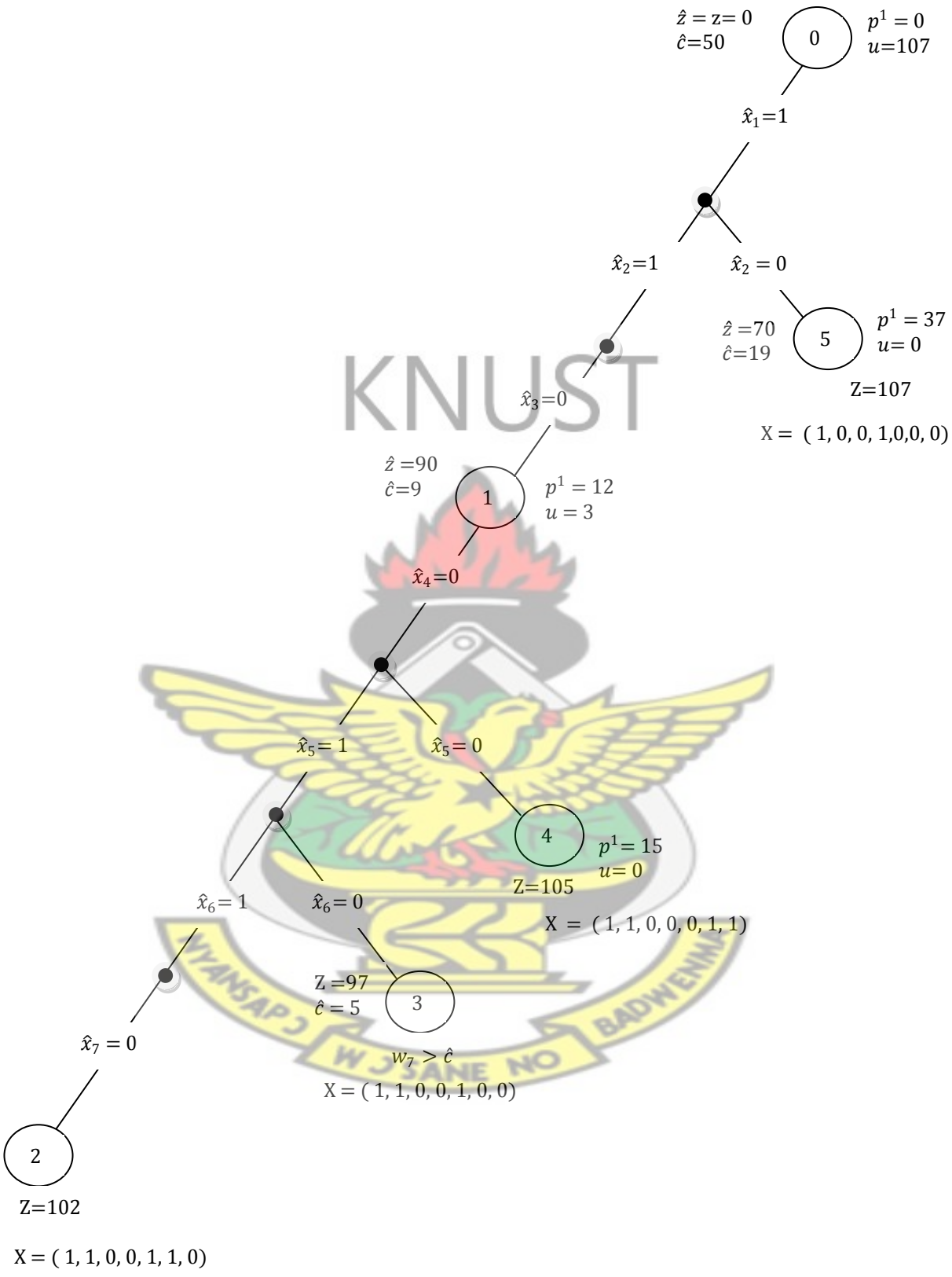


Figure 3.2 : Decision tree of Martello-Toth algorithm.

The optimal solution of this example from the decision tree of Martello and Toth algorithm is X=(1,0,0,1,0,0,0) with a value of 107 .

3.4.2 Dynamic Programming Method

Dynamic programming is a method for solving optimization problems. The idea is to compute the solutions to the sub problems once and store the solutions in a table, so that they can be reused later.

The idea of developing Dynamic programming Algorithm is as follows

Step 1: Structure: Characterize the structure of an optimal solution

Decompose the problems into smaller problems, and find a relation between the structure of the optimal solution of the original problem and the solutions of the smaller problems

Step 2: Principle of Optimality: Recursively define the value of an optimal solution.

Express the solution of the original problem in terms of optimal solutions for smaller problems.

Step 3: Bottom-up computation: Compute the value of an optimal solution in a bottom-up fashion by using a table structure.

Step 4: Construction of optimal solution: Construct an optimal solution from computed information.

Dynamic programming algorithm for Knapsacks

$A_{i,k}$: = optimum value (cost)

$L_{i,k}$: = set of indices {i} for which $x_i = 1$ when $A_{i,k}$ is attained

Require: Positive integers w_i, c_i, and k for $1 \leq i \leq n$

Final solution: A, L

for k: = 0 to K do

$A_{i,k}$: = 0

$L_{i,k}$: = { }

end for

for i: = 0 to n do

$A_{i,0} = 0$

$L_{i,0}$: = { }

for k: = 1 to K do

if $w_i \leq k$ then

if $c_i + A_{i-1,k-w_i} , > A_{i-1,k}$ then

$A_{i,k}$: = $c_i + A_{i-1,k-w_i}$

$L_{i,k}$: = $L_{i-1,k-w_i} \cup \{i\}$

else

$A_{i,k}$: = $A_{i-1,k}$

$L_{i,k}$: = $L_{i-1,k}$

end if

else

$A_{i,k}$: = $A_{i-1,k}$

$L_{i,k}$: = $L_{i-1,k}$

end if

end for

end for

A: = $A_{n,k}$

L: = $L_{n,k}$

Example 3.2 consider the example illustrated in Table 3.1 below

It consists of five different items with their corresponding weights and costs

Table 3.1: Example of Dynamic programming approach

Item (i)	1	2	3	4	5
Weight (w_i)	1	1	3	2	2
Cost (c_i)	6	11	17	3	9

Capacity of knapsack (K) =5

In this example we are to find the best set of items selection which will give a maximum cost.

Applying the above dynamic programming algorithm, the results are shown in Table 3.2 and Table 3.3

Table 3.2 (Results for maximum cost ($A_{i,k}$))

i \ k	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	11	17	17	17	17
3	0	11	17	17	28	34
4	0	11	17	17	28	34
5	0	11	17	20	28	34

Table 3.3 (Results for the set of items selected ($L_{i,k}$))

i \ k	0	1	2	3	4	5
0	{}	{}	{}	{}	{}	{}
1	{}	{1}	{1}	{1}	{1}	{1}
2	{}	{2}	{1,2}	{1,2}	{1,2}	{1,2}
3	{}	{2}	{1,2}	{1,2}	{2,3}	{1,2,3}
4	{}	{2}	{1,2}	{1,2}	{2,3}	{1,2,3}
5	{}	{2}	{1,2}	{2,5}	{2,3}	{1,2,3}

From the above, the optimum solution is $L_{5,5}$ = {1, 2, 3} with the maximum cost $A_{5,5}$= 34

$L_{5,5}$ = {1, 2, 3} which is located in Table 3.3 gives the best selection with a corresponding value of { c_1+ c_2+ c_3} = { 6+ 11+ 17} = {34} from the third row of Table 3.1, even though $L_{3,5}$ = {1, 2, 3} and $L_{4,5}$ = {1, 2, 3} also gave a maximum cost of 34, the backtracking algorithm broke the arithmetic ties arbitrarily hence the maximum cost stored in Table 3.2 will be $A_{5,5}$= 34 and that will be the final output.

CHAPTER 4

DATA COLLECTION AND ANALYSIS

4.1 DATA COLLECTION

Our research work is to find a selection approach that can determine the number of cars the workshop can repair (i.e., VW, SKODA, AUDI and VOLVO) in order to maximize revenue every month.

Premier Technik Motors carries out 10 different types of car repair jobs on each of the four brands of car in their workshop. The repair types (R_i) are: Periodic service, Brake repairs, Steering repairs, Engine repairs, Gearbox repairs, Electrical repairs, Suspension repairs, Body works, Upholstery works and Air-condition repairs. The maximum labour time available for each technician team of the workshop each month to work on these 10 different types of repair jobs for their customers is 176 hours. All the repair types have their repair time and associated labour charges respectively. Repair time for Volvo, Audi, Vw and Skoda are shown in column 3, column 4, column 5 and column 6 respectively in Table 4.1. Labour charges for Volvo, Audi, Vw and Skoda are shown in column 7, column 8, column 9 and column 10 respectively in Table 4.1

Table 4.1 below gives the details of their repair schedule.

Table 4.1: Workshop Labour charges and repair time

Repair No.	Repair type	Repair time (hrs)				Labour charge (GH¢)			
		Volvo	**Audi**	**Vw**	**Skoda**	**Volvo**	**Audi**	**Vw**	**Skoda**
1	R_1	3.00	2.00	2.00	2.00	65.00	30.00	45.00	33.00
2	R_2	5.00	4.00	4.00	4.00	180.00	100.00	90.00	180.00
3	R_3	8.00	6.00	5.00	6.00	220.00	120.00	120.00	110.00
4	R_4	22.00	18.00	16.00	18.00	300.00	220.00	110.00	300.00
5	R_5	12.00	10.00	6.00	10.00	300.00	200.00	180.00	300.00
6	R_6	3.00	8.00	6.00	5.00	120.00	150.00	75.00	80.00
7	R_7	8.00	7.00	8.00	8.00	210.00	180.00	200.00	105.00
8	R_8	40.00	30.00	24.00	24.00	600.00	450.00	400.00	495.00
9	R_9	16.00	10.00	8.00	10.00	240.00	160.00	160.00	120.00
10	R_{10}	10.00	8.00	12.00	10.00	260.00	240.00	260.00	173.00

In a month, the job scheduler receives an average of 92 work orders (cars due for repairs for all the four car brands) from corporate organizations and individuals.

