Journal of Heuristics, 4: 63–86 (1998)

A Genetic Algorithm for the Multidimensional Knapsack Problem

P.C. CHU AND J.E. BEASLEY
The Management School, Imperial College, London SW7 2AZ, England
email: p.chu@ic.ac.uk; j.beasley@ic.ac.uk

Abstract

In this paper we present a heuristic based upon genetic algorithms for the multidimensional knapsack problem. A heuristic operator which utilises problem-specific knowledge is incorporated into the standard genetic algorithm approach. Computational results show that the genetic algorithm heuristic is capable of obtaining high-quality solutions for problems of various characteristics, whilst requiring only a modest amount of computational effort. Computational results also show that the genetic algorithm heuristic gives superior quality solutions to a number of other heuristics.

Key Words: genetic algorithms, multidimensional knapsack, multiconstraint knapsack, combinatorial optimisation

1. Introduction

In this paper, we present a genetic algorithm (GA) based heuristic for a well-known NP-hard problem, the *multidimensional knapsack problem* (MKP), which can be formulated as:

$$\text{maximise} \quad \sum_{j=1}^{n} p_j x_j, \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3}$$

Each of the m constraints described in Eq. (2) is called a *knapsack* constraint, so the MKP is also called the *m-dimensional knapsack problem*. Other names given to this problem in the literature are the *multiconstraint knapsack problem*, the *multi-knapsack problem* and the *multiple knapsack problem*. Some authors also include the term *zero-one* in their name for the problem, e.g., the *multidimensional zero-one knapsack problem*. The special case corresponding to $m = 2$ is known as the *bidimensional knapsack problem* or the *bi-knapsack problem*.

We would comment here that this use of alternative names for the same problem is obviously unnecessary and potentially confusing. Historically the majority of authors appear to have used the name *multidimensional knapsack problem* and so we have also used this name in our paper.

Let $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$, with $b_i \geq 0$ for all $i \in I$ and $r_{ij} \geq 0$ for all $i \in I$, $j \in J$. A *well-stated* MKP assumes that $p_j > 0$ and $r_{ij} \leq b_i < \sum_{j=1}^{n} r_{ij}$ for all $i \in I$, $j \in J$, since any violation of these conditions will result in some x_j's being fixed to zero and/or some constraints being eliminated.

Many practical problems can be formulated as a MKP, for example, the capital budgeting problem where project j has profit p_j and consumes r_{ij} units of resource i. The goal is to find a subset of the n projects such that the total profit is maximised and all resource constraints are satisfied. Other applications of the problem include allocating processors and databases in a distributed computer system (Gavish and Pirkul (1982)), project selection and cargo loading (Shih (1979)), and cutting stock problems (Gilmore and Gomory (1966)).

We would note here that the MKP can be regarded as a *general statement* of any zero-one integer programming problem with non-negative coefficients. Indeed much of the early work on the problem (e.g., Hillier (1969), Kochenberger, McCarl, and Wymann (1974), Senju and Toyoda (1968), Zanakis (1977)) viewed the problem in this way.

2. Literature survey

Most of the research on knapsack problems deals with the much simpler single constraint version ($m = 1$). For the single constraint case the problem is not strongly NP-hard and effective approximation algorithms have been developed for obtaining near-optimal solutions. A good review of the single knapsack problem and its associated exact and heuristic algorithms is given by Martello and Toth (1990).

Below we review the literature for the MKP structuring our review by first considering exact algorithms and then heuristic algorithms.

2.1. Exact algorithms

Shih (1979) presented a branch and bound algorithm for the MKP. In his method, an upper bound was obtained by computing the objective function value associated with the optimal fractional solution algorithm (Dantzig (1957)) for each of the m single constraint knapsack problems separately and selecting the minimum objective function value among those as the upper bound. His algorithm was tested on a set of randomly generated test problems with sizes up to $m = 5$ and $n = 90$, and with various constraint tightness. Computational results showed that his algorithm performed better than the general zero-one additive algorithm of Balas (1965).

Another branch and bound algorithm was developed by Gavish and Pirkul (1985). Various relaxations of the problem, including lagrangean, surrogate and composite relaxations were used. Experiments were carried out to evaluate the quality of the bounds generated by these different relaxations. Their results showed that the composite relaxation (which used a subgradient optimisation procedure to determine the multipliers) yielded the best bounds overall, albeit at extra computational effort. New algorithms for obtaining surrogate bounds were also developed and tested. Rules for reducing the problem size were suggested and shown to be effective through computational tests. Their final algorithm was tested on a set of randomly generated problems with sizes up to $m = 5$ and $n = 200$. Their algorithm was

compared with the exact method of Shih (1979) and was found to be faster by at least one order of magnitude. It was also compared to, and found to be faster than, the commercial mixed-integer programming solver Sciconic/VM. Their algorithm can also be used as a heuristic by terminating it before the tree search is completed. They found that such a scheme is superior to a heuristic developed by Loulou and Michaelides (1979).

Other previous exact algorithms, with only limited success being reported, include dynamic programming based methods (Gilmore and Gomory (1966), Nemhauser and Ullmann (1969), Weingartner (1967), Weingartner and Ness (1967)), an enumeration algorithm based on Fourier-Motzkin elimination (Cabot (1970)) and an iterative scheme in which linear programs were solved to generate subproblems that were then solved through implicit enumeration (Soyster, Lev, and Slivka (1978)).

Crama and Mazzola (1994) showed that although the bounds derived from the well-known relaxations, such as lagrangean, surrogate, or composite relaxations, are stronger than the bounds obtained from the linear programming (LP) relaxation, the improvement in the bound that can be realised using these relaxations is limited. In particular, they showed that the improvement in the quality of the bounds using any of these relaxations cannot exceed the magnitude of the largest coefficient in the objective function.

There have been a limited number of papers considering a statistical/asymptotic analysis of the MKP. An asymptotic analysis was presented by Schilling (1990) who computed the asymptotic ($n \to \infty$ with m fixed) objective function value for the MKP where the r_{ij}'s and p_j's were uniformly (and independently) distributed over the unit interval and where $b_i = 1$. Szkatula (1994) generalised that analysis to the case where the b_i were not restricted to be one (see also Szkatula (1997)). A statistical analysis of the MKP was conducted by Fontanari (1995), who investigated the dependence of the objective function on the knapsack capacities and on the number of capacity constraints, in the case when all n objects were assigned the same profit value and the r_{ij}'s were uniformly distributed over the unit interval. A rigorous upper bound on the optimal profit was obtained by employing the annealed approximation and then compared with the exact value obtained through a lagrangean relaxation method.

2.2. Heuristic algorithms

Below we review heuristic algorithms for the MKP, structuring our review by clustering related papers together. The clusters we consider are:

- early heuristics, early heuristic approaches
- bound based heuristics, which make use of an upper bound on the optimal solution to the MKP
- tabu search heuristics, based on tabu search concepts
- genetic algorithm heuristics, based on genetic algorithm concepts
- analysed heuristics, with some theoretical underlying analysis relating to their worst-case or probabilistic performance
- other heuristics.

2.2.1. Early heuristics. Zanakis (1977) gave detailed results comparing three algorithms from Hillier (1969), Kochenberger, McCarl, and Wymann (1974), and Senju and Toyoda (1968). No one heuristic was found to dominate the others computationally.

Loulou and Michaelides (1979) presented a greedy-like method based on Toyoda's primal heuristic (1975). Primal heuristics start with a zero solution, after which a succession of variables are assigned the value one, according to a given rule, as long as the solution remain feasible. Their algorithm selects a variable having the maximum "pseudo-utility" among all candidates in each step. The pseudo-utility is defined as $u_j = p_j/v_j$, where v_j is the *penalty factor* of variable j, which depends on the resource coefficients r_{ij} and can be defined in several ways. Their heuristic was tested on randomly generated problems of relatively small sizes as well as some larger real-world problems. Their results showed that the average deviation from optimum ranged from 0.26% to 1.08% for smaller problems and up to 14% for larger problems.

