Comparing Representations and Recombination Operators for the Multi-Objective 0/1 Knapsack Problem

Christine L. Mumford (Valenzuela)
Department of Computer Science
Cardiff University
CF24 3XF, United Kingdom
christine@cs.cf.ac.uk

Abstract- The multiple knapsack problem (MKP) is a popular test-bed for researchers developing new Pareto-based multi-objective evolutionary algorithms. This paper explores a range of different representations and operators for the MKP which have been adapted from the single objective case. Results indicate that order-based approaches are superior to binary representations for the problem instances considered here.

1 Introduction

The 0/1 knapsack problem involves selecting from among various items those that will be most profitable, given the knapsack has limited capacity. Knapsack problems have been extensively studied and provide a useful test-bed when developing new optimization techniques. Their simple structure, and close relationship with many real industrial problems make them an ideal choice, and they have certainly proven to be very popular with researchers in the field of evolutionary computation. The 0/1 knapsack problem, in its simplest form, consists of a set of objects $O = \{o_1, o_2, o_3, ..., o_n\}$ and a knapsack of capacity C. Each object o_i has an associated profit p_i and weight w_i. The objective is to find a subset $S \subseteq O$ such that the weighted sum over the objects in S does not exceed the knapsack capacity and yields a maximum profit.

A number of representations and genetic operators have been developed over the years for the simple 0/1 knapsack problem [4, 5, 9, 10], and comparative studies indicate that some representations give much better results than others, although performance does depend, to some extent, on characteristics of the chosen data sets. Researchers developing evolutionary algorithms (EAs) for multi-objective optimization problems, however, have favored one particular approach over all others when solving 0/1 multiple knapsack problem - the binary (or bit string) representation with greedy repair [19] (described later). This paper concentrates on extending the main representations and operators found useful for the single objective problem to the multi-objective case and comparing their performance.

Multi-objective optimization problems involve the simultaneous optimization of several (often competing) objectives, and usually there is no single optimal solution. Instead, multi-objective optimization problems tend to be characterized by a set of alternative solutions, each of which must be considered equivalent in the absence of further information regarding the relative importance of each of the objectives in the solution vectors. Such a solution set is called the *Pareto-optimal set*, and the objective values in the set are located at the *Pareto front*. Pareto-optimal solutions are *non-dominated solutions* in the sense that it is not possible to improve the value of any one of the objectives, in such a solution, without simultaneously degrading the quality of one or more of the other objectives in the vector.

The multi-objective, or multiple knapsack problem (MKP) is a generalization of the simple 0/1 knapsack problem. The 0/1 MKP involves m knapsacks of capacities $c_1, c_2, c_3, ..., c_m$. Every selected object must be placed in all m knapsacks, although neither the weight of an object o_i nor its profit is fixed, and will probably have different values in each knapsack. The present study is restricted to problems involving two knapsacks.

To avoid confusion, it is worth mentioning a related problem which is also known as the multiple knapsack problem. In this version, although each item can have a different weight in each knapsack, it always has the same profit. This is not a multi-objective problem, although it has multiple knapsacks and is thus multi-constrained [7, 13].

For the purposes of the present study, the test problems are taken from Zitzler and Thiele [19]:

http://www.tik.ee.ethz.ch/zitzler/testdata.html.

These problems were randomly generated with uncorrelated profits and weights, and the knapsack capacities were set to half the total weight, corresponding to each knapsack, of all the items. Problem sizes vary between 100 and 750 items. Pareto optimal solutions for three of the problems (used in some 2D plots) are also obtained from the above web site.

2 Representations and Operators for the 0/1 Knapsack Problem

Representations and operators used for the single objective knapsack problem are easily adapted for knapsack problems with two or more objectives. One of the major difficulties with single and multi-objective versions alike, however, is handling the capacity constraint. Section 2.1 summarizes the various representations and operators that have

been used in the single objective case, and Section 2.2 then discusses how some of these approaches have been adapted for the multiple objective case in the present study. Several methods for handling capacity constraints are explored. All experiments with the various representations and genetic operators for the multi-objective case take place within the framework of the SEAMO algorithm [17], which is described in Section 4.