Out of the 92 work orders, 19 are Volvo cars which require a total time of 245 hours to repair, 22 are Audi cars which require a total time of 220 hours to repair, 24 are Vw cars which require a total time of 205 hours to repair and 27 are Skoda cars which require a total time of 233 hours to repair. The total repair time for the car brands is computed by summing the repair time of all the repair types in column 7, column 8, column 9 and column 10 in Table 4.2 respectively. The total number of cars to repair for each car brand is computed by summing the number of cars in

column 3, column 4, column 5 and column 6 in Table 4.2 respectively. The values in columns 7,8,9 and 10 in Table 4.2 were obtained by multiplying the values of columns 3,4,5 and 6 in Table 4.2 by the corresponding values in columns 3,4,5 and 6 in Table 4.1.

The details of Table 4.2 is shown below.

Table 4.2: Average number of cars to repair in a month

Repair No.	Repair type	Number of cars to repair				Total Repair time (hrs)			
		Volvo	Audi	Vw	Skoda	Volvo	Audi	Vw	Skoda
1	R_1	2	4	3	4	6.00	8.00	6.00	8.00
2	R_2	1	1	4	1	5.00	4.00	16.00	4.00
3	R_3	1	2	1	2	8.00	12.00	5.00	12.00
4	R_4	2	3	2	1	44.00	54.00	32.00	18.00
5	R_5	3	2	1	3	36.00	20.00	6.00	30.00
6	R_6	2	1	2	3	6.00	8.00	12.00	15.00
7	R_7	3	4	3	6	24.00	28.00	24.00	48.00
8	R_8	2	2	2	2	80.00	60.00	48.00	48.00
9	R_9	1	1	4	2	16.00	10.00	32.00	20.00
10	R_{10}	2	2	2	3	20.00	16.00	24.00	30.00
	TOTAL	**19**	**22**	**24**	**27**	**245.00**	**220.00**	**205.00**	**233.00**

The problem here is to select the repair type of each car brand in such a way that the labour charges will be maximized in the workshop without exceeding the available time of 176 hours allocated for each technician team for the month.

R_1 → Periodic service

R_2	→	Brake repairs
R_3	→	Steering repairs
R_4	→	Engine repairs
R_5	→	Gearbox repairs
R_6	→	Electrical repairs
R_7	→	Suspension repairs
R_8	→	Body works
R_9	→	Upholstery work
R_{10}	→	Air-condition

4.2 FORMULATION OF PROBLEM INSTANCE

By comparing this to the knapsack model, the capacity of the bag is the time limit. The items to be considered are the different car repair types, the weight of each item is the repair time and the value of the item is the cost of repair type or Labour charge.

The problem is then modeled as:

Maximize : $V = \sum_{i=1}^{n} V_i x_i$

Subject to $\sum_{i=1}^{n} T_i x_i \leq T$

$x_i \in \{0,1\}$, $i = 1,\ldots, n$

Where V = Total Labour charge

V_i = Labour charge of each car repair type

x_i = Number of each car repair type

T_i = Time of each car repair type

T = Total available time (resource limit)

The coefficients of the objective functions below for Volvo, Audi, Vw and Skoda respectively can be obtained by using the Labour charges in Table 4.1.

The coefficients of the constraints below for Volvo, Audi, Vw and Skoda respectively can be obtained by using the Repair time in Table 4.1.

Thus, using the values in Table 4.1 and Table 4.2 and re-arranging the number of cars to repair in Table 4.2 in increasing order, the problem of the four (4) different car types would be formulated as follows ;

Volvo

Maximize $V = 180(x_1) + 220(x_2) + 240(x_3) + 65(x_4 + x_5) + 300(x_6 + x_7) + 120(x_8 + x_9) + 600(x_{10} + x_{11}) + 260(x_{12} + x_{13}) + 300(x_{14} + x_{15} + x_{16}) + 210(x_{17} + x_{18} + x_{19})$

Subject to $5(x_1) + 8(x_2) + 16(x_3) + 3(x_4 + x_5) + 22(x_6 + x_7) + 3(x_8 + x_9) + 40(x_{10} + x_{11}) + 260(x_{12} + x_{13}) + 10(x_{14} + x_{15} + x_{16}) + 8(x_{17} + x_{18} + x_{19}) \leq 176$

Audi

Maximize $V = 100(x_1) + 150(x_2) + 160(x_3) + 120(x_4 + x_5) + 200(x_6 + x_7) + 450(x_8 + x_9) + 240(x_{10} + x_{11}) + 220(x_{12} + x_{13} + x_{14}) + 30(x_{15} + x_{16} + x_{17} + x_{18}) + 180(x_{19} + x_{20} + x_{21} + x_{22})$

Subject to $4(x_1) + 8(x_2) + 10(x_3) + 6(x_4 + x_5) + 10(x_6 + x_7) + 30(x_8 + x_9) + 8(x_{10} + x_{11}) + 18(x_{12} + x_{13} + x_{14}) + 2(x_{15} + x_{16} + x_{17} + x_{18}) + 7(x_{19} + x_{20} + x_{21} + x_{22}) \leq 176$

Vw

Maximize $V = 120(x_1) + 180(x_2) + 110(x_3 + x_4) + 75(x_6 + x_7) + 400(x_7 + x_8) + 260(x_9 + x_{10}) + 45(x_{11} + x_{12} + x_{13}) + 200(x_{14} + x_{15} + x_{16}) + 90(x_{17} + x_{18} + x_{19} + x_{20}) + 160(x_{21} + x_{22} + x_{23} + x_{24})$

Subject to $5(x_1) + 6(x_2) + 16(x_3 + x_4) + 6(x_6 + x_7) + 24(x_7 + x_8) + 12(x_9 + x_{10}) + 2(x_{11} + x_{12} + x_{13}) + 8(x_{14} + x_{15} + x_{16}) + 4(x_{17} + x_{18} + x_{19} + x_{20}) + 8(x_{21} + x_{22} + x_{23} + x_{24}) \leq 176$

Skoda

Maximize $V = 180(x_1) + 300(x_2) + 110(x_3 + x_4) + 495(x_5 + x_6) + 120(x_7 + x_8) + 300(x_9 + x_{10} + x_{11}) + 80(x_{12} + x_{13} + x_{14}) + 173(x_{15} + x_{16} + x_{17}) + 33(x_{18} + x_{19} + x_{20} + x_{21}) + 105(x_{22} + x_{23} + x_{24}\ x_{25} + x_{26} + x_{27})$

Subject to $4(x_1) + 18(x_2) + 6(x_3 + x_4) + 24(x_5 + x_6) + 10(x_7 + x_8) + 10(x_9 + x_{10} + x_{11}) + 5(x_{12} + x_{13} + x_{14}) + 10(x_{15} + x_{16} + x_{17}) + 2(x_{18} + x_{19} + x_{20} + x_{21}) + 8(x_{22} + x_{23} + x_{24}\ x_{25} + x_{26} + x_{27}) \leq 176$

4.3 ALGORITHM

To carry out the iterative computation of the above model, we apply the Dynamic programming algorithm as shown below:

Notation

Let V_C : =The current solution

T_C : = The current weight

V_{BS}: = The best feasible value

T_{BS}: = The best feasible weight

T_{KN} = Available resource (Time) = z = 176 hours

a= Repair 1

b= Repair 2

c= Repair 3

d= Repair 4

e= Repair 5

f= Repair 6

g= Repair 7

h= Repair 8

i= Repair 9

j= Repair 10

(1) [Initialization]

T_{KN}:= z

T_C : = 0

V_C : = 0

T_{BS} : = 0

V_{BS} : = 0

(2) [Perform Computations]

for a= 0, maximum of a

for b = 0, maximum of b

for c = 0, maximum of c

for d = 0, maximum of d

for e = 0, maximum of e

for f = 0, maximum of f

for g = 0, maximum of g

for h = 0, maximum of h

for i = 0, maximum of i

for j = 0, maximum of j

(a) Solve the following

$V_C = j^* V_j + i^* V_i + h^* V_h + g^* V_g + f^* V_f + e^* V_e + d^* V_d + c^* V_c + b^* V_b + a^* V_a$

$T_C = j^* T_j + i^* T_i + h^* T_h + g^* T_g + f^* T_f + e^* T_e + d^* T_d + c^* T_c + b^* T_b + a^* T_a$

(b) Check for feasibility

While $T_C < T_{KN}$ and $V_{BS} < V_C$

$V_{BS} = V_C$

$T_{BS} = T_C$

end

end

(3) [Output]

Report {a, b, c, d, e, f, g, h, i, j}, V_{BS}, and T_{BS}

end for

4.4 COMPUTATION PROCEDURE

We code the dynamic programming algorithm in Fortran 90 language. A PC with a Pentium III 700 Mhz. processor is used to perform the computations. The Fortran 90 code is shown in Appendix A.

The simple feature of the software allows the data to be fixed into the code. Finally, the software displays the final optimal solution for the data received. The computational iterative values for the various optimal solutions for the four brands of car were as follows:

The number of iterations generated by the Volvo data was 7722.

The number iterations generated by the Audi data was 12929.

The number of iterations generated by the VW data was 25445.