2.2.2. Bound based heuristics. Balas and Martin (1980) used linear programming by relaxing the integrality constraints and heuristically set the fractional solution integer whilst maintaining feasibility.

Magazine and Oguz (1984) presented a heuristic algorithm that combines the ideas of Senju and Toyoda's dual heuristic (1968) with Everett's generalised lagrange multiplier approach (1963). Dual heuristics start with the all-ones solution, variables are then successively set to zero according to heuristic rules until a feasible solution is obtained. Their algorithm computes the approximate solution and uses the multipliers generated to obtain an upper bound. This upper bound is generally loose compared to the bound obtained by LP relaxation. Their algorithm (MKNAP) was tested on a large set of randomly generated problems varying in size from $m = 20$ to 1,000 and $n = 20$ to 1,000, and was compared with two other heuristic algorithms: the primal heuristics of Kochenberger, McCarl, and Wymann (1974) (KOCH) and the dual approach of Senju and Toyoda (1968) (SEN). They noted that, in terms of solution quality, KOCH performed slightly better than MKNAP overall, followed by SEN. MKNAP and SEN performed markedly better than KOCH in terms of computation time. The time complexity of KOCH, SEN and MKNAP were shown to be $O(mn^2)$, $O(mn^2)$ and $O(mn^3)$, respectively.

Pirkul (1987) presented a heuristic algorithm which makes use of surrogate duality. The m knapsack constraints were transformed into a single knapsack constraint using surrogate multipliers. A feasible solution was obtained by packing this single knapsack in the decreasing order of profit/weight ratios. This ratio was defined as $p_j / \sum_{i=1}^{m} \omega_i r_{ij}$ where ω_i is the surrogate multiplier for constraint i. Surrogate multipliers were determined using a descent procedure. New feasible solutions were obtained by fixing variables to zero and the best feasible solution chosen. Extensive computational testing indicated that the algorithm generated good feasible solutions. The performance of his heuristic is considerably better than the heuristic of Loulou and Michaelides (1979) and similar to the pivot and complement heuristic of Balas and Martin (1980) in terms of solution quality. We implemented this heuristic (see Section 5.4.1), but using an alternative method for finding surrogate multipliers, which involves solving the LP relaxation of the original problem and using the dual variables (the shadow prices) from that solution as surrogate multipliers.

Lee and Guignard (1988) presented an heuristic that combined Toyoda's primal heuristic (1975) with variable fixing, linear programming and a complementing procedure from Balas and Martin (1980). Computational results were presented with some standard test problems and some randomly generated problems with sizes up to $m = 20$ and $n = 200$ which indicated that their heuristic produces better quality results than Toyoda (1975) and Magazine and Oguz (1984), but is out-performed by Balas and Martin (1980).

Volgenant and Zoon (1990) extended Magazine and Oguz's heuristic in two ways: (1) in each step, not one, but more, multiplier values are computed simultaneously, and (2) at the end of the procedure the upper bound is sharpened by changing some multiplier values. From a comparison using a set of test problems, these extensions appeared to yield an improvement, on average, at the cost of only a modest amount of extra computing time. The time complexity of their algorithm is $O(n(n+m))$.

Freville and Plateau (1994) presented an efficient preprocessing algorithm for the MKP. Their algorithm (AGNES), based on their previous work (Freville and Plateau (1986)), provided sharp lower and upper bounds on the optimal value, and also a tighter equivalent representation by reducing the continuous feasible set and by eliminating constraints and variables. Their scheme was shown to be very effective through experiments with standard test problems and large-scale randomly generated problems with sizes up to $m = 30$ and $n = 500$. Their computational results indicated the superiority of their algorithm over the surrogate heuristic of Pirkul (1987) (denoted by MKHEUR), although our implementation of MKHEUR (see Section 5.4.1) showed significantly better results than the ones reported by them for similar problems. Their algorithm also compared favourably with the heuristic and early termination algorithm of Gavish and Pirkul (1985) (denoted by G&P). However, we note here that the large computation times for the G&P algorithm reported in their study (Freville and Plateau (1994)) are somewhat inconsistent with the small computation times reported in (Gavish and Pirkul (1985)).

Freville and Plateau (1997) presented a heuristic for the special case corresponding to $m = 2$, the bidimensional knapsack problem. Their heuristic incorporates a number of components including problem reduction, a bound based upon surrogate relaxation and partial enumeration. Computational results were presented for randomly generated problems of sizes up to $n = 750$ comparing their heuristic with the exact algorithm of Gavish and Pirkul (1985).

2.2.3. Tabu search heuristics. A number of papers involving the use of tabu search to solve the MKP have appeared in recent years.

Dammeyer and Voss (1993) presented a tabu search heuristic based on reverse elimination. Computational results were presented for 57 standard test problems from the literature indicating that they found the optimal solution for 41 of these problems.

Aboudi and Jörnsten (1994) combined tabu search with the pivot and complement heuristic of Balas and Martin (1980) in a heuristic for general zero-one integer programming. Computational results were presented for 57 standard MKP test problems from the literature indicating that they found the optimal solution for 49 of these problems. A similar approach is presented in Løkketangen, Jörnsten, and Storøy (1994) with computational results for the same set of test problems indicating that they found the optimal solution for 39 of these problems.

Battiti and Tecchiolli (1995) presented a heuristic based on reactive tabu search (essentially tabu search but with the length of the tabu list varied over the course of the algorithm). Computational results were presented for their heuristic for problems of sizes up to $m = n = 500$. They also presented results for a number of other heuristics based upon repeated local search, simulated annealing, genetic algorithms and neural networks.

Glover and Kochenberger (1996) presented a heuristic based on tabu search. Their approach employed a flexible memory structure that integrates recency and frequency information keyed to "critical events" in the search process. Their method was enhanced by a strategic oscillation scheme that alternates between constructive (current solution feasible) and destructive (current solution infeasible) phases. They define a "critical event" as the last feasible solution found after a transition between phases. Such solutions were subjected to a simple local optimisation procedure in an attempt to improve them. Their approach successfully obtained optimal solutions for each of 57 standard test problems from the literature. They also report that for 24 randomly generated problems of sizes up to $m = 25$ and $n = 500$ their heuristic outperformed an incomplete branch and bound algorithm.

Løkketangen and Glover (1996) presented a heuristic based on probabilistic tabu search (essentially tabu search but with the acceptance/rejection of a potential move controlled by a probabilistic process). Computational results were presented for 18 standard problems taken from the literature of sizes up to $m = 30$ and $n = 90$. Optimal solutions were obtained for 13 of these problems.

Hanafi and Freville (1997) presented a heuristic strongly related to the work of Glover and Kochenberger (1996). They solve a subset of the same test problems and report better quality results for this subset than Glover and Kochenberger.

Løkketangen and Glover (1997) presented a tabu search heuristic designed to solve general zero-one mixed-integer programming problems. Applying their approach to 57 standard MKP test problems from the literature they obtained (using one or other variant of their heuristic) optimal solutions for all but 3 of these test problems.

2.2.4. Genetic algorithm heuristics. A number of papers involving the use of genetic algorithms to solve the MKP have appeared in recent years.

In the GA of Khuri, Bäck, and Heitkötter (1994) infeasible solutions were allowed to participate in the search and a simple fitness function which uses a graded penalty term was used. Their heuristic was tested on a small number of standard test problems; only moderate results were reported.

In Thiel and Voss (1994) simple heuristic operators based on local search algorithms were used, and a hybrid algorithm based on combining a GA with a tabu search heuristic was suggested. Their heuristic was tested on a set of standard test problems, but the results were not computationally competitive with those obtained using other heuristic methods.

In Rudolph and Sprave (1995, 1996) a GA was presented where parent selection is not unrestricted (as in a standard GA) but is restricted to be between "neighbouring" solutions. Infeasible solutions were penalised as in Khuri, Bäck, and Heitkötter (1994). An adaptive threshold acceptance schedule (motivated by Dueck and Scheuer (Dueck (1993), Dueck and Scheuer (1990)) for child acceptance was used. Computational results were presented for one problem of size $m = 5$ and $n = 50$.