2.1 Representations for the Single Objective Problem

Robert Hinterding [5] provides the following broad classification of representational techniques for the 0/1 knapsack problem:

- **Binary representation** – where bit i is set if the ith item from the list of items is included in the knapsack.
- **Numeric representation** – here the genes are numbers instead of bits, and a decoder is used to produce an ordering of the items.
- **Symbolic representation** – the genes represent the items themselves in a list. The list can either contain a subset of the items, or may consist of a permutation of all the items. In the latter case a heuristic is used to select items from the list to fill the knapsack.

When the binary representation is used, the capacity constraint can be handled either by penalizing solutions which violate the constraint, [5, 9, 10, 14] or by using a heuristic mechanism to correct any violations, [5, 9, 10]. Numeric representations [5, 9, 10] and symbolic representations [4, 5], on the other hand, employ heuristic decoders to ensure that only legal knapsacks are produced.

2.2 Representations for the Multiple Objective Problem

For the MKP most researchers to date (for example, [6, 8, 19, 20]) have favored the binary representation and a greedy repair heuristic adapted from [9] by Zitzler and Thiele [19]. The present paper concentrates on comparing the performance of an evolutionary algorithm using this most popular approach to the performance of the same EA using other approaches, including the use of penalty functions. According to Michalewicz and Arabas [9, 10] algorithms using a penalty function with the binary representation perform well on some single objective problems, although they are reported to perform exceptionally badly on problems with *restricted knapsack capacity*, i.e. only a few of the items can be packed. In addition to exploring a number of alternatives based on the binary representation, the present study also includes an order-based representation introduced in [4] for the single objective problem and adapted by the present author for the 0/1 MKP [17]. Numeric representations for the 0/1 MKP are not covered, however, due to space limitations. In any case, previous researchers have reported rather poor results for these techniques on the single objective problem, [5, 10].

2.2.1 The Binary Representation with a Penalty Function

A binary representation consists of a vector, $\mathbf{x} = (x_1, x_2, x_3, ..., x_n)$, with $x_i = 1$ if item i is included in the knapsack(s), and $x_i = 0$, otherwise. Generating bits at random or using EAs, can easily produce violated capacity constraints, so that knapsacks over-fill. The simplest way to deal with violated constraints is to apply a penalty function.

Michalewicz and Arabus [9, 10] and Olsen [14] report good results when penalties are used with binary representations of certain 0/1 knapsack problems in the single objective case. Extending the penalty approach to the MKP requires that penalties are applied to m knapsacks, any or all of which may be over-filled. Following some initial experimentation, the penalty function chosen for the present study is derived from the linear model of Michalewicz and Arabus. In this model the penalty function grows in a linear fashion, in proportion to the extent of the constraint violation. The alternative schemes proposed by Michalewicz and Arabas, involving logarithmic or quadratic growth, did not work very well when adapted for the multi-objective instances explored in the present paper. Logarithmic growth produced penalties that were far too small for the instances of the MKP explored here, and the resulting populations consisted entirely of illegal solutions with violated constraints. Quadratic growth, on the other hand, produced penalties that were too large and overwhelmed the objective functions. The linear model adopted is defined below:

$$Pen_j(\mathbf{x}) = \rho_j.(\textstyle\sum_{i=1}^{n} x_i.w_{ij} - C_j)$$
$$eval_j(\mathbf{x}) = \textstyle\sum_{i=1}^{n} x_i.p_{ij} - Pen(x_j)$$

where $Pen_j(\mathbf{x})$ is the penalty applied to knapsack j, p_{ij} and w_{ij} represent the profit and weight respectively of the ith item in the jth knapsack, $\rho_j = max_{i=1..n}\{p_{ij}/w_{ij}\}$, C_j is the capacity of knapsack j, and $eval_j(\mathbf{x})$ is the adjusted value of the total profit in knapsack j.
Note: Penalty functions are applied only to knapsacks which are over capacity.