The number of iterations generated by the Skoda data was 34654.

By application of the Bottom-up computation, the repair order for each of the cars was re-arranged in increasing order as follows:

Volvo: { R_2, R_3, R_9, R_1, R_4, R_6, R_8, R_{10}, R_5, R_7}→{1,1,1,2,2,2,2,2,3,3}

Audi : { R_2, R_6, R_9, R_3, R_5, R_8, R_{10}, R_4, R_1, R_7}→{1,1,1,2,2,2,2,3,4,4}

VW : { R_3, R_5, R_4, R_6, R_8, R_{10}, R_1, R_7, R_2, R_9}→{1,1,2,2,2,2,3,3,4,4}

Skoda: { R_2, R_4, R_3, R_8, R_9, R_5, R_6, R_{10}, R_1, R_7}→{1,1,2,2,2,3,3,3,4,6}

The summarized results of the optimal solutions for the various iterative stages of the four brands of cars are shown below.

4.5 RESULTS

The various feasible combinations of repair types to be selected for the Volvo car brand in order to achieve optimal solutions for the various iterative stages are shown in Table 4.3 in Appendix E.

At the end of the 7722th iterative stage, the algorithm displayed the best solution as GH¢ 3770 and a best time of 176 hours for the selection of {1,1,0,0,0,1,2,2,3,3} for a month.

The above solution started from iteration 5900 in the table.

Items beyond {1,1,0,0,0,1,2,2,3,3} will give $T_{BS} > T_{KN}$ which will violate the resource limit of 176 hours, hence the algorithm is completed and displays an optimal value of GH¢ 3770 as the maximum revenue that can be generated from the cars selected.

Audi

The various feasible combinations of repair types to be selected for the Audi car brand in order to achieve optimal solutions for the various iterative stages are shown in Table 4.4 in Appendix F.

At the end of the 12929th iterative stage, the algorithm displayed the best solution as GH¢ 3370 and a best time of 176 hours for the selection of {1, 1, 1, 2, 2, 2, 2, 1, 0, 4} for a month.

Items beyond {1, 1, 1, 2, 2, 2, 2, 1, 0, 4} will give $T_{BS} > T_{KN}$ which will violate the resource limit of 176 hours, hence the algorithm is completed and displays an optimal value of GH¢ 3370 as the maximum revenue that can be generated from the cars selected.

Vw

At the end of the 25445th iterative stage, the algorithm displayed the best solution as GH¢3505 and a best time of 173 hours for the selection of {1, 1, 0, 2, 2, 2, 3, 3, 4, 4} for a month.

The above solution started at iteration 21400 in the table.

Items beyond {1, 1, 0, 2, 2, 2, 3, 3, 4, 4} will give $T_{BS} > T_{KN}$ which will violate the resource limit of 173hours, hence the algorithm is completed and displays an optimal value of GH¢3505 as the maximum revenue that can be generated from the cars selected.

Skoda

The various feasible combinations of repair types to be selected for the Vw car brand in order to achieve optimal solutions for the various iterative stages are shown in Table 4.5 in Appendix G.

At the end of the 34654th iterative stage, the algorithm displayed the best solution as GH¢ 3611 and a best time of 176 hours for the selection of {1,1,2,2,0,3,2,3,4,2} for a month.

The above solution started at iteration 34000 in the table.

Items beyond {1,1,2,2,0,3,2,3,4,2} will give $T_{BS} > T_{KN}$ which will violate the resource limit of 176 hours, hence the algorithm is completed and displays an optimal value of GH¢ 3611 as the maximum revenue that can be generated from the cars selected.

4.6 DISCUSSIONS

Our scientific approach generates an initial solution and other feasible solutions for the problem and then selects the optimal solution. The optimal solution gives the solution string, the weight and the value.

From the above summarized results, we would like to compare our scientific approach results with the random selection method currently employed by the workshop.

Table 4.7 (Total revenue generated by Random selection method)

Brand of car	Number of cars received	Repair types Randomly selected	Number of cars selected	Repair time (hrs)	Optimal value (GH¢)
Volvo	19	{1,1,0,0,2,1,2,0,1,3}	11	176	3250
Audi	22	{1,1,1,1,0,2,2,3,2,2}	15	176	2990
VW	24	{1,1,2,2,2,2,3,3,2,1}	19	173	3065
Skoda	27	{0,1,1,2,1,3,2,3,4,2}	19	176	3441
Total	**92**		**64**		**12746**

Table 4.8 (Total revenue generated by Scientific approach)

Brand of car	Number of cars received	Repair types Knapsack selection	Number of cars selected	Repair time (hrs)	Optimal value (GH¢)
Volvo	19	{1,1,0,0,0,1,2,2,3,3}	13	176	3770
Audi	22	{1,1,1,2,2,2,2,1,0,4}	16	176	3370
VW	24	{1,1,0,2,2,2,3,3,4,4}	22	173	3505
Skoda	27	{1,1,2,2,0,3,2,3,4,2}	20	176	3611
Total	**92**		**71**		**14256**

From table 4.7, the total labour charges for all the four brands of car amounted to GH¢12,746.00 and the number of cars selected was 64 out of the 92 received by the workshop.

From table 4.8, the total labour charges for all the four brands of car also amounted to GH¢14,256.00 and the number of cars selected by knapsack was 71 out of the 92 received by the workshop for the different types of repairs.

The number of cars selected is the summation of the number of repair types selected as shown in columns 3 and 4 from Table 4.8.

The repair types selected by the knapsack are as follows:

Volvo: { R_2, R_3, R_6, R_8, R_{10}, R_5, R_7}→{1,1,1, 2,2,3,3}

Audi : { R_2, R_6, R_9, R_3, R_5, R_8, R_{10}, R_4, R_7}→{1,1,1,2,2,2,2,3,1,4}

VW : { R_3, R_5, R_6, R_8, R_{10}, R_1, R_7, R_2, R_9}→{1,1,2,2,2,3,3,4,4}

Skoda: { R_2, R_4, R_3, R_8, R_5, R_6, R_{10}, R_1, R_7}→{1,1,2,2,3,2,3,4, 2}

These figures shown by our scientific approach are higher as compared with the results of the random selection method used by the workshop Job scheduler.

Hence our approach is the best out of the two and as a result the old revenue would increase by an amount of GH¢1,510.00 per month and the intake of cars can also increase from 64 to 71 as shown in the two tables above.

From the various optimal solutions tables, that is Table 4.3, Table 4.4 and Table 4.6, some solutions were generated at 176 hours but these were not selected by the program. This was because their optimal values were less than that of the solutions selected finally by the program. The program was to output the best or maximum optimal values at a best time.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

This thesis seeks to solve a real life problem of an Auto firm known as Premier Technik Motors in Ghana by modeling their problem as a 0-1 knapsack problem. Our first objective was to model this real life problem of maximization as a knapsack problem; this was modeled using the data collected as shown in Chapter 4.

Our second objective was to solve the knapsack problem using Dynamic programming and Fortran 90. At the end of the various computations, it was discovered that the solution that gave the most desirable maximum value was our scientific approach. This means that for a maximum value to be achieved every month (on the average), the number of cars to be selected and their repair types are 13 volvos with repair types R_2, R_3, R_6, R_8, R_{10}, R_5 and R_7 , 16 Audi's with repair types R_2, R_6, R_9, R_3, R_5, R_8, R_{10}, R_4 and R_7, 22 Vws with repair types R_3, R_5, R_6, R_8, R_{10}, R_1, R_7, R_2 and R_9 and finally 20 Skodas with repair types R_2, R_4, R_3, R_8, R_5, R_6, R_{10}, R_1 and R_7 .

The other findings were that, the use of this scientific approach is transparent as compared with the random selection. From this approach, an amount of GH¢ 14,256.00 would be obtained from the selection of 71 cars from the four brands of car according to the type of repairs. However, the random selection method used by the company also yields a maximum value of GH¢ 12,746.00 with the selection of 64 cars. Hence, higher returns can be achieved by this workshop by the use of this scientific approach.

Secondly, this approach can select the optimal number of cars to be repaired for each month whenever there is an over-booking as a result of high demand of service from

customers. This can help the company to reschedule some of the cars whose problems are not too serious and urgent to be repaired or they can sublet them to other workshops.

5.2 Recommendations

In order to achieve the most desirable maximum revenue from the repairs of cars in their workshop, we recommend that Premier Technik Motors should adopt this knapsack problem model approach which is more scientific and transparent to determine the number of cars that must be selected from each of the four types of cars whenever they encounter time constraint problems.

Service managers and workshop managers of other automobile companies in Ghana can also benefit from this proposed scientific approach for selecting the right cars at the right time to be repaired in their workshops in order to maximize revenue within the maximum working hours.

Finally, this approach from our research can be applied to problems that can be modeled as a 0/1 knapsack problem.