In the GA of Hoff, Løkketangen, and Mittet (1996) only feasible solutions were allowed. Their GA successfully obtained optimal solutions for 56 out of 57 standard test problems taken from the literature (when replicated ten times for each problem).

2.2.5. Analysed heuristics. Frieze and Clarke (1984) described a polynomial approximation scheme based on the use of the dual simplex algorithm for linear programming, and analysed the asymptotic properties of a particular random model (see also (Schilling (1990), Szkatula (1994, 1997)) for other asymptotic analyses). No computational results were given.

Rinnooy Kan, Stougie, and Vercellis (1993) proposed a class of generalised greedy algorithms in which items are selected according to decreasing ratios of their profit and a weighted sum of their resource coefficients. They investigated the complexity of computing a set of weights that gives the maximum greedy solution value. Their heuristics were subjected to both a worst-case, and a probabilistic, performance analysis. No computational results were given.

Averbakh (1994) investigated the properties of several dual characteristics of the MKP for different probabilistic models. He also presented a fast statistically efficient approximate algorithm with linear running time complexity for problems with random coefficients.

2.2.6. Other heuristics. Fox and Scudder (1985) presented a heuristic based on starting from setting all variables to zero(one) and successively choosing variables to set to one(zero). Computational results were presented for randomly generated test problems with sizes up to $m = 100$ and $n = 100$ but with $p_j = 1$ and $r_{ij} = 0$ or 1.

Drexel (1988) presented a heuristic based upon simulated annealing. Computational results were presented for 57 test problems indicating that the optimal solution was found for 25 of these problems.

Glover (1994) presented a heuristic based on ghost image processes and reported that, compared to Senju and Toyoda's dual heuristic (1968), his heuristic gave superior quality results on randomly generated problems with sizes up to $m = 20$ and $n = 100$.

Hanafi, Freville, and Abedellaoui (1996) presented a simple multistage algorithm (SMA) within which a number of different local search procedures (such as greedy, simulated annealing, threshold accepting (Dueck (1993), Dueck and Scheuer (1990)) and noising (Charon and Hudry (1993)) can be used. A computational comparison of SMA with AGNES (Freville and Plateau (1994)) was made using 54 test problems taken from (Freville and Plateau (1990)). They also presented a comparison between two tabu search heuristics of their own devising and a number of other tabu search heuristics (Aboudi and Jörnsten (1994), Glover and Kochenberger (1996), Løkketangen and Glover (1996, 1997)). Overall one of their tabu search heuristics produced results equal in quality to the tabu search heuristic of Glover and Kochenberger (1996).

3. Genetic algorithms

A genetic algorithm can be understood as an "intelligent" probabilistic search algorithm which can be applied to a variety of combinatorial optimisation problems (Reeves (1993)). The theoretical foundations of GAs were originally developed by Holland (1975). GAs

are based on the evolutionary process of biological organisms in nature. During the course of evolution, natural populations evolve according to the principles of natural selection and "survival of the fittest". Individuals which are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individuals which are less fit will be eliminated. This means that the *genes* from the highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted ancestors may produce even more fit offspring. In this way, species evolve to become more and more well adapted to their environment.

A GA simulates these processes by taking an initial population of individuals and applying genetic operators in each reproduction. In optimisation terms, each individual in the population is encoded into a string or *chromosome* which represents a possible *solution* to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or *solutions* are given opportunities to reproduce by exchanging pieces of their genetic information, in a *crossover* procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e., *children*), which share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (*generational* approach) or replace less fit individuals (*steady-state* approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found. The basic steps of a simple GA are shown below.

```
Generate an initial population;
Evaluate fitness of individuals in the population;
repeat:
   Select parents from the population;
   Recombine (mate) parents to produce children;
   Evaluate fitness of the children;
   Replace some or all of the population by the children;
until a satisfactory solution has been found.
```

A more comprehensive overview of GAs can be found in (Bäck, Fogel, and Michalewicz (1997), Beasley, Bull, and Martin (1993a, 1993b), Goldberg (1989), Mitchell (1996), Reeves (1993)).

4. A GA for the MKP

We modified the basic GA described in the previous section in such a way that problem-specific knowledge is considered. Our modified GA for the MKP is as follows.

4.1. Representation and fitness function

The first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme, i.e., a way to represent individuals in the GA population. The

j	1	2	3	4	5	$\cdots$	$n-1$	n
$S[j]$	0	1	0	0	1	$\cdots$	0	1

Figure 1. Binary representation of a MKP solution.

standard GA 0-1 binary representation is an obvious choice for the MKP since it represents the underlying 0-1 integer variables.

Hence, in our representation, we used a n-bit binary string, where n is the number of variables in the MKP. In this representation a value of 0 or 1 at the jth bit implies that $x_j = 0$ or 1 in the solution, respectively. This binary representation of an individual's chromosome (solution) for the MKP is illustrated in Figure 1.

We note that a bit string $S \in \{0, 1\}^n$ might represent an infeasible solution. An infeasible solution is one for which at least one of the knapsack constraints is violated, i.e., $\sum_{j=1}^{n} r_{ij} S[j] > b_i$ for some $i \in I$.

There are a number of standard ways (Chu and Beasley (1995), Davis and Steenstrup (1987), Michalewicz (1995)) of dealing with constraints and infeasible solutions in GAs:

- to use a representation that automatically ensures that all solutions are feasible,
- to separate the evaluation of fitness and infeasibility (Chu and Beasley (1995)),
- to design a heuristic operator (often called a *repair operator*) which guarantees to transform any infeasible solution into a feasible solution,
- to apply a penalty function (Goldberg (1989), Powell and Skolnick (1993), Richardson et al. (1989), Smith and Tate (1993)) to penalise the fitness of any infeasible solution without distorting the fitness landscape.

Based upon our previous experience with GAs (Beasley and Chu (1996), Chu (1997), Chu and Beasley (1995, 1997)) we adopted the approach of using a heuristic operator to convert an infeasible solution into a feasible one, since a simple greedy heuristic (see Section 4.4) exists that is guaranteed to construct a feasible MKP solution. By restricting the GA to search only the feasible region of the solution space, we have the following single fitness function based entirely on the objective function to be maximised:

$$f(S) = \sum_{j=1}^{n} p_j S[j]. \tag{4}$$

Note here that we are trying to *maximise* fitness—in other words, the higher the fitness the better a MKP solution is.

4.2. Parent selection

Parent selection is the task of assigning reproductive opportunities to each individual in the population. Typically in a GA we need to generate two parents who will have (one or more) children.

The tournament selection method works by forming two pools of individuals, each consisting of T individuals drawn from the population randomly. Two individuals with the best fitness, each taken from one of the two tournament pools, are chosen to be parents. Using a larger value for T has the effect of increasing selection pressure on the more fit individuals.

We adopted the standard *binary* tournament selection method (i.e., $T = 2$) as the method for parent selection because it can be implemented very efficiently.

4.3. Crossover and mutation

The binary, problem-independent, representation we have adopted for the MKP allows a wide range of the standard GA crossover and mutation operators to be adopted. Based on our previous work, (Beasley and Chu (1996), Chu (1997), Chu and Beasley (1995, 1997)), indicating that the overall performance of GAs for combinatorial optimisation problems is often relatively insensitive to the particular choice of crossover operator, as well as some limited computational experience in the context of the MKP to re-confirm this observation, we arbitrarily adopted the uniform crossover operator as the default crossover operator.

In uniform crossover two parents have a single child. Each bit in the child solution is created by copying the corresponding bit from one or the other parent, chosen according to a binary random number generator [0, 1]. If the random number is a 0, the bit is copied from the first parent, if it is a 1, the bit is copied from the second parent.

Once a child solution has been generated through crossover, a mutation procedure is performed that mutates some randomly selected bits in the child solution, i.e., causes these chosen bits to change from 0 to 1 or vice versa. The rate of mutation is generally set to be a small value (in the order of 1 or 2 bits per string).