2.2.2 The Binary Representation with Repair

As an alternative to penalties, heuristics may be used to deal with capacity constraints. These will either form part of a decoder, or be incorporated into a repair mechanism. A decoder ensures that only legal solutions are generated, using a step-by-step procedure to add items one at a time to initially empty knapsacks, stopping before any constraints are violated. A repair mechanism, on the other hand, begins

with knapsacks already packed with all the items that have their bits set, and sequentially removes items from the solution until all constraints are satisfied, and no knapsack is over-filled. Hinterding [5] has experimented with a decoder based on a first fit algorithm, for the single objective case, but repair methods based on the work of Michalewicz and Arabas would seem to be far more popular. For the MKP the repair method adapted by Zitzler and Thiele [19] is the approach favored by researchers. This repair algorithm removes items, one at a time, from the solution until all the capacity constraints are satisfied. The order in which the items are deleted is determined by the maximum profit/weight ratio per item, with the item which is least profitable, per unit weight, being the first to be removed. Zitzler and Thiele use chromosomes repaired in this way to ensure that only legal solutions are produced. They do not write the repaired chromosomes back into the population, however.

Michalewicz and Arabas experimented by writing back to the population various proportions (from 0 % to 100 %) of the repaired chromosomes in the single objective case, and found that whether or not the chromosomes were replaced made no significant difference to the result. Experiments carried out by the present author on some multi-objective problems confirm these findings, although the results are omitted due to space limitations. None of the results presented in this paper rely on the replacement of repaired chromosomes into the population.

Two variations of repair have been coded for the present study, both depend on sequential removal of objects with the least profitable, per unit weight, being deleted first. However, the order in which the items are deleted is slightly different. The two methods are:

- the mechanism due to Zitzler and Thiele where items are deleted according to their *maximum* profit/weight ratio,
- and an alternative mechanism where items are deleted according to their *average* profit/weight ratio.

One-point crossover and point mutation are used for all experiments using the binary representation, for penalty as well as repair methods.

2.2.3 The Order-Based Representation

In the order-based representation the genes represent the items themselves and the chromosomes consist of orderings of all the items. Because every item is included on each chromosome, a decoder is required to produce legal solutions. Hinterding [4] used a first fit heuristic for his order-based representation in the single objective case. Starting with an empty knapsack, he selected items in sequence from a permutation list, starting with the first item on the list, then working through to the second, then the third and so on. Whenever inclusion of an item from the list would result in a constraint violation, that item was skipped over and the next item tried. Adapting a decoder based on the first fit algorithm for the multiple knapsack problem simply requires that the constraints are tested for all the knapsacks each time an item is considered for inclusion in the solution. The present study explores two decoders: a decoder based on the first fit heuristic, as used by Hinterding, and one based on the next fit heuristic, as used previously by the current author [17]. The next fit heuristic is very similar to the first fit heuristic: it selects items in sequence from the permutation list. It differs from the first fit heuristic only when it encounters an item that cannot be accommodated, at which stage the next fit heuristic halts, rather than explore the list any further.

Cycle crossover, CX, [12] is used as the recombination operator for the order based experiments, and the mutation operator swaps two arbitrarily selected objects within a single permutation list. CX was selected as the recombination operator because it produced better results than other permutation crossovers in some test runs (see Figure 1). CX transmits absolute positions of objects in the permutation lists from the parents to the offspring. Neither edge based nor order based operators would seem to be appropriate here, for a set membership problem such as this. Other recombination operators tried were the following: partially matched crossover (PMX) [3], a version [11] of order crossover [1] that preserves absolute positions better than the original, and uniform order based crossover (UOBX) [2].