REFERENCES

1. African Journal of Mathematics and Computer Science Research Vol. 4(4), pp. 170-176, April 2011 ISSN 2006-9731
2. Akinc, U. (2006). Approximate and exact algorithms for the fixed-charge knapsack problem at www.sciencedirect.com
3. Amponsah, S.K and Darkwah, K.F.(2009). Lecture notes on Operational Research, Kwame Nkrumah University of Science and Technology, Kumasi-GHANA.
4. Balev,S., Yanev, N., Freville, A. and Andonov, R. (2008). A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem.
5. Beasley, J.E. and Chu, P.C. (1996). A genetic Algorithm for the set covering problem. Eur.J. Operations Research. 94: 392-404
6. Captivo , E.M., Climaco, J., Figueira, J., Martins, E. and Santos, J.L. (2003). Solving bicriteria 0-1 knapsack problems using a labeling algorithm.
7. Chang, J.T., Meade,N., Beasley, J.E. and Sharaiha, Y.M.(2000). Heuristics for cardinality constrained portfolio optimization. Comp. Operations.Research. 27:1271-1302
8. Chu, P.C and Beasley, J.E.(1998b). Constraint handling in genetic algorithm: the set partitioning problem. Journal Heuristics 4:323-357
9. Figueira, J.R., Tavares, G., Wiecek, M.M. (2009). Labelling algorithms for multiple objective integer knapsack problems.
10. Gilmore, P.C. and Gomory, R.E. (1963). "A Linear Programming Approach to the cutting Stock problem II" Operations Research., 11:863-888
11. Gilmore, P.C. and Gomory, R.E. (1965). "Multi-Stage Cutting Stock problems of Two and More Dimensions". Operations Research., 13, 94-120

12. Gilmore, P.C. and Gomory, R.E.(1966). "The theory and Computation of Knapsack Functions". Operations Research., 14:1045-1074
13. Greenbrg,H. and Hagerich, R.L.(1970). " A branch Search Algorithm for Knapsack Problem". Manage Sci., 16:327-332
14. Ingargiola, G.P. and Korsh, J.F.A. (1973). "Reduction Algorithm for Zero-One Single Knapsack Problems" Manage. Sci., 20, 460-463
15. Kaplan, S.(1996). " Solution of the Lorie-savage and similar Integer Programming Problems by the Generalized Lagrange Multiplier Method", Operations Research., 14, 1130-1136
16. Kostas, F.,Mavrotas, G. and Diakoulaki,D.(2009). Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms.
17. Martello, S., Pisinger, D. and Toth, P. (2000). New trends in exact algorithms for the 0-1 knapsack problem.
18. Martello,S. and Toth,P. Knapsack Problems, Algorithms and Computer Implementations.
19. Martin,B.(2004). Combinatorial Aspects of Yield Management, a Reinforcement Learning Approach. Retrieved from http://www.brunomartin.org/YM.html
20. Munapo,E.(2008). "The efficiency enhanced branch and bound algorithm for the knapsack model" Adv.Appl. Math. Anal. ISSN 0973-5313, 3(1): 81-89
21. Pendharkar, P.C. and Rodger, J.A.(2006). Information Technology Capital Budgeting (ITCB) problem. Int.Trans. Operation Research. 13:333-351
22. Pferschy, U., Pisinger, D. and Woeginger, G.J. (1997). Simple but efficient approaches for the collapsing knapsack problem at www.sciencedirect.com
23. Pisinger, D. (1995). An expanding-core algorithm for the exact 0-1 knapsack problem

24. Realff M.J., Kvam, P.H. and Taylor, W.E.(1999) Combined analytical and empirical learning framework for branch and bound algorithms: the knapsack problem at www.sciencedirect.com
25. Research Journal of Information Technology 3(1): 49-54, 2011, S.K Amponsah, ISSN: 2041-3114
26. Taniguchi, F., Yamada, T. and Kataoka, S. (2008). Heuristic and exact algorithms for the max-min optimization of the multi-scenario knapsack problem.
27. Yamada, T., Futakawa, M. and Kataoka,S. (1998). Some exact algorithms for the knapsack sharing problem at www.sciencedirect.com

APPENDIX A

Program Knapsack_volvo

IMPLICIT NONE

Real:: TotalWeight

Integer:: Max R_2, Max R_3, Max R_9 Max R_1, Max R_4, Max R_6, Max R_8, Max R_{10}, Max R_5, Max R_7,

MaxValue = 0

Integer:: a, b, c, d, e, f, g, h, i, j, n (10)

Type Bounty

Integer:: Val

Real:: Wht

End Type Bounty

Type (Bounty) :: R_2, R_3, R_9, R_1, R_4, R_6, R_8, R_{10}, R_5, R_7, Sack, Current

R_2 = Bounty (5, 180)

R_3 = Bounty (8, 220)

R_9 = Bounty (16, 240)

R_1 = Bounty (3, 65)

R_4 = Bounty (22, 300)

R_6 = Bounty (3, 120)

R_8 = Bounty (40, 600)

R_{10} = Bounty (10, 260)

R_5 = Bounty (12, 300)

R_7 = Bounty (8, 210)

Sack = Bounty (0, 176)

Max R_2 = 1**,** Max R_3 = 1**,** Max R_9 = 1**,** Max R_1 = 2, Max R_4 = 2**,** Max R_6 = 2**,** Max R_8 = 2**,** Max R_{10} = 2, Max R_5 = 3**,** Max R_7 = 3

Do a = 0, MaxR_2

Do b = 0, MaxR_3

Do c = 0, MaxR_9

Do d = 0, MaxR_1

Do e = 0, MaxR_4

Do f = 0, MaxR_6

Do g = 0, MaxR_8

Do h = 0, MaxR_{10}

Do i = 0, MaxR_5

Do j = 0, MaxR_7

Current% Val = j* R_7% Val + i* R_5% Val + h* R_{10}% Val + g* R_8% Val + f* R_6% Val + e* R_4% Val + d* R_1% Val + c* R_9% Val + b* R_3% Val + a* R_2% Val

Current% Wht = j* R_7% Wht + i* R_5% Wht + h* R_{10}% Wht + g* R_8% Wht + f* R_6% Wht + e* R_4% Wht + d* R_1% Wht + c* R_9% Wht + b* R_3% Wht + a* R_2% Wht

If (Current% Wht < Sack% Wht) Then

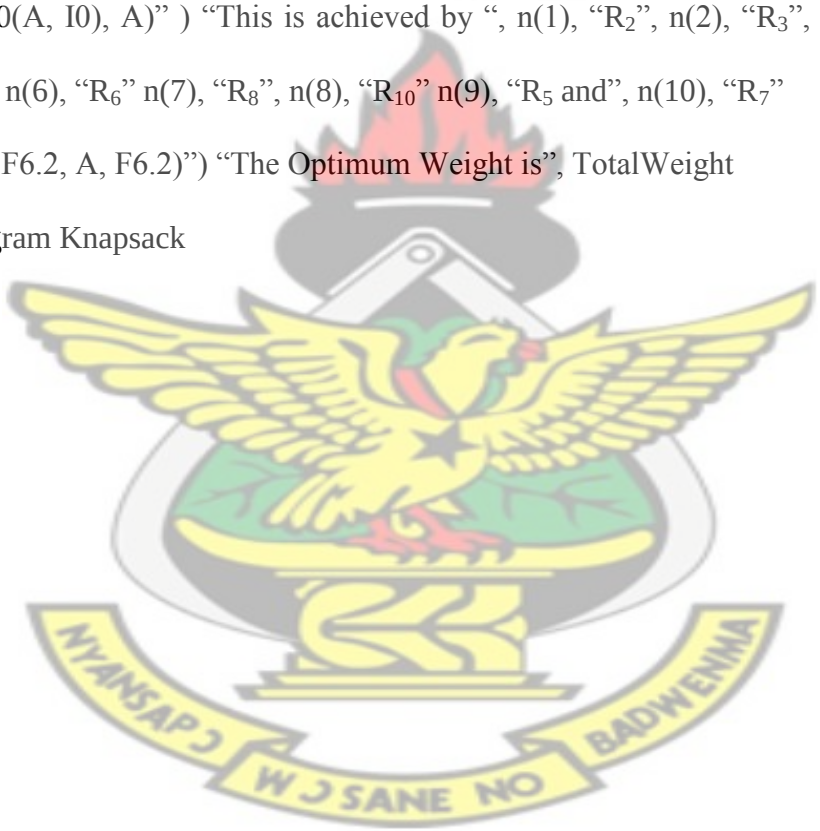
If (MaxValue < Current% Val) Then

MaxValue = Current% Val

TotalWeight = Current% Wht

n (1) = a, n (2) = b, n (3) = c, n(4) = d, n (5) = e, n (6) = f, n (7) = g, n(8) = h, n (9) = i, n (10) = j

```
        End If
End If
            End Do
        End Do
    End Do
End Do
WRITE (*, "(A, I0) ") "Optimum Value achievable is ", MaxValue
WRITE (*, "( 10(A, I0), A)" ) "This is achieved by ", n(1), "R2", n(2), "R3", n(3), "R9", n(4),
"R1," n(5), "R4", n(6), "R6" n(7), "R8", n(8), "R10" n(9), "R5 and", n(10), "R7"
WRITE (*, "(A, F6.2, A, F6.2)") "The Optimum Weight is", TotalWeight
        End Program Knapsack
```

APPENDIX B

Program Knapsack_Audi

IMPLICIT NONE

Real:: TotalWeight

Integer:: Max R_2, Max R_6, Max R_9 Max R_3, Max R_5, Max R_8, Max R_{10}, Max R_4, Max R_1, Max R_7,

MaxValue = 0

Integer:: a, b, c, d, e, f, g, h, i, j, n (10)

Type Bounty

Integer:: Val

Real:: Wht

End Type Bounty

Type (Bounty) :: R_2, R_6, R_9, R_3, R_5, R_8, R_{10}, R_4, R_1, R_7, Sack, Current

R_2 = Bounty (4, 100)

R_6 = Bounty (8, 150)

R_9 = Bounty (10, 160)

R_3 = Bounty (6, 120)

R_5 = Bounty (10, 200)

R_8 = Bounty (30, 450)

R_{10} = Bounty (8, 240)

R_4 = Bounty (18, 220)

R_1 = Bounty (2, 30)

R_7 = Bounty (7, 180)

Sack = Bounty (0, 176)

Max R_2 = 1**,** Max R_6 = 1**,** Max R_9 = 1**,** Max R_3 = 2, Max R_5 = 2**,** Max R_8 = 2**,** Max R_{10} = 2**,**