We would comment here that one useful approach in GA work is to first make simple standard choices for crossover and mutation operators and only re-examine these choices if computational results are disappointing. As will become apparent below the computational results for the GA with these simple choices were not disappointing.

4.4. Repair operator

Clearly, the child solution generated by the crossover and mutation procedures may not be feasible because the knapsack constraints may not all be satisfied. In order to guarantee feasibility, a heuristic operator based on a simple greedy algorithm was applied.

Note here that the standard GA term for such an operator is to call it a *repair operator*. We believe that this is really an inappropriate term because the term *repair* implies fixing something which once worked and now is broken. In fact the GA representation never worked in the first place, i.e., it never had the property that it guaranteed feasibility for the child. However, because the term repair operator is widely used in GAs, we will also use it in this paper.

The general technique used to design greedy-like heuristics for the MKP follows the notion of the *pseudo-utility* ratios (for solving the single constraint problem) which are defined as the ratios of the objective function coefficients (p_j's) to the coefficients of the single knapsack constraint (r_j's). The greater the ratio, the higher the chance that the corresponding

variable will be equal to one in the solution. However, when more than one constraint is present, it is not quite clear how this approach can be generalised. Several ways to calculate pseudo-utility ratios for the MKP, with various degrees of complexity, have been proposed (e.g., see (Loulou and Michaelides (1979), Pirkul (1987), Toyoda (1975)). Here we adopted the *surrogate duality* approach of Pirkul (1987) in determining the pseudo-utility ratios as it is conceptually simple and computationally straightforward. The general idea behind this method is described very briefly as follows.

The surrogate relaxation problem of the MKP (denoted by SR-MKP) can be defined as:

$$\text{maximise} \quad \sum_{j=1}^{n} p_j x_j, \tag{5}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \left(\sum_{i=1}^{m} \omega_i r_{ij} \right) x_j \leq \sum_{i=1}^{m} \omega_i b_i, \tag{6}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n, \tag{7}$$

where $\omega = \{\omega_1, \ldots, \omega_m\}$ is a set of surrogate multipliers (or *weights*) of some positive real numbers. The inequality, Eq. (6), is called the surrogate constraint. Surrogate constraints were first introduced by Glover (1965, 1977) to provide choice rule evaluations and bounds for integer programming problems in (Glover (1965)) and to transform infeasible solutions into feasible solutions in the context of an evolutionary procedure in (Glover (1977)).

SR-MKP is often solved to obtain an upper bound on the original MKP (see Gavish and Pirkul (1985)). The best possible bound using this relaxation method can be obtained by finding an optimal set of weights that minimises the solution value of SR-MKP. It has been shown that if the optimal weights are known, then the bound generated by this relaxation is better than, or equal to, bounds generated by both lagrangean relaxation and LP relaxation. In practice, however, deriving optimal weights for SR-MKP is a difficult task (Gavish and Pirkul (1985)).

Since SR-MKP is essentially a single constraint knapsack problem, the pseudo-utility ratio for each variable, based on the surrogate constraint coefficient, is simply $u_j = p_j / \sum_{i=1}^{m} \omega_i r_{ij}$. Clearly, the effectiveness of the greedy heuristic based on the surrogate problem (in the form of a single constraint knapsack problem) strongly depends on the ability of the surrogate constraint to capture the aggregate weighted consumption level of resources for each variable, and this in turn relies on the determination of a good set of weights.

Pirkul (1987) suggested several methods to derive these weights. One of the simplest methods to obtain reasonably good weights is to solve the LP relaxation of the original MKP (using a linear programming solver like CPLEX (1995)) and to use the values of the *dual variables* as the weights. In other words, ω_i is set equal to the *shadow price* of the ith constraint in the LP relaxation of the MKP.

We would comment here that although Gavish and Pirkul (1985) found that solving the LP relaxation of the MKP was time-consuming (using the code (Land and Powell (1973)) available to them at that time) it is clear that linear programming solution technology has advanced considerably since then. Certainly our computational experience has been (see Section 5.3) that a modern code like CPLEX has absolutely no difficulty in solving the LP relaxation of the MKP.

Having obtained the ω_i's and subsequently computed the u_j's ($u_j = p_j / \sum_{i=1}^{m} \omega_i r_{ij}$), a repair operator can then be designed which considers the inclusion and exclusion of each variable in the child solution based on u_j. Our repair operator consists of two phases. The first phase (called DROP) examines each variable in *increasing* order of u_j and changes the variable from one to zero if feasibility is violated. The second phase (called ADD) reverses the process by examining each variable in *decreasing* order of u_j and changes the variable from zero to one as long as feasibility is not violated.

The aim of the DROP phase is to obtain a feasible solution from an infeasible solution, whilst the ADD phase seeks to improve the fitness of a feasible solution. We remark that in order to achieve an efficient implementation of the repair operator, a preprocessing routine is applied to each problem that sorts and renumbers variables according to the *decreasing* order of their u_j's. The pseudo-code for the repair operator (after this preprocessing has been carried out) is given in Algorithm 1.

Algorithm 1. Repair operator for the MKP
Let: R_i = the accumulated resources of constraint i in S.
1: initialise $R_i = \sum_{j=1}^{n} r_{ij} S[j], \forall i \in I$;
2: **for** $j = n$ to 1 **do** /* DROP phase */
3: **if** ($S[j] = 1$) and ($R_i > b_i$, for any $i \in I$) **then**
4: set $S[j] \leftarrow 0$;
5: set $R_i \leftarrow R_i - r_{ij}, \forall i \in I$;
6: **end if**
7: **end for**
8: **for** $j = 1$ to n **do** /* ADD phase */
9: **if** ($S[j] = 0$) and ($R_i + r_{ij} \leq b_i, \forall i \in I$) **then**
10: set $S[j] \leftarrow 1$;
11: set $R_i \leftarrow R_i + r_{ij}, \forall i \in I$.
12: **end if**
13: **end for**

Algorithm 1 is "greedy" in the sense that during the DROP phase (steps 2–7), variables, with the lowest u_j being considered first, are successively removed from the solution until a feasible solution is achieved. This is followed by the ADD phase (steps 8–13), which successively adds variables, with the highest u_j being considered first, back into the solution until a *complete* solution has been found, i.e., the remaining free resources are not sufficient to add another variable to the solution. Algorithm 1 is *guaranteed* to always produce a feasible solution for the MKP, irrespective of the initial child solution.

Note here that elements of Algorithm 1 have appeared in many papers down the years. For example phases equivalent (or very similar) to DROP and ADD above have appeared before, e.g.,

- for the DROP phase see (Fox and Scudder (1985), Magazine and Oguz (1984), Senju and Toyoda (1968)),
- for the ADD phase see (Fox and Scudder (1985), Kochenberger, McCarl, and Wymann (1974), Loulou and Michaelides (1979), Pirkul (1987), Toyoda (1975)),

- for both phases see (Freville and Plateau (1994, 1997), Glover and Kochenberger (1996), Hanafi and Freville (1997), Hanafi, Freville, and Abedellaoui (1996), Theil and Voss (1994)).

One difference however between our algorithm and this work is the use of linear programming dual variables as multipliers in the pseudo-utility ratio ($u_j = p_j / \sum_{i=1}^{m} \omega_i r_{ij}$) to set the order in which variables are considered. The work mentioned above typically uses multipliers in the pseudo-utility ratio based directly on the original problem coefficients or calculated by a heuristic procedure for solving the surrogate relaxation SR-MKP (Eqs. (5)–(7)).

4.5. Initial population

In order to achieve sufficient diversification, the initial population, with the size being fixed at $N = 100$, was randomly generated. Each of the initial *feasible* solutions was constructed by a primitive primal heuristic that repeatedly randomly selects a variable and sets it to one if the solution is feasible. The heuristic terminates when the selected variable cannot be added to the solution (see Algorithm 2).