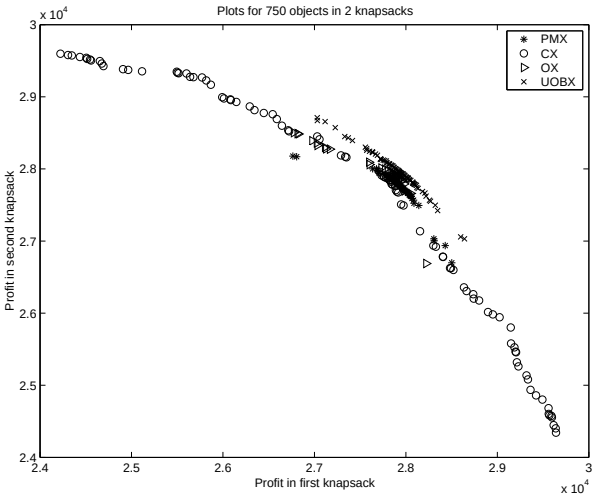

Figure 1: Non-dominated solutions for different crossovers

The comparative runs shown in Figure 1 support the choice of CX as the recombination operator. This figure compares the performance of the four above mentioned

order-based operators on Zitzler and Thiele's 750 object, 2 knapsack problem. Populations of 250 were used for these experiments, and each of the test runs was terminated after 15,000 generations. Clearly the plot using CX gives a much more diverse set of solutions than any of the other plots, although the UOBX solutions are of slightly better quality. The quality of the CX solutions, relative to UOBX, was found to improve when the population size was increased, however.

3 Performance Measures Used

Presenting and analyzing the results of multi-objective optimization is a challenge. Algorithms that solve single objective problems produce single answers that are easy to present graphically and lend themselves readily to statistical analysis. Multi-objective techniques, on the other hand, usually produce a large set of answers that are far more difficult to handle, and there is no general agreement amongst researchers on which performance measures are best for such problems. In the present study the $\mathcal{C}$, and $\mathcal{S}$ metrics of Zitzler and Thiele [19] are used, together with some 2D plots. The $\mathcal{C}$ and $\mathcal{S}$ metrics are described below.

3.1 The $\mathcal{C}$ Metric

This is a measure of the coverage of two sets of solution vectors. Let $X', X'' \subseteq X$ be two sets of solutions vectors. The function $\mathcal{C}$ maps the ordered pair (X', X'') to the interval $[0, 1]$

$$\frac{\mathcal{C}(X',X'') = |\{a'' \in X''; \exists\, a' \in X' \colon a' \succeq a''\}|}{|X''|}$$

The value $\mathcal{C}(X', X'') = 1$ means that all the points in X'' are dominated by or equal to points in X', (i.e. all the points in X'' are *weakly dominated* by points in X'). The opposite, $\mathcal{C}(X', X'') = 0$, represents the situation when none of the points in X'' is weakly dominated by X'. Note that both $\mathcal{C}(X', X'')$ and $\mathcal{C}(X'', X')$ have to be considered, since $\mathcal{C}(X', X'')$ is not necessarily equal to $1 - \mathcal{C}(X'', X')$ (i.e. when many solutions in X' and X'' neither dominate nor are they dominated by solutions in the alternative set set).

The $\mathcal{C}$ metric can be used for both maximization and minimization problems, and can be used to tell which of two algorithms produces the better outcomes. It does not, however, give any indication as to how much better these outcomes are. Results are quoted as percentages in the present paper.