Max R_4 = 3, Max R_1 = 4**,** Max R_7 = 4

Do a = 0, MaxR_2

Do b = 0, MaxR_6

Do c = 0, MaxR_9

Do d = 0, MaxR_3

Do e = 0, MaxR_5

Do f = 0, MaxR_8

Do g = 0, MaxR_{10}

Do h = 0, MaxR_4

Do i = 0, MaxR_1

Do j = 0, MaxR_7

Current% Val = j* R_7% Val + i* R_1% Val + h* R_4% Val + g* R_{10}% Val + f* R_8% Val + e* R_5% Val + d* R_3% Val + c* R_9% Val + b* R_6% Val + a* R_2% Val

Current% Wht = j* R_7% Wht + i* R_1% Wht + h* R_4% Wht + g* R_{10}% Wht + f* R_8% Wht + e* R_5% Wht + d* R_3% Wht + c* R_9% Wht + b* R_6% Wht + a* R_2% Wht

If (Current% Wht < Sack% Wht) Then

If (MaxValue < Current% Val) Then

MaxValue = Current% Val

TotalWeight = Current% Wht

n (1) = a, n (2) = b, n (3) = c, n(4) = d, n (5) = e, n (6) = f, n (7) = g, n(8) = h, n (9) = i, n (10) = j

```
        End If
End If
            End Do
        End Do
    End Do
End Do
WRITE (*, "(A, I0) ") "Optimum Value achievable is ", MaxValue
WRITE (*, "( 10(A, I0), A)" ) "This is achieved by ", n(1), "R2", n(2), "R6", n(3), "R9", n(4),
"R3," n(5), "R5", n(6), "R8" n(7), "R10", n(8), "R4" n(9), "R1 and", n(10), "R7"
WRITE (*, "(A, F6.2, A, F6.2)") "The Optimum Weight is", TotalWeight

End Program Knapsack
```

APPENDIX C

```
Program Knapsack_vw

IMPLICIT NONE

Real:: TotalWeight

Integer:: Max R3, Max R5, Max R4 Max R6, Max R8, Max R10, Max R1, Max R7, Max R2, Max R9,

MaxValue = 0

Integer:: a, b, c, d, e, f, g, h, i, j, n (10)

Type Bounty

   Integer:: Val

   Real:: Wht

End Type Bounty

Type (Bounty) :: R3, R5, R4, R6, R8, R10, R1, R7, R2, R9, Sack, Current

R3 = Bounty (5, 120)

R5 = Bounty (6, 180)

R4 = Bounty (16, 110)

R6 = Bounty (6, 75)

R8 = Bounty (24, 400)

R10 = Bounty (12, 260)

R1 = Bounty (2, 45)

R7 = Bounty (8, 200)

R2 = Bounty (4, 90)

R9 = Bounty (8, 160)
```

Sack = Bounty (0, 176)

Max R_3 = 1**,** Max R_5 = 1**,** Max R_4 = 2**,** Max R_6 = 2, Max R_8 = 2**,** Max R_{10} = 2**,** Max R_1 = 3**,**

Max R_7 = 3, Max R_2 = 4**,** Max R_9 = 4

Do a = 0, MaxR_3

Do b = 0, MaxR_5

Do c = 0, MaxR_4

Do d = 0, MaxR_6

Do e = 0, MaxR_8

Do f = 0, MaxR_{10}

Do g = 0, MaxR_1

Do h = 0, MaxR_7

Do i = 0, MaxR_2

Do j = 0, MaxR_9

Current% Val = j* R_9% Val + i* R_2% Val + h* R_7% Val + g* R_1% Val + f* R_{10}% Val + e* R_8% Val + d* R_6% Val + c* R_4% Val + b* R_5% Val + a* R_3% Val

Current% Wht = j* R_9% Wht + i* R_2% Wht + h* R_7% Wht + g* R_1% Wht + f* R_{10}% Wht + e* R_8% Wht + d* R_6% Wht + c* R_4% Wht + b* R_5% Wht + a* R_3% Wht

If (Current% Wht < Sack% Wht) Then

If (MaxValue < Current% Val) Then

MaxValue = Current% Val

TotalWeight = Current% Wht

n (1) = a, n (2) = b, n (3) = c, n(4) = d, n (5) = e, n (6) = f, n (7) = g, n(8) = h, n (9) = i, n (10) = j

```
        End If
End If
            End Do
        End Do
    End Do
End Do
WRITE (*, "(A, I0) ") "Optimum Value achievable is ", MaxValue
WRITE (*, "( 10(A, I0), A)" ) "This is achieved by ", n(1), "R3", n(2), "R5", n(3), "R4", n(4),
"R6," n(5), "R8", n(6), "R10" n(7), "R1", n(8), "R7" n(9), "R2 and", n(10), "R9"
WRITE (*, "(A, F6.2, A, F6.2)") "The Optimum Weight is", TotalWeight

End Program Knapsack
```

APPENDIX D

Program Knapsack_skoda

IMPLICIT NONE

Real:: TotalWeight

Integer:: Max R_2, Max R_4, Max R_3 Max R_8, Max R_9, Max R_5, Max R_6, Max R_{10}, Max R_1, Max R_7,

MaxValue = 0

Integer:: a, b, c, d, e, f, g, h, i, j, n (10)

Type Bounty

Integer:: Val

Real:: Wht

End Type Bounty

Type (Bounty) :: R_2, R_4, R_3, R_8, R_9, R_5, R_6, R_{10}, R_1, R_7, Sack, Current

R_2 = Bounty (4, 180)

R_4 = Bounty (18, 300)

R_3 = Bounty (6, 110)

R_8 = Bounty (24, 495)

R_9 = Bounty (10, 120)

R_5 = Bounty (10, 300)

R_6 = Bounty (5, 80)

R_{10} = Bounty (10, 173)

R_1 = Bounty (2, 33)

R_7 = Bounty (8, 105)

Sack = Bounty (0, 176)

Max R_2 = 1**,** Max R_4 = 1**,** Max R_3 = 2**,** Max R_8 = 2, Max R_9 = 2**,** Max R_5 = 3**,** Max R_6 = 3**,** Max R_{10} = 3, Max R_1 = 4**,** Max R_7 = 6

Do a = 0, MaxR_2

Do b = 0, MaxR_4

Do c = 0, MaxR_3

Do d = 0, MaxR_8

Do e = 0, MaxR_9

Do f = 0, MaxR_5

Do g = 0, MaxR_6

Do h = 0, MaxR_{10}

Do i = 0, MaxR_1

Do j = 0, MaxR_7

Current% Val = j* R_7% Val + i* R_1% Val + h* R_{10}% Val + g* R_6% Val + f* R_5% Val + e* R_9% Val + d* R_8% Val + c* R_3% Val + b* R_4% Val + a* R_2% Val

Current% Wht = j* R_7% Wht + i* R_1% Wht + h* R_{10}% Wht + g* R_6% Wht + f* R_5% Wht + e* R_9% Wht + d* R_8% Wht + c* R_3% Wht + b* R_4% Wht + a* R_2% Wht

If (Current% Wht < Sack% Wht) Then

If (MaxValue < Current% Val) Then

MaxValue = Current% Val

TotalWeight = Current% Wht

n (1) = a, n (2) = b, n (3) = c, n(4) = d, n (5) = e, n (6) = f, n (7) = g, n(8) = h, n (9) = i, n (10) = j

```
        End If
End If
            End Do
        End Do
    End Do
End Do
WRITE (*, "(A, I0) ") "Optimum Value achievable is ", MaxValue
WRITE (*, "( 10(A, I0), A)" ) "This is achieved by ", n(1), "R2", n(2), "R4", n(3), "R3", n(4),
"R8," n(5), "R9", n(6), "R5" n(7), "R6", n(8), "R10" n(9), "R1 and", n(10), "R7"
WRITE (*, "(A, F6.2, A, F6.2)") "The Optimum Weight is", TotalWeight

End Program Knapsack
```

APPENDIX E

Table 4.3: Optimal Solutions of Volvo cars for the various iterative stages

Iteration	Item selected	Optimal weight (hrs)	Optimal value (GH¢)
1	{0,0,0,0,0,0,0,0,0,3}	24	630
100	{0,0,0,0,0,1,2,2,3,3}	163	3370
200	{0,0,0,0,0,1,2,2,3,3}	166	3490
300	{0,0,0,0,0,1,2,2,3,3}	166	3490
400	{0,0,0,1,0,0,2,0,3,3}	176	3120
500	{0,0,0,1,0,2,2,2,3,3}	169	3555
600	{0,0,0,1,0,2,2,2,3,3}	169	3555
700	{0,0,0,1,0,2,2,2,3,3}	169	3555
800	{0,0,0,2,0,2,2,2,3,3}	172	3620
900	{0,0,0,2,0,2,2,2,3,3}	172	3620
1000	{0,0,0,2,0,2,2,2,3,3}	172	3620
1100	{0,0,0,2,0,2,2,2,3,3}	172	3620
1200	{0,0,0,2,0,2,2,2,3,3}	172	3620
1300	{0,0,0,2,0,2,2,2,3,3}	172	3620
1400	{0,0,0,2,0,2,2,2,3,3}	172	3620
1500	{0,0,0,2,0,2,2,2,3,3}	172	3620
1600	{0,0,0,2,0,2,2,2,3,3}	172	3620
1700	{0,0,0,2,0,2,2,2,3,3}	172	3620
1800	{0,0,0,2,0,2,2,2,3,3}	172	3620
1900	{0,0,0,2,0,2,2,2,3,3}	172	3620
2000	{0,1,0,0,0,2,2,2,3,3}	174	3710
2100	{0,1,0,0,0,2,2,2,3,3}	174	3710
2200	{0,1,0,0,0,2,2,2,3,3}	174	3710
2300	{0,1,0,0,0,2,2,2,3,3}	174	3710
2400	{0,1,0,0,0,2,2,2,3,3}	174	3710
2500	{0,1,0,0,0,2,2,2,3,3}	174	3710
2600	{0,1,0,0,0,2,2,2,3,3}	174	3710
2700	{0,1,0,0,0,2,2,2,3,3}	174	3710
2800	{0,1,0,0,0,2,2,2,3,3}	174	3710
2900	{0,1,0,0,0,2,2,2,3,3}	174	3710
3000	{0,1,0,0,0,2,2,2,3,3}	174	3710
3100	{0,1,0,0,0,2,2,2,3,3}	174	3710