Algorithm 2. Initialise $\mathcal{P}(0)$ for the MKP
for $k = 1$ to N **do**
  set $S_k[j] \leftarrow 0, \forall j \in J$;
  set $T \leftarrow J$; /* T is a dummy set */
  randomly select a $j \in T$ and set $T \leftarrow T - j$;
  while $R_i + r_{ij} \leq b_i, \forall i \in I$ **do**
    set $S_k[j] \leftarrow 1$;
    set $R_i \leftarrow R_i + r_{ij}, \forall i \in I$;
    randomly select a $j \in T$ and set $T \leftarrow T - j$.
  end while
end for

4.6. Algorithmic outline

The outline of the GA heuristic which we have developed for the MKP is shown in Algorithm 3. The default settings for our GA are:

- the binary tournament selection method,
- the uniform crossover operator,
- a mutation rate equal to 2 bits per child string,
- to discard any duplicate children (i.e., discard any child which is the same as a member of the population),
- the steady-state replacement method based on eliminating the individual with the lowest fitness value.

Since the repair operator is the most computationally expensive procedure in the algorithm, we will restrict the complexity analysis of Algorithm 3 *per iteration* (per child generated) to the repair operator.

Recall here that we remarked previously in Section 4.4 that in order to achieve an efficient implementation of the repair operator a preprocessing routine is applied to each problem to sort variables. This is only done once however, and so does not affect the per iteration time complexity analysis given below.

Algorithm 3. A GA for the MKP

```
 1: set t := 0;
 2: initialise P(t) := {S_1, ..., S_N}, S_i ∈ {0, 1}^n;
 3: evaluate P(t) : {f(S_1), ..., f(S_N)};
 4: find S* ∈ P(t) s.t. f(S*) ≥ f(S), ∀S ∈ P(t);
 5: while t < t_max do
 6:    select {P_1, P_2} := Φ(P(t)); /* Φ = binary tournament selection */
 7:    crossover C := Ω_c(P_1, P_2); /* Ω_c = uniform crossover operator */
 8:    mutate C ← Ω_m(C); /* Ω_m = mutation operator */
 9:    make C feasible, C ← Ω_r(C); /* Ω_r = repair operator */
10:    if C ≡ any S ∈ P(t) then /* C is a duplicate of a member of the population */
11:       discard C and go to 6;
12:    end if
13:    evaluate f(C);
14:    find S' ∈ P(t) s.t. f(S') ≤ f(S), ∀S ∈ P(t)and replace S' ← C;
       /* steady-state replacement */
15:    if f(C) > f(S*) then
16:       S* ← C;
17:    end if /* update best solution S* found */
18:    t ← t + 1;
19: end while
20: return S*, f(S*).
```

It can be easily seen that in Algorithm 1, the ADD and DROP phases require at most $O(mn)$ operations each, although this worst-case scenario is unlikely to occur since the two if statements (steps 3 and 9 in Algorithm 1) will not always be true for every j examined. Nevertheless, we conclude that the time complexity of the repair operator, as well as the complexity of the GA per iteration, is approximately $O(mn)$. Since this time complexity is relatively small we can expect the GA to execute each iteration fairly quickly.

5. Computational study

5.1. Results for small problems

Our GA heuristic was initially tested on 55 standard test problems (divided into six different sets) which are available from OR-Library (Beasley (1990,1996)) (email the message *mknapinfo* to *o.rlibrary@ic.ac.uk* or see the WWW address *http://mscmga.ms.ic.ac.uk/jeb/*

orlib/mknapinfo.html). These problems are small real-world problems consisting of $m = 2$ to 30 and $n = 6$ to 105 and their optimal values are known. Many of these problems have been used by other authors (Aboudi and Jörnsten (1994), Dammeyer and Voss (1993), Drexl (1988), Glover and Kochenberger (1996), Hoff, Løkketangen, and Mittet (1996), Khuri, Bäck, and Heitkötter (1994), Løkketangen and Glover (1996), Løkketangen, Jörnsten, and Storøy (1994), Thiel and Voss (1994)).

We solved these problems on our Silicon Graphics Indigo workstation (R4000, 100 MHz, 48 MB main memory), using both the general-purpose CPLEX mixed-integer programming (MIP) solver (version 4.0), and our GA heuristic which was coded in C. The GA heuristic was run once for each of the problems and each run terminated when 10^4 non-duplicate children had been generated. The results are shown in Table 1.

The first two columns in Table 1 indicate the problem set name and the number of problems in that problem set. The next two columns report for CPLEX the average solution time (in CPU seconds) and the average number of nodes searched (all problems were solved to optimality).

The final three columns in Table 1 report for our GA the average best-solution time, which is the time that the GA takes to first reach the final best solution, the average execution time, which is the total time that the GA takes before termination, and the number of problems for which the GA solution is optimal.

It is clear from Table 1 that our GA finds the optimal solution in all 55 test problems. However, it is *clearly apparent* from the computation times, both for CPLEX and for our GA, that these problems are either too small, or too easy, to draw any meaningful conclusions with respect to the effectiveness of our algorithm. However, we remark here that these standard problems did present some challenges for other GA heuristics, (e.g., Khuri, Bäck, and Heitkötter (1994), Thiel and Voss (1994)), as well as for several other heuristic methods, e.g., (Aboudi and Jörnsten (1994), Dammeyer and Voss (1993), Drexl (1988), Løkketangen and Glover (1996), Løkketangen, Jörnsten, and Storøy (1994)).

Table 1. Computational results for CPLEX and the GA—small problems.

Problem set name	No. of problems	CPLEX MIP solver		GA		
		Average solution time	Average number of nodes	A.B.S.T	A.E.T	NOPT
HP	2	0.3	73	0.4	2.6	2
PB	6	2.8	350	0.1	5.2	6
PETERSEN	7	0.2	75	0.2	3.2	7
SENTO	2	12.0	1149	0.3	11.5	2
WEING	8	0.6	263	0.4	4.3	8
WEISH	30	0.5	127	0.1	6.4	30
Average		1.1	200	0.2	5.6	

A.B.S.T = average best-solution time (CPU seconds).
A.E.T = average execution time (CPU seconds).
NOPT = number of problems for which the GA finds the optimal solution.

5.2. Problem generation

As far as we are aware, there exist no publically available standard MKP test problems of larger sizes and of more difficult types than the problems we have already considered in Table 1. Therefore, in order to better test the effectiveness of our GA, we independently generated a set of large MKP instances using the procedure suggested by Freville and Plateau (1994). The number of constraints m was set to 5, 10 and 30, and the number of variables n was set to 100, 250 and 500. Thirty problems were generated for each m-n combination, giving a total of 270 problems.

The r_{ij} were integer numbers drawn from the discrete uniform generator $U(0, 1000)$. For each m-n combination, the right-hand side coefficients (b_i's) were set using $b_i = \alpha \sum_{j=1}^{n} r_{ij}$ where α is a tightness ratio and $\alpha = 0.25$ for the first ten problems, $\alpha = 0.50$ for the next ten problems and $\alpha = 0.75$ for the remaining ten problems. The objective function coefficients (p_j's) were correlated to r_{ij} and generated as follows:

$$p_j = \sum_{i=1}^{m} r_{ij}/m + 500\, q_j \quad j = 1, \ldots, n,$$

where q_j is a *real* number drawn from the continuous uniform generator $U(0, 1)$. In general, correlated problems are more difficult to solve than uncorrelated problems (Gavish and Pirkul (1985), Pirkul (1987)).

We have made these test problems publically available (email the message *mknapinfo* to *o.rlibrary@ic.ac.uk* or see the WWW address *http://mscmga.ms.ic.ac.uk/jeb/orlib/ mknapinfo.html*).

5.3. Results for large problems

As before we solved these problems using both CPLEX and our GA heuristic. The results are shown in Table 2.

Since the optimal solution values for most of the problems in Table 2 are not known, the quality of the solutions generated (either by CPLEX or by our GA) are measured by the percentage gap between the best solution value found and the optimal value of the LP relaxation, i.e., 100 (optimal LP value—best solution value)/(optimal LP value).

The first three columns in Table 2 indicate the sizes (m and n) and the tightness ratio (α) of a particular problem structure, with each problem structure containing 10 problem instances. The next three columns report for CPLEX the average solution time (in CPU seconds), the average number of nodes searched and the average percentage gap. The average figures shown are the average values over 10 problem instances for each problem structure.