3.2 The $\mathcal{S}$ Metric

This is a measure of the size of the dominated space, or hypervolume. Let $X' = (x_1, x_2, \ldots, x_l) \subseteq X$ be a set of l solution vectors. The function $\mathcal{S}(X')$ gives the volume enclosed by the union of the polytypes $p_1, p_2, \ldots, p_l$, where each p_i is formed by the intersection of the following hyperplanes arising out of x_i, along with the axes: for each axis in the objective space, there exists a hyperplane perpendicular to the axis and passing through the point $(f_1(x_i), f_2(x_i), \ldots, f_k(x_i))$. In the two dimensional case, each p_i represents a rectangle defined by the points $(0, 0)$ and $(f_1(x_i), f_2(x_i))$. The size of the dominated space is quoted as a percentage of the reference volume between the origin and a utopia point, defined as the profit sums of all items in each objective in [20]. The $\mathcal{S}$ metric can be used only for maximization problems.

4 The SEAMO Algorithm

A Simple Evolutionary Algorithm for Multi-objective Optimization (SEAMO) [17] was implemented and used for all the experiments reported in this paper. SEAMO is an uncomplicated, steady-state algorithm which relies on a few very basic techniques. In an earlier study [17] some excellent results were obtained with SEAMO for the multi-objective 0/1 knapsack problem.

Using SEAMO as the framework for the experiments in the current study, it is possible to concentrate on the representational issues, without the need to fine-tune very many parameters. For example, there is no auxiliary population in SEAMO and selection is uniform and not based on the values of (arbitrary) fitness functions. Crossover is applied at 100 % and mutation is always exactly one mutation per individual (these rates of crossover and mutation rate have produced good and reliable results for the present author, over a number of different applications, for example see [15, 16, 18]). Thus the only parameters we need be concerned with are population sizes and stopping criteria. The Simple Evolutionary Algorithm for Multi-objective Optimization (SEAMO), is outlined in Figure 2.

The algorithm sequentially selects every individual in the population to serve as the first parent once, pairing it with a second parent that is selected at random (uniformly). A single crossover is then applied that produces one offspring, and this is followed by a single mutation.

The replacement of a parent by its offspring is considered whenever an offspring is deemed to be superior to that parent. The offspring is compared, first of all, to the first parent. If the offspring is considered to be better than the first parent, it may replace it in the population. If this is not the case, then the same test is made between the offspring and the second parent. For this purpose superiority is normally judged as a dominance relationship, i.e. if an offspring dominates its parent, it may replace it in the population. The replacement of population members by dominating offspring ensures that the solution vectors move closer to the Pareto front as the search progresses. To addition-

```
Procedure SEAMO
Begin
    Generate N random individuals {N is the population size}
    Evaluate the objective vector for each population member and store it
    Record the global best-so-far for each objective function in the vector
    Repeat
        For each member of the population
            This individual becomes the first parent
            Select a second parent at random
            Apply crossover to produce offspring
            Apply a single mutation to the offspring
            Evaluate the objective vector produced by the offspring
            If any element of the offspring's objective vector improves on a
                    global best-so-far
                Then the offspring replaces one of the parents
                        (or occasionally another individual)
                                and best-so-far is updated
            Else If the offspring dominates one of the parents
                Then it replaces it
                        (unless it is a duplicate, then it is deleted)
        Endfor
    Until stopping condition satisfied
    Print all non-dominated solutions in the final population
End
```

Figure 2: Algorithm 1 A Simple Evolutionary Algorithm for Multi-objective Optimization (SEAMO)

ally ensure an improved range of coverage, the dominance condition is relaxed whenever a new global best value is discovered for any of the individual components of the solution vector (i.e. for improved maximum profits in individual knapsacks). Care has to be taken, however, to ensure that global best values for other components (i.e. maximum profits in other knapsacks) are not lost when a dominance condition is relaxed. Ensuring elitism (i.e. that the best solutions are not lost) at this level is straightforward if multi-objective optimization is restricted to two components in the solution vector. Whenever an offspring produces an improved global best for either of the components, if the global best for the second component happens to occur in one of the parents, the offspring will simply replace the other parent.