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
3200	{0,1,0,0,0,2,2,2,3,3}	174	3710
3300	{0,1,0,0,0,2,2,2,3,3}	174	3710
3400	{0,1,1,1,1,2,1,2,3,3}	175	3715
3500	{0,1,1,1,1,2,1,2,3,3}	175	3715
3600	{0,1,1,1,1,2,1,2,3,3}	175	3715
3700	{0,1,1,1,1,2,1,2,3,3}	175	3715
3800	{0,1,1,1,1,2,1,2,3,3}	175	3715
3900	{0,1,1,1,1,2,1,2,3,3}	175	3715
4000	{0,1,1,1,1,2,1,2,3,3}	175	3715
4100	{0,1,1,1,1,2,1,2,3,3}	175	3715
4200	{0,1,1,1,1,2,1,2,3,3}	175	3715
4300	{1,0,0,1,0,2,2,2,3,3}	174	3735
4400	{1,0,0,1,0,2,2,2,3,3}	174	3735
4500	{1,0,0,1,0,2,2,2,3,3}	174	3735
4600	{1,0,0,1,0,2,2,2,3,3}	174	3735
4700	{1,0,0,1,0,2,2,2,3,3}	174	3735
4800	{1,0,0,1,0,2,2,2,3,3}	174	3735
4900	{1,0,0,1,0,2,2,2,3,3}	174	3735
5000	{1,0,0,1,0,2,2,2,3,3}	174	3735
5100	{1,0,0,1,0,2,2,2,3,3}	174	3735
5200	{1,0,0,1,0,2,2,2,3,3}	174	3735
5300	{1,0,0,1,0,2,2,2,3,3}	174	3735
5400	{1,0,0,1,0,2,2,2,3,3}	174	3735
5500	{1,0,0,1,0,2,2,2,3,3}	174	3735
5600	{1,0,0,1,0,2,2,2,3,3}	174	3735
5700	{1,0,1,2,1,2,1,2,3,3}	175	3740
5800	{1,0,1,2,1,2,1,2,3,3}	175	3740
5900	{1,1,0,0,0,1,2,2,3,3}	176	3770
6000	{1,1,0,0,0,1,2,2,3,3}	176	3770
6100	{1,1,0,0,0,1,2,2,3,3}	176	3770
6200	{1,1,0,0,0,1,2,2,3,3}	176	3770
6300	{1,1,0,0,0,1,2,2,3,3}	176	3770
6400	{1,1,0,0,0,1,2,2,3,3}	176	3770
6500	{1,1,0,0,0,1,2,2,3,3}	176	3770
6600	{1,1,0,0,0,1,2,2,3,3}	176	3770

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
6700	{1,1,0,0,0,1,2,2,3,3}	176	3770
6800	{1,1,0,0,0,1,2,2,3,3}	176	3770
6900	{1,1,0,0,0,1,2,2,3,3}	176	3770
7000	{1,1,0,0,0,1,2,2,3,3}	176	3770
7100	{1,1,0,0,0,1,2,2,3,3}	176	3770
7200	{1,1,0,0,0,1,2,2,3,3}	176	3770
7300	{1,1,0,0,0,1,2,2,3,3}	176	3770
7400	{1,1,0,0,0,1,2,2,3,3}	176	3770
7500	{1,1,0,0,0,1,2,2,3,3}	176	3770
7600	{1,1,0,0,0,1,2,2,3,3}	176	3770
7700	{1,1,0,0,0,1,2,2,3,3}	176	3770
7722	{1,1,0,0,0,1,2,2,3,3}	176	3770

APPENDIX F

Table 4.4: Optimal Solutions of Audi cars for the various iterative stages

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
1	{0,0,0,0,0,0,0,0,0,4}	28	720
100	{0,0,0,0,0,1,1,3,4,4}	128	2190
200	{0,0,0,0,0,2,2,3,4,4}	166	2880
300	{0,0,0,0,0,2,2,3,4,4}	166	2880
400	{0,0,0,0,1,2,2,3,4,4}	176	3080
500	{0,0,0,0,1,2,2,3,4,4}	176	3080
600	{0,0,0,0,1,2,2,3,4,4}	176	3080
700	{0,0,0,0,1,2,2,3,4,4}	176	3080
800	{0,0,0,0,1,2,2,3,4,4}	176	3080
900	{0,0,0,1,1,2,2,3,1,4}	176	3110
1000	{0,0,0,1,1,2,2,3,1,4}	176	3110
1100	{0,0,0,1,2,2,2,2,4,4}	174	3180
1200	{0,0,0,1,2,2,2,2,4,4}	174	3180
1300	{0,0,0,1,2,2,2,2,4,4}	174	3180
1400	{0,0,0,1,2,2,2,2,4,4}	174	3180
1500	{0,0,0,1,2,2,2,2,4,4}	174	3180
1600	{0,0,0,1,2,2,2,2,4,4}	174	3180
1700	{0,0,0,2,2,2,2,2,2,4}	176	3240
1800	{0,0,0,2,2,2,2,2,2,4}	176	3240
1900	{0,0,0,2,2,2,2,2,2,4}	176	3240
2000	{0,0,0,2,2,2,2,2,2,4}	176	3240
2100	{0,0,0,2,2,2,2,2,2,4}	176	3240
2200	{0,0,0,2,2,2,2,2,2,4}	176	3240
2300	{0,0,0,2,2,2,2,2,2,4}	176	3240
2400	{0,0,0,2,2,2,2,2,2,4}	176	3240
2500	{0,0,0,2,2,2,2,2,2,4}	176	3240
2600	{0,0,0,2,2,2,2,2,2,4}	176	3240
2700	{0,0,0,2,2,2,2,2,2,4}	176	3240
2800	{0,0,0,2,2,2,2,2,2,4}	176	3240
2900	{0,0,0,2,2,2,2,2,2,4}	176	3240
3000	{0,0,0,2,2,2,2,2,2,4}	176	3240
3100	{0,0,0,2,2,2,2,2,2,4}	176	3240
3200	{0,0,0,2,2,2,2,2,2,4}	176	3240
3300	{0,0,0,2,2,2,2,2,2,4}	176	3240
3400	{0,0,0,2,2,2,2,2,2,4}	176	3240

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
3500	{0,0,0,2,2,2,2,2,2,4}	176	3240
3600	{0,0,0,2,2,2,2,2,2,4}	176	3240
3700	{0,0,0,2,2,2,2,2,2,4}	176	3240
3800	{0,0,0,2,2,2,2,2,2,4}	176	3240
3900	{0,0,0,2,2,2,2,2,2,4}	176	3240
4000	{0,0,0,2,2,2,2,2,2,4}	176	3240
4100	{0,0,0,2,2,2,2,2,2,4}	176	3240
4200	{0,0,0,2,2,2,2,2,2,4}	176	3240
4300	{0,0,0,2,2,2,2,2,2,4}	176	3240
4400	{0,0,0,2,2,2,2,2,2,4}	176	3240
4500	{0,0,0,2,2,2,2,2,2,4}	176	3240
4600	{0,0,0,2,2,2,2,2,2,4}	176	3240
4700	{0,0,0,2,2,2,2,2,2,4}	176	3240
4800	{0,0,0,2,2,2,2,2,2,4}	176	3240
4900	{0,0,0,2,2,2,2,2,2,4}	176	3240
5000	{0,0,0,2,2,2,2,2,2,4}	176	3240
5100	{0,0,0,2,2,2,2,2,2,4}	176	3240
5200	{0,0,0,2,2,2,2,2,2,4}	176	3240
5300	{0,0,0,2,2,2,2,2,2,4}	176	3240
5400	{0,0,0,2,2,2,2,2,2,4}	176	3240
5500	{0,0,0,2,2,2,2,2,2,4}	176	3240
5600	{0,0,0,2,2,2,2,2,2,4}	176	3240
5700	{0,0,0,2,2,2,2,2,2,4}	176	3240
5800	{0,0,0,2,2,2,2,2,2,4}	176	3240
5900	{0,0,0,2,2,2,2,2,2,4}	176	3240
6000	{0,1,1,1,2,2,2,1,4,4}	174	3270
6100	{0,1,1,1,2,2,2,1,4,4}	174	3270
6200	{0,1,1,1,2,2,2,1,4,4}	174	3270
6300	{0,1,1,1,2,2,2,1,4,4}	174	3270
6400	{0,1,1,1,2,2,2,1,4,4}	174	3270
6500	{0,1,1,2,2,2,2,1,2,4}	176	3330
6600	{0,1,1,2,2,2,2,1,2,4}	176	3330
6700	{0,1,1,2,2,2,2,1,2,4}	176	3330
6800	{0,1,1,2,2,2,2,1,2,4}	176	3330
6900	{0,1,1,2,2,2,2,1,2,4}	176	3330
7000	{0,1,1,2,2,2,2,1,2,4}	176	3330
7100	{0,1,1,2,2,2,2,1,2,4}	176	3330
7200	{0,1,1,2,2,2,2,1,2,4}	176	3330