Computational results using the GA heuristic presented in this paper, are shown in the remaining columns of Table 2. The GA heuristic was run once for each of the 270 problems. Each run terminated when 10^6 non-duplicate children had been generated. The average percentage gap, the average best-solution time, which is the time that the GA takes to first reach the final best solution, and the average execution time, which is the total time that

Table 2. Computational results for CPLEX and the GA—large problems.

Problem			CPLEX MIP solver			GA			
			Average solution time	Average number of nodes	Average % gap	Average % gap	A.B.S.T	A.E.T	NOPT
m	*n*	α							
5	100	0.25	519.8	134869	0.99	0.99	9.6	345.9	10
		0.50	580.8	159731	0.45	0.45	23.5	347.3	10
		0.75	178.3	51777	0.32	0.32	26.9	361.7	10
5	250	0.25	25853.6	5018607	0.22	0.23	50.7	682.0	8
		0.50	31162.3	6228417	0.11	0.12	276.7	709.4	5
		0.75	12398.5	2457427	0.08	0.08	195.9	763.3	5
10	100	0.25	5417.8	1011031	1.56	1.56	97.5	384.1	10
		0.50	6086.8	1242041	0.79	0.79	97.3	418.9	9
		0.75	1241.1	299155	0.48	0.48	16.8	462.6	10
5	500	0.25	(981.6)	64398	4.68	0.09	264.6	1271.9	n/k
		0.50	(1048.0)	72744	5.02	0.04	291.3	1345.9	n/k
		0.75	(1129.8)	80101	2.49	0.03	386.2	1412.6	n/k
10	250	0.25	(1006.2)	69545	4.80	0.51	359.0	870.9	n/k
		0.50	(1054.7)	78502	5.41	0.25	342.2	931.5	n/k
		0.75	(1195.2)	81475	1.85	0.15	129.1	1011.2	n/k
10	500	0.25	(1738.2)	68723	4.88	0.24	702.5	1504.9	n/k
		0.50	(1651.2)	68929	5.50	0.11	562.2	1728.8	n/k
		0.75	(1795.0)	68492	2.33	0.07	937.6	1931.7	n/k
30	100	0.25	(1800.0)	99154	4.95	2.91	177.4	604.5	n/k
		0.50	(1800.0)	109163	4.79	1.34	118.0	782.1	n/k
		0.75	(1800.0)	110272	1.80	0.83	90.1	904.2	n/k
30	250	0.25	(1800.0)	39071	6.42	1.19	582.9	1499.5	n/k
		0.50	(1800.0)	49453	6.34	0.53	901.5	1980.0	n/k
		0.75	(1800.0)	48539	2.86	0.31	1059.3	2441.4	n/k
30	500	0.25	(1800.0)	22902	6.30	0.61	1127.2	2437.7	n/k
		0.50	(1800.0)	27401	6.42	0.26	1121.6	3198.9	n/k
		0.75	(1800.0)	27435	2.94	0.17	1903.3	3888.2	n/k
Average			4120.0	658865	3.14	0.54	438.9	1267.4	

n/k = not known.
A.B.S.T = average best-solution time (CPU seconds).
A.E.T = average execution time (CPU seconds).
NOPT = number of instances (out of 10) the GA finds the optimal solution.
A number in brackets for CPLEX indicates that the problems were terminated early.

the GA takes before termination, over 10 instances for each problem structure are reported. Computational times are given in CPU seconds. In the last column, the number of instances (out of 10) for which the GA finds optimal solution values, if known, is also shown.

Table 2 is split into two parts. The results for the first three m-n combinations relate to using CPLEX to solve problems to optimality. These figures indicate that it would not be computationally practicable on our Silicon Graphics Indigo workstation to use CPLEX to solve all problems to optimality (since CPLEX required a large amount of memory/time, especially when an excessive number of nodes were needed even to reach an integer solution of reasonable quality). For this reason the results shown for CPLEX in the second part of Table 2 for the remaining six m-n combinations are based on terminating whenever tree memory exceeds 42 MB or after 1800 CPU seconds.

These results illustrate that considerable computational effort is required by CPLEX to solve even the smallest problems in our test set. It is also clear that in all but two ($m = 5$, $n = 250$, $\alpha = 0.25, 0.50$) of the 27 problem structures in Table 2, the gap produced by the GA is at least as good as the gap produced by CPLEX. The average percentage gap (over all 270 test problems) is much lower for the GA (0.54%) than for CPLEX (3.14%).

We also observe that, both for the GA and for CPLEX, for the same m and α, as n increases, the problems become much harder and take noticeably more time to solve. Likewise, if m increases while fixing n and α, the difficulty also increases. Finally, the tightness ratio α has a predictable influence on the relative gaps; the smaller the α ratios (i.e., tighter constraints), the larger the gaps.

The results obtained by the GA indicate that the GA heuristic is very effective for large MKP instances of various structures, judging by the small percentage gaps shown. We should emphasise here that the reported percentage gaps are a measure of how close the heuristic solution is to the linear programming optimum, therefore much smaller gaps are expected if compared to the (unknown) integer optimum. This can be partially verified by comparing the average percentage gap columns given in Table 2. For those problems with known optimal values, the percentage gaps reported by CPLEX and the GA are similar, indicating that the heuristic solution values are indeed very close to the true optimal values. Finally, the ability of the GA to generate optimal solutions is demonstrated in the last column, in which the GA is able to find optimal values for many instances. The computation times for the GA are reasonable (less than one CPU hour in most cases). We should point out that a generous termination condition (10^6 non-duplicate children) was given for our GA just to demonstrate that the GA is capable of converging to high-quality solutions. Judging from the average best-solution times shown in Table 2 there is a potential to obtain similar results with a reduced number of iterations.

Finally we would remark that CPLEX had no difficulty solving the LP relaxation of the MKP for the problems shown in Table 2. The average solution time for the LP relaxation of the problems shown there was only 0.23 CPU seconds.

5.4. Performance comparison with other heuristics

Given the results presented in Table 1 for small problems we believe that there is little that can be gained by comparing heuristics with respect to their performance on this standard

set of small problems. Hence we shall, in this section, restrict our performance comparison solely to large problems.

5.4.1. Direct comparison. Table 3 directly compares the performance of the GA with other well-known heuristic methods by means of relative percentage gaps. More precisely, we have compared the performance of our GA with the heuristic of Magazine and Oguz (1984) (M&O), the heuristic of Volgenant and Zoon (1990) (V&Z) and the heuristic of Pirkul (1987) (MKHEUR) on the problems considered in Table 2.

For all these algorithms, since the original codes were not available to us upon request, we have coded the algorithms ourselves based on the descriptions of the methods outlined in the corresponding papers. These algorithms were then tested on each of our test problems, and average figures taken over 10 problem instances for each problem structure.

Table 3 clearly indicates the superiority of our GA over these other heuristic methods in terms of the quality of the solutions obtained. The GA generates solutions that on average have much smaller gaps than the other heuristics in all cases.

In terms of computation time, the GA required much more computation time than that required by the other heuristics. In our implementation the time complexity of M&O and V&Z was $O(mn)$, whilst that of MKHEUR was $O(mn^2)$. Consequently these algorithms required between 0.5 and 5 seconds to solve each of the 270 problems shown in Table 3. By contrast, as discussed above (see Section 4.6), the time complexity of our GA *per iteration* was approximately $O(mn)$.

However, since the computation times for the GA are within reasonable limits (see Table 2), we strongly feel that the time requirement is not a decisive factor here when judging the overall effectiveness of an algorithm. The significant improvement in solution quality which the GA made over the other heuristics, whilst requiring only modest computing efforts, favours the choice of the GA.

5.4.2. Indirect comparison. There are number of heuristics presented in the literature (additional to those already considered in Table 3) for which computational results for problems of a size at least equal to those shown in Table 2 have been presented.

Below we briefly highlight the most recent of these heuristics and give an indication of the results presented. However, given the fact that different papers consider different test problems, we would strongly caution the reader against attempting to pick the "best" heuristic. We would mention here that in making our test problems publically available we hope that future authors will be better able to compare their algorithms than can be achieved today.

- Battiti and Tecchiolli (1995), 8 problems of size $m = n = 500$, reactive tabu search found the best solution for 6 of the 8 problems
- Freville and Plateau (1994), 270 problems of sizes up to $m = 30$ and $n = 500$, average gap 1.91%
- Gavish and Pirkul (1985) and Freville and Plateau (1994), 270 problems of sizes up to $m = 30$ and $n = 500$, average gap 1.98%
- Glover and Kochenberger (1996), 24 problems of sizes up to $m = 25$ and $n = 500$, results better than incomplete tree search

Table 3. Performance comparison of the GA with other heuristic methods.

Problem			Average % gap			
m	*n*	*α*	M&O	V&Z	MKHEUR	GA
5	100	0.25	13.69	10.30	1.59	0.99
		0.50	6.71	6.90	0.77	0.45
		0.75	5.11	5.68	0.48	0.32
	Average		8.50	7.63	0.95	0.59
5	250	0.25	6.64	5.85	0.53	0.23
		0.50	5.22	4.40	0.24	0.12
		0.75	3.56	3.59	0.16	0.08
	Average		5.14	4.61	0.31	0.14
5	500	0.25	4.93	4.11	0.22	0.09
		0.50	2.96	2.53	0.08	0.04
		0.75	2.31	2.41	0.06	0.03
	Average		3.40	3.02	0.12	0.05
10	100	0.25	15.88	15.55	3.43	1.56
		0.50	10.41	10.72	1.84	0.79
		0.75	6.07	5.67	1.06	0.48
	Average		10.79	10.65	2.11	0.94
10	250	0.25	11.73	10.53	1.07	0.51
		0.50	6.83	5.92	0.57	0.25
		0.75	4.42	3.77	0.33	0.15
	Average		7.66	6.74	0.66	0.30
10	500	0.25	8.81	7.90	0.52	0.24
		0.50	5.71	4.14	0.22	0.11
		0.75	3.62	2.93	0.14	0.07
	Average		6.05	4.99	0.29	0.14
30	100	0.25	17.39	17.21	9.02	2.91
		0.50	11.82	10.19	3.51	1.34
		0.75	6.58	5.92	2.03	0.83
	Average		11.93	11.11	4.85	1.69
30	250	0.25	13.54	12.41	3.70	1.19
		0.50	8.64	7.12	1.53	0.53
		0.75	4.49	3.91	0.84	0.31
	Average		8.89	7.81	2.02	0.68
30	500	0.25	9.84	9.62	1.89	0.61
		0.50	7.10	5.71	0.73	0.26
		0.75	3.72	3.51	0.48	0.17
	Average		6.89	6.28	1.03	0.35
Average			7.69	6.98	1.37	0.54

- Hanafi and Freville (1997), 7 problems of sizes up to $m = 25$ and $n = 500$, results better than Glover and Kochenberger (1996)
- the GA heuristic presented in this paper, 270 problems of sizes up to $m = 30$ and $n = 500$, average gap 0.54%

6. Conclusions

In this paper we have presented a heuristic algorithm based on GAs for solving multidimensional knapsack problems. Most of the components of our GA are comparable to those used in a standard GA. Our approach differs from previous GA based techniques in the way that a heuristic operator which utilises problem-specific knowledge is incorporated. This operator guarantees that the child solutions can be made feasible. Our positive results support the idea that this is a desirable approach for tackling the constraints and the feasibility issue for the MKP.

On a large set of randomly generated problems, we have shown that the GA heuristic is capable of obtaining high-quality solutions for problems of various characteristics, whilst requiring only a modest amount of computational effort. Unlike the standard test set used by many authors, these test problems are of large sizes and are more difficult to solve, in the sense that we were not able to compute the optimal solutions and prove optimality.

Our algorithm was also directly compared with several other heuristic methods for the problem. Computational results showed that the GA gave superior quality solutions to these heuristics.

References

Aboudi, R. and K. Jörnsten. (1994). "Tabu Search for General Zero-One Integer Programs Using the Pivot and Complement Heuristic," *ORSA Journal on Computing* 6, 82–93.

Averbakh, I. (1994). "Probabilistic Properties of the Dual Structure of the Multidimensional Knapsack Problem and Fast Statistically Efficient Algorithms," *Mathematical Programming* 65, 311–330.

Bäck, T., D.B. Fogel, and Z. Michalewicz (eds.). (1997). *Handbook of Evolutionary Computation*. Oxford University Press.

Balas, E. (1965). "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Operations Research* 13, 517–546.

Balas, E. and C.H. Martin. (1980). "Pivot and Complement—A Heuristic for 0-1 Programming," *Management Science* 26, 86–96.

Battiti, R. and G. Tecchiolli. (1995). "Local Search with Memory: Benchmarking RTS," *OR Spektrum* 17, 67–86.

Beasley, D., D.R. Bull, and R.R. Martin. (1993a). "An Overview of Genetic Algorithms: Part I, Fundamentals," *University Computing* 15, 58–69.

Beasley, D., D.R. Bull, and R.R. Martin. (1993b). "An Overview of Genetic Algorithms: Part II, Research Topics," *University Computing* 15, 170–181.

Beasley, J.E. (1990). "OR-Library: Distributing Test Problems by Electronic Mail," *Journal of the Operational Research Society* 41, 1069–1072.

Beasley, J.E. (1996). "Obtaining Test Problems via Internet," *Journal of Global Optimization* 8, 429–433.

Beasley, J.E. and P.C. Chu. (1996). "A Genetic Algorithm for the Set Covering Problem," *European Journal of Operational Research* 94, 392–404.

Cabot, A.V. (1970). "An Enumeration Algorithm for Knapsack Problems," *Operations Research* 18, 306–311.

Charon, I. and O. Hudry. (1993). "The Noising Method: A New Method for Combinatorial Optimization," *Operations Research Letters* 14, 133–137.

Chu, P.C. (1997). "A Genetic Algorithm Approach for Combinatorial Optimisation Problems," Ph.D. Thesis, University of London.

Chu, P.C. and J.E. Beasley. (1995). "Constraint Handling in Genetic Algorithms: The Set Partitioning Problem," Imperial College, Working paper. To appear in *Journal of Heuristics*.

Chu, P.C. and J.E. Beasley. (1997). "A Genetic Algorithm for the Generalised Assignment Problem," *Computers and Operations Research* 24, 17–23.

CPLEX Optimization Inc. (1995). *Using the CPLEX Callable Library*. CPLEX Optimization Inc., Suite 279, 930 Tahoe Blvd., Bldg. 802, Incline Valley, NV 89451-9436, USA.

Crama, Y. and J.B. Mazzola. (1994). "On the Strength of Relaxations of Multidimensional Knapsack Problems," *INFOR* 32, 219–225.

Dammeyer, F. and S. Voss. (1993). "Dynamic Tabu List Management Using Reverse Elimination Method," *Annals of Operations Research* 41, 31–46.

Dantzig, G.B. (1957). "Discrete Variable Problems," *Operations Research* 5, 266–277.

Davis, L. and M. Steenstrup. (1987). "Genetic Algorithms and Simulated Annealing: An Overview." In L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann. pp. 1–11.

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

Dueck, G. (1993). "New Optimization Heuristics: The Grand Deluge Algorithm And the Record-to-Record Travel," *Journal of Computational Physics* 104, 86–92.

Dueck, G. and T. Scheuer. (1990). "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing," *Journal of Computational Physics* 90, 161–175.

Everett, H. (1963). "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," *Operations Research* 11, 399–417.

Fontanari, J.F. (1995). "A Statistical Analysis of the Knapsack Problem," *Journal of Physics A—Mathematical and General* 28, 4751–4759.

Fox, G.E. and G.D. Scudder. (1985). "A Heuristic with Tie Breaking for Certain 0-1 Integer Programming Models," *Naval Research Logistics Quarterly* 32, 613–623.