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
7300	{0,1,1,2,2,2,2,1,2,4}	176	3330
7400	{0,1,1,2,2,2,2,1,2,4}	176	3330
7500	{0,1,1,2,2,2,2,1,2,4}	176	3330
7600	{0,1,1,2,2,2,2,1,2,4}	176	3330
7700	{0,1,1,2,2,2,2,1,2,4}	176	3330
7800	{0,1,1,2,2,2,2,1,2,4}	176	3330
7900	{0,1,1,2,2,2,2,1,2,4}	176	3330
8000	{0,1,1,2,2,2,2,1,2,4}	176	3330
8100	{0,1,1,2,2,2,2,1,2,4}	176	3330
8200	{0,1,1,2,2,2,2,1,2,4}	176	3330
8300	{0,1,1,2,2,2,2,1,2,4}	176	3330
8400	{0,1,1,2,2,2,2,1,2,4}	176	3330
8500	{0,1,1,2,2,2,2,1,2,4}	176	3330
8600	{0,1,1,2,2,2,2,1,2,4}	176	3330
8700	{0,1,1,2,2,2,2,1,2,4}	176	3330
8800	{0,1,1,2,2,2,2,1,2,4}	176	3330
8900	{0,1,1,2,2,2,2,1,2,4}	176	3330
9000	{0,1,1,2,2,2,2,1,2,4}	176	3330
9100	{0,1,1,2,2,2,2,1,2,4}	176	3330
9200	{0,1,1,2,2,2,2,1,2,4}	176	3330
9300	{0,1,1,2,2,2,2,1,2,4}	176	3330
9400	{0,1,1,2,2,2,2,1,2,4}	176	3330
9500	{0,1,1,2,2,2,2,1,2,4}	176	3330
9600	{0,1,1,2,2,2,2,1,2,4}	176	3330
9700	{1,0,1,2,2,2,2,1,4,4}	176	3340
9800	{1,0,1,2,2,2,2,1,4,4}	176	3340
9900	{1,0,1,2,2,2,2,1,4,4}	176	3340
10000	{1,0,1,2,2,2,2,1,4,4}	176	3340
10100	{1,0,1,2,2,2,2,1,4,4}	176	3340
10200	{1,0,1,2,2,2,2,1,4,4}	176	3340
10300	{1,0,1,2,2,2,2,1,4,4}	176	3340
10400	{1,0,1,2,2,2,2,1,4,4}	176	3340
10500	{1,0,1,2,2,2,2,1,4,4}	176	3340
10600	{1,0,1,2,2,2,2,1,4,4}	176	3340
10700	{1,0,1,2,2,2,2,1,4,4}	176	3340
10800	{1,0,1,2,2,2,2,1,4,4}	176	3340
10900	{1,0,1,2,2,2,2,1,4,4}	176	3340
11000	{1,0,1,2,2,2,2,1,4,4}	176	3340

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
11100	{1,0,1,2,2,2,2,1,4,4}	176	3340
11200	{1,0,1,2,2,2,2,1,4,4}	176	3340
11300	{1,0,1,2,2,2,2,1,4,4}	176	3340
11400	{1,0,1,2,2,2,2,1,4,4}	176	3340
11500	{1,0,1,2,2,2,2,1,4,4}	176	3340
11600	{1,0,1,2,2,2,2,1,4,4}	176	3340
11700	{1,0,1,2,2,2,2,1,4,4}	176	3340
11800	{1,0,1,2,2,2,2,1,4,4}	176	3340
11900	{1,0,1,2,2,2,2,1,4,4}	176	3340
12000	{1,0,1,2,2,2,2,1,4,4}	176	3340
12100	{1,0,1,2,2,2,2,1,4,4}	176	3340
12200	{1,0,1,2,2,2,2,1,4,4}	176	3340
12300	{1,0,1,2,2,2,2,1,4,4}	176	3340
12400	{1,0,1,2,2,2,2,1,4,4}	176	3340
12500	{1,0,1,2,2,2,2,1,4,4}	176	3340
12600	{1,0,1,2,2,2,2,1,4,4}	176	3340
12700	{1,0,1,2,2,2,2,1,4,4}	176	3340
12800	{1,0,1,2,2,2,2,1,4,4}	176	3340
12900	{1,0,1,2,2,2,2,1,4,4}	176	3340
12929	{1,1,1,2,2,2,2,1,0,4}	176	3370

APPENDIX G

Table 4.5: Optimal Solutions of Vw cars for the various iterative stages

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
1	{0,0,0,0,0,0,0,0,0,4}	32	640
200	{0,0,0,0,0,2,1,3,4,4}	98	2165
400	{0,0,0,0,1,1,3,3,4,4}	114	2395
600	{0,0,0,0,2,1,1,3,4,4}	134	2705
800	{0,0,0,0,2,2,3,3,4,4}	150	3055
1000	{0,0,0,0,2,2,3,3,4,4}	150	3055
1200	{0,0,0,0,2,2,3,3,4,4}	150	3055
1400	{0,0,0,1,2,2,2,3,4,4}	154	3085
1600	{0,0,0,1,2,2,2,3,4,4}	154	3085
1800	{0,0,0,1,2,2,2,3,4,4}	154	3085
2000	{0,0,0,2,2,2,3,3,4,4}	162	3205
2200	{0,0,0,2,2,2,3,3,4,4}	162	3205
2400	{0,0,0,2,2,2,3,3,4,4}	162	3205
2600	{0,0,0,2,2,2,3,3,4,4}	162	3205
2800	{0,0,0,2,2,2,3,3,4,4}	162	3205
3000	{0,0,0,2,2,2,3,3,4,4}	162	3205
3200	{0,0,0,2,2,2,3,3,4,4}	162	3205
3400	{0,0,1,1,2,2,3,3,4,4}	172	3240
3600	{0,0,1,1,2,2,3,3,4,4}	172	3240
3800	{0,0,1,1,2,2,3,3,4,4}	172	3240
4000	{0,0,1,1,2,2,3,3,4,4}	172	3240
4200	{0,0,1,1,2,2,3,3,4,4}	172	3240
4400	{0,0,1,1,2,2,3,3,4,4}	172	3240
4600	{0,0,1,1,2,2,3,3,4,4}	172	3240
4800	{0,0,1,1,2,2,3,3,4,4}	172	3240
5000	{0,0,1,1,2,2,3,3,4,4}	172	3240
5200	{0,0,1,1,2,2,3,3,4,4}	172	3240
5400	{0,0,1,1,2,2,3,3,4,4}	172	3240
5600	{0,0,1,1,2,2,3,3,4,4}	172	3240
5800	{0,0,1,1,2,2,3,3,4,4}	172	3240
6000	{0,0,1,1,2,2,3,3,4,4}	172	3240
6200	{0,0,1,1,2,2,3,3,4,4}	172	3240
6400	{0,0,1,1,2,2,3,3,4,4}	172	3240
6600	{0,0,1,1,2,2,3,3,4,4}	172	3240
6800	{0,0,1,1,2,2,3,3,4,4}	172	3240

7000	{0,0,1,1,2,2,3,3,4,4}	172	3240
7200	{0,0,1,1,2,2,3,3,4,4}	172	3240
7400	{0,0,1,1,2,2,3,3,4,4}	172	3240
7600	{0,0,1,1,2,2,3,3,4,4}	172	3240
7800	{0,1,0,1,2,2,3,3,4,4}	162	3310
8000	{0,1,0,1,2,2,3,3,4,4}	162	3310
8200	{0,1,0,1,2,2,3,3,4,4}	162	3310
8400	{0,1,0,2,2,2,3,3,4,4}	168	3385
8600	{0,1,0,2,2,2,3,3,4,4}	168	3385
8800	{0,1,0,2,2,2,3,3,4,4}	168	3385
9000	{0,1,0,2,2,2,3,3,4,4}	168	3385
9200	{0,1,0,2,2,2,3,3,4,4}	168	3385
9400	{0,1,0,2,2,2,3,3,4,4}	168	3385
9600	{0,1,0,2,2,2,3,3,4,4}	168	3385
9800	{0,1,0,2,2,2,3,3,4,4}	168	3385
10000	{0,1,0,2,2,2,3,3,4,4}	168	3385
10200	{0,1,0,2,2,2,3,3,4,4}	168	3385
10400	{0,1,0,2,2,2,3,3,4,4}	168	3385
10600	{0,1,0,2,2,2,3,3,4,4}	168	3385
10800	{0,1,0,2,2,2,3,3,4,4}	168	3385
11000	{0,1,0,2,2,2,3,3,4,4}	168	3385
11200	{0,1,0,2,2,2,3,3,4,4}	168	3385
11400	{0,1,0,2,2,2,3,3,4,4}	168	3385
11600	{0,1,0,2,2,2,3,3,4,4}	168	3385
11800	{0,1,0,2,2,2,3,3,4,4}	168	3385
12000	{0,1,0,2,2,2,3,3,4,4}	168	3385
12200	{0,1,0,2,2,2,3,3,4,4}	168	3385
12400	{0,1,0,2,2,2,3,3,4,4}	168	3385
12600	{0,1,0,2,2,2,3,3,4,4}	168	3385
12800	{0,1,0,2,2,2,3,3,4,4}	168	3385
13000	{0,1,0,2,2,2,3,3,4,4}	168	3385
13200	{0,1,0,2,2,2,3,3,4,4}	168	3385
13400	{0,1,0,2,2,2,3,3,4,4}	168	3385
13600	{0,1,0,2,2,2,3,3,4,4}	168	3385
13800	{0,1,0,2,2,2,3,3,4,4}	168	3385
14000	{0,1,0,2,2,2,3,3,4,4}	168	3385
14200	{0,1,0,2,2,2,3,3,4,4}	168	3385
14400	{0,1,0,2,2,2,3,3,4,4}	168	3385
14600	{0,1,0,2,2,2,3,3,4,4}	168	3385