Freville, A. and G. Plateau. (1986). "Heuristics and Reduction Methods for Multiple Constraints 0-1 Linear Programming Problems," *European Journal of Operational Research* 24, 206–215.

Freville, A. and G. Plateau. (1990). "Hard 0-1 Multiknapsack Test Problems for Size Reduction Methods," *Investigacion Operativa* 1, 251–270.

Freville, A. and G. Plateau. (1994). "An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem," *Discrete Applied Mathematics* 49, 189–212.

Freville, A. and G. Plateau. (1997). "The 0-1 Bidimensional Knapsack Problem: Toward an Efficient High-Level Primitive Tool," *Journal of Heuristics* 2, 147–167.

Frieze, A.M. and M.R.B. Clarke. (1984). "Approximation Algorithms for the m-Dimensional 0-1 Knapsack Problem: Worst-Case and Probabilistic Analysis," *European Journal of Operational Research* 15, 100–109.

Gavish, B. and H. Pirkul. (1982). "Allocation of Databases and Processors in a Distributed Computing System." In J. Akoka (ed.) *Management of Distributed Data Processing*, North-Holland, pp. 215–231.

Gavish, B. and H. Pirkul. (1985). "Efficient Algorithms for Solving Multiconstraint Zero-One Knapsack Problems to Optimality," *Mathematical Programming* 31, 78–105.

Gilmore, P.C. and R.E. Gomory. (1966). "The Theory and Computation of Knapsack Functions," *Operations Research* 14, 1045–1075.

Glover, F. (1965). "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research* 13, 879–919.

Glover, F. (1977). "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences* 8, 156–166.

Glover, F. (1994). "Optimization by Ghost Image Processes in Neural Networks," *Computers and Operations Research* 21, 801–822.

Glover, F. and G.A. Kochenberger. (1996). "Critical Event Tabu Search for Multidimensional Knapsack Problems." In I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, pp. 407–427.

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

Hanafi, S. and A. Freville. (1997). "An Efficient Tabu Search Approach for the 0-1 Multidimensional Knapsack Problem," To appear in *European Journal of Operational Research*.

Hanafi, S., A. Freville and A.El. Abedellaoui. (1996). "Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem," In I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, pp. 449–465.

Hillier, F.S. (1969). "Efficient Heuristic Procedures for Integer Linear Programming with an Interior," *Operations Research* 17, 600–637.

Hoff, A., A. Løkketangen, and I. Mittet. (1996). "Genetic Algorithms for 0/1 Multidimensional Knapsack Problems." Working Paper, Molde College, Britveien 2, 6400 Molde, Norway.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press.

Khuri, S., T. Bäck, and J. Heitkötter. (1994). "The Zero/One Multiple Knapsack Problem and Genetic Algorithms," *Proceedings of the 1994 ACM Symposium on Applied Computing (SAC'94)*, ACM Press, pp. 188–193.

Kochenberger, G.A., B.A. McCarl, and F.P. Wymann. (1974). "A Heuristic for General Integer Programming," *Decision Sciences* 5, 36–44.

Land, A.H. and S. Powell. (1973). *Fortran Codes for Mathematical Programming Linear, Quadratic and Discrete*. Wiley.

Lee, J.S. and M. Guignard. (1988). "An Approximate Algorithm for Multidimensional Zero-One Knapsack Problems—a Parametric Approach," *Management Science* 34, 402–410.

Løkketangen, A. and F. Glover. (1996). "Probabilistic Move Selection in Tabu Search for Zero-One Mixed Integer Programming Problems." In I.H. Osman and J.P. Kelly (eds.), *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, pp. 467–487.

Løkketangen, A. and F. Glover. (1997). "Solving Zero-One Mixed Integer Programming Problems Using Tabu Search," to appear in *European Journal of Operational Research*.

Løkketangen, A., K. Jörnsten, and S. Storøy. (1994). "Tabu Search Within a Pivot and Complement Framework," *International Transactions of Operations Research* 1, 305–316.

Loulou, R. and E. Michaelides. (1979). "New Greedy-Like Heuristics for the Multidimensional 0-1 Knapsack Problem," *Operations Research* 27, 1101–1114.

Magazine, M.J. and O. Oguz. (1984). "A Heuristic Algorithm for the Multidimensional Zero-One Knapsack Problem," *European Journal of Operational Research* 16, 319–326.

Martello, S. and P. Toth. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons.

Michalewicz, Z. (1995). "A Perspective on Evolutionary Computation." In X. Yao (ed.), *Progress in Evolutionary Computation.* Springer-Verlag, pp. 73–89.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Nemhauser, G.L. and Z. Ullmann. (1969). "Discrete Dynamic Programming and Capital Allocation," *Management Science* 15, 494–505.

Pirkul, H. (1987). "A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem," *Naval Research Logistics* 34, 161–172.

Powell, D. and M.M. Skolnick. (1993). "Using Genetic Algorithms in Engineering Design Optimization with Nonlinear Constraints." In S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp. 424–431.

Reeves, C.R. (1993). *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific.

Richardson, J., M. Palmer, G. Liepins, and M. Hillard. (1989). "Some Guidelines for Genetic Algorithms with Penalty Functions." In J. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, pp. 191–197.

Rinnooy Kan, A.H.G., L. Stougie, and C. Vercellis. (1993). "A Class of Generalized Greedy Algorithms for the Multi-knapsack Problem," *Discrete Applied Mathematics* 42, 279–290.

Rudolph, G. and J. Sprave. (1995). "A Cellular Genetic Algorithm with Self-adjusting Acceptance Threshold," *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*. IEE, London, pp. 365–372.

Rudolph, G. and J. Sprave. (1996). "Significance of Locality and Selection Pressure in the Grand Deluge Evolutionary Algorithm," In H.M. Voigt, W. Ebeling, I. Rechenberg, and H.P. Schwefel (eds.), *Parallel Problem Solving from Nature IV. Proceedings of the International Conference on Evolutionary Computation*. Lecture Notes in Computer Science, Springer, pp. 686–694.

Schilling, K.E. (1990). "The Growth of *m*-Constraint Random Knapsacks," *European Journal of Operational Research* 46, 109–112.

Senju, S. and Y. Toyoda. (1968). "An Approach to Linear Programming with 0-1 Variables," *Management Science* 15, 196–207.

Shih, W. (1979). "A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem," *Journal of the Operational Research Society* 30, 369–378.

Smith, A.E. and D.M. Tate. (1993). "Genetic Optimization Using a Penalty-Function." In S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp. 499–505.

Soyster, A.L., B. Lev, and W. Slivka. (1978). "Zero-One Programming with Many Variables and Few Constraints," *European Journal of Operational Research* 2, 195–201.

Szkatula, K. (1994). "The Growth of Multi-constraint Random Knapsacks with Various Right-hand Sides of the Constraints," *European Journal of Operational Research* 73, 199–204.

Szkatula, K. (1997). "The Growth of Multi-constraint Random Knapsacks with Large Right-hand Sides of the Constraints," *Operations Research Letters* 21, 25–30.

Thiel, J. and S. Voss. (1994). "Some Experiences on Solving Multiconstraint Zero-One Knapsack Problems with Genetic Algorithms," *INFOR* 32, 226–242.

Toyoda, Y. (1975). "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems," *Management Science* 21, 1417–1427.

Volgenant, A. and J.A. Zoon. (1990). "An Improved Heuristic for Multidimensional 0-1 Knapsack Problems," *Journal of the Operational Research Society* 41, 963–970.

Weingartner, H.M. (1967). *Mathematical Programming and the Analysis of Capital Budgeting Problems*. Chicago: Markham Publishing.

Weingartner, H.M. and D.N. Ness. (1967). "Methods for the Solution of the Multidimensional 0/1 Knapsack Problem," *Operations Research* 15, 83–103.

Zanakis, S.H. (1977). "Heuristic 0-1 Linear Programming: An Experimental Comparison of Three Methods," *Management Science* 24, 91–104.