14800	{0,1,0,2,2,2,3,3,4,4}	168	3385
15000	{0,1,0,2,2,2,3,3,4,4}	168	3385
15200	{0,1,0,2,2,2,3,3,4,4}	168	3385
15400	{0,1,0,2,2,2,3,3,4,4}	168	3385
15600	{0,1,0,2,2,2,3,3,4,4}	168	3385
15800	{0,1,0,2,2,2,3,3,4,4}	168	3385
16000	{0,1,0,2,2,2,3,3,4,4}	168	3385
16200	{0,1,0,2,2,2,3,3,4,4}	168	3385
16400	{0,1,0,2,2,2,3,3,4,4}	168	3385
16600	{0,1,0,2,2,2,3,3,4,4}	168	3385
16800	{0,1,0,2,2,2,3,3,4,4}	168	3385
17000	{0,1,0,2,2,2,3,3,4,4}	168	3385
17200	{0,1,0,2,2,2,3,3,4,4}	168	3385
17400	{0,1,0,2,2,2,3,3,4,4}	168	3385
17600	{0,1,0,2,2,2,3,3,4,4}	168	3385
17800	{0,1,0,2,2,2,3,3,4,4}	168	3385
18000	{0,1,0,2,2,2,3,3,4,4}	168	3385
18200	{0,1,0,2,2,2,3,3,4,4}	168	3385
18400	{0,1,0,2,2,2,3,3,4,4}	168	3385
18600	{0,1,0,2,2,2,3,3,4,4}	168	3385
18800	{0,1,0,2,2,2,3,3,4,4}	168	3385
19000	{0,1,0,2,2,2,3,3,4,4}	168	3385
19200	{0,1,0,2,2,2,3,3,4,4}	168	3385
19400	{0,1,0,2,2,2,3,3,4,4}	168	3385
19600	{0,1,0,2,2,2,3,3,4,4}	168	3385
19800	{0,1,0,2,2,2,3,3,4,4}	168	3385
20000	{0,1,0,2,2,2,3,3,4,4}	168	3385
20200	{0,1,0,2,2,2,3,3,4,4}	168	3385
20400	{0,1,0,2,2,2,3,3,4,4}	168	3385
20600	{1,1,0,1,2,2,3,3,4,4}	167	3430
20800	{1,1,0,1,2,2,3,3,4,4}	167	3430
21000	{1,1,0,1,2,2,3,3,4,4}	167	3430
21200	{1,1,0,2,2,2,2,3,4,4}	171	3460
21400	{1,1,0,2,2,2,3,3,4,4}	173	3505
21600	{1,1,0,2,2,2,3,3,4,4}	173	3505
21800	{1,1,0,2,2,2,3,3,4,4}	173	3505
22000	{1,1,0,2,2,2,3,3,4,4}	173	3505
22200	{1,1,0,2,2,2,3,3,4,4}	173	3505
22400	{1,1,0,2,2,2,3,3,4,4}	173	3505

22600	{1,1,0,2,2,2,3,3,4,4}	173	3505
22800	{1,1,0,2,2,2,3,3,4,4}	173	3505
23000	{1,1,0,2,2,2,3,3,4,4}	173	3505
23200	{1,1,0,2,2,2,3,3,4,4}	173	3505
23400	{1,1,0,2,2,2,3,3,4,4}	173	3505
23600	{1,1,0,2,2,2,3,3,4,4}	173	3505
23800	{1,1,0,2,2,2,3,3,4,4}	173	3505
24000	{1,1,0,2,2,2,3,3,4,4}	173	3505
24200	{1,1,0,2,2,2,3,3,4,4}	173	3505
24400	{1,1,0,2,2,2,3,3,4,4}	173	3505
24600	{1,1,0,2,2,2,3,3,4,4}	173	3505
24800	{1,1,0,2,2,2,3,3,4,4}	173	3505
25000	{1,1,0,2,2,2,3,3,4,4}	173	3505
25200	{1,1,0,2,2,2,3,3,4,4}	173	3505
25400	{1,1,0,2,2,2,3,3,4,4}	173	3505
25445	{1,1,0,2,2,2,3,3,4,4}	173	3505

APPENDIX H

Table 4.6: Optimal Solutions of Skoda cars for the various iterative stages

Iteration	Items selected	Optimal weight (hrs)	Optimal value (GH¢)
1	{0,0,0,0,0,0,0,0,0,6}	48	630
500	{0,0,0,0,0,3,3,3,4,6}	131	2421
1000	{0,0,0,0,2,3,3,3,4,6}	151	2661
1500	{0,0,0,1,0,3,3,3,4,6}	155	2916
2000	{0,0,0,1,2,3,3,3,4,6}	175	3156
2500	{0,0,0,2,0,3,3,3,2,6}	175	3345
3000	{0,0,0,2,0,3,3,3,2,6}	175	3345
3500	{0,0,0,2,0,3,3,3,2,6}	175	3345
4000	{0,0,0,2,0,3,3,3,2,6}	175	3345
4500	{0,0,0,2,0,3,3,3,2,6}	175	3345
5000	{0,0,0,2,0,3,3,3,2,6}	175	3345
5500	{0,0,1,2,0,3,3,3,3,5}	175	3383
6000	{0,0,1,2,0,3,3,3,3,5}	175	3383
6500	{0,0,1,2,0,3,3,3,3,5}	175	3383
7000	{0,0,1,2,0,3,3,3,3,5}	175	3383
7500	{0,0,1,2,0,3,3,3,3,5}	175	3383
8000	{0,0,2,2,0,3,0,3,4,6}	176	3391
8500	{0,0,2,2,0,3,3,3,4,4}	176	3421
9000	{0,0,2,2,0,3,3,3,4,4}	176	3421
9500	{0,0,2,2,0,3,3,3,4,4}	176	3421
10000	{0,0,2,2,0,3,3,3,4,4}	176	3421
10500	{0,0,2,2,0,3,3,3,4,4}	176	3421
11000	{0,0,2,2,0,3,3,3,4,4}	176	3421
11500	{0,0,2,2,0,3,3,3,4,4}	176	3421
12000	{0,0,2,2,0,3,3,3,4,4}	176	3421
12500	{0,0,2,2,0,3,3,3,4,4}	176	3421
13000	{0,0,2,2,0,3,3,3,4,4}	176	3421
13500	{0,0,2,2,0,3,3,3,4,4}	176	3421
14000	{0,1,1,2,0,3,3,3,2,3}	175	3440
14500	{0,1,1,2,1,3,2,3,4,2}	176	3441
15000	{0,1,1,2,1,3,2,3,4,2}	176	3441
15500	{0,1,1,2,1,3,2,3,4,2}	176	3441
16000	{0,1,1,2,1,3,2,3,4,2}	176	3441
16500	{0,1,1,2,1,3,2,3,4,2}	176	3441
17000	{0,1,2,2,0,3,3,3,3,2}	175	3478
17500	{0,1,2,2,0,3,3,3,3,2}	175	3478

18000	{0,1,2,2,0,3,3,3,3,2}	175	3478
18500	{0,1,2,2,0,3,3,3,3,2}	175	3478
19000	{0,1,2,2,0,3,3,3,3,2}	175	3478
19500	{0,1,2,2,0,3,3,3,3,2}	175	3478
20000	{1,0,0,2,0,3,3,3,4,5}	175	3486
20500	{1,0,0,2,0,3,3,3,4,5}	175	3486
21000	{1,0,0,2,0,3,3,3,4,5}	175	3486
21500	{1,0,0,2,0,3,3,3,4,5}	175	3486
22000	{1,0,0,2,0,3,3,3,4,5}	175	3486
22500	{1,0,1,2,0,3,2,3,4,5}	176	3516
23000	{1,0,1,2,0,3,2,3,4,5}	176	3516
23500	{1,0,1,2,0,3,2,3,4,5}	176	3516
24000	{1,0,1,2,0,3,2,3,4,5}	176	3516
24500	{1,0,1,2,0,3,2,3,4,5}	176	3516
25000	{1,0,1,2,0,3,2,3,4,5}	176	3516
25500	{1,0,2,2,0,3,3,3,2,4}	176	3527
26000	{1,0,2,2,1,3,2,3,4,3}	176	3536
26500	{1,0,2,2,1,3,2,3,4,3}	176	3536
27000	{1,0,2,2,1,3,2,3,4,3}	176	3536
27500	{1,0,2,2,1,3,2,3,4,3}	176	3536
28000	{1,0,2,2,1,3,2,3,4,3}	176	3536
28500	{1,1,0,2,0,3,3,3,3,3}	175	3543
29000	{1,1,0,2,0,3,3,3,3,3}	175	3543
29500	{1,1,0,2,0,3,3,3,3,3}	175	3543
30000	{1,1,0,2,0,3,3,3,3,3}	175	3543
30500	{1,1,0,2,0,3,3,3,3,3}	175	3543
31000	{1,1,0,2,0,3,3,3,3,3}	175	3543
31500	{1,1,1,2,0,3,3,3,4,2}	175	3581
32000	{1,1,1,2,0,3,3,3,4,2}	175	3581
32500	{1,1,1,2,0,3,3,3,4,2}	175	3581
33000	{1,1,1,2,0,3,3,3,4,2}	175	3581
33500	{1,1,1,2,0,3,3,3,4,2}	175	3581
34000	{1,1,2,2,0,3,2,3,4,2}	176	3611
34500	{1,1,2,2,0,3,2,3,4,2}	176	3611
34654	{1,1,2,2,0,3,2,3,4,2}	176	3611