Before a final decision is made on replacement, a solution vector for a dominating offspring will be compared with all the solution vectors in the current population. If the offspring's solution vector is duplicated elsewhere in the population, the offspring dies and does not replace its parent.

5 Experimental Method

For most of the experiments populations of 250 are used, regardless of the problems size, and the EAs are each run for 5,000 generations. Long runs are used to ensure convergence, although good results can be obtained by running for a much shorter time. Each experiment consists of 30 replicate runs, initialized with different random seeds. Results are averaged to obtain representative $\mathcal{C}$ and $\mathcal{S}$ metrics, and tests for statistical significance carried out on the raw $\mathcal{C}$ and $\mathcal{S}$ values, where practical. A goodness of fit test (*lillietest* from the MATLAB statistical toolbox) is used to ensure that the raw values are a reasonable fit to a normal distribution, prior to significance testing.

The 2D plots are obtained by combining all 30 results files, for each experiment, and extracting the non-dominated solutions from the combined results.

6 Results

Tables 1 and 2 summarize the dominated space and coverage, respectively, produced by the various representations and operators. In the tables 'R_MAX' and 'R_AVE' denote the algorithms that use the binary representation and a repair mechanism. 'MAX' is the scheme based on the maximum profit/weight ratio, and 'AVE' is the scheme that uses the average profit/weight ratio. 'OBFF' stands for the order-based representation with the first fit decoder, and 'OBNF' stands for the order-based representation with the next fit decoder. The results of the experiments with the penalty function are recorded in the rows labelled 'PEN'. The test problems kn*x.y* denote knapsack problems with *x* items and *y* objectives (or knapsacks).

Algorithm	kn100.2	kn250.2	kn500.2	kn750.2
R_MAX	55.28	50.54	49.52	48.64
R_AVE	55.14	50.44	49.52	48.82
OBFF	56.03	52.10	51.62	51.14
OBNF	55.88	51.82	51.42	51.15
PEN	54.08	49.14	48.53	47.68

Table 1: Average percentage of dominated space , $\mathcal{S}$

Table 1 indicates that order-based techniques tend to produce better results that dominate more space than solutions produced using binary string representations. Also the order-based algorithm with the first fit decoder appears to perform marginally better than the version that uses the next fit decoder. Repair mechanisms appear to do better than penalty functions, but not as well as the order-based algorithms. The repair heuristic based on average profit/weight ratios gives slightly better results than the one based on maximum profit/weight ratios.

Separate one-way analysis of variance tests (*anova1* from MATLAB) of the $\mathcal{S}$ values for individual runs on kn100.2, kn250.2, kn500.2 and kn750.2, demonstrate the differences in performances observed in Table 1 to be highly significant at the 0.0001 level.

Table 2 shows the coverage of various combinations of

Coverage $(A \succeq B)$					
Algorithm		Test problems			
A	B	kn100.2	kn250.2	kn500.2	kn750.2
R_MAX	R_AVE	63.6	4.7	2.2	1.5
	OBFF	46.1	5.6	15.6	29.2
	OBNF	54.8	14.1	28.2	35.9
	PEN	65.6	52.8	84.9	79.5
R_AVE	R_MAX	45.6	79.9	81.2	68.8
	OBFF	30.6	33.4	40.0	39.4
	OBNF	41.9	48.7	46.6	43.2
	PEN	49.1	89.6	91.1	95.4
OBFF	R_MAX	73.2	77.9	49.9	3.5
	R_AVE	75.1	30.4	5.6	0.3
	OBNF	71.9	62.2	65.8	62.6
	PEN	76.4	88.1	85.0	49.9
OBNF	R_MAX	58.1	59.1	10.8	1.2
	R_AVE	62.9	11.2	1.4	0.1
	OBFF	48.9	14.7	13.2	15.5
	PEN	60.6	75.8	67.9	18.7
PEN	R_MAX	50.8	27.6	1.3	1.7
	R_AVE	55.8	1.1	0.1	0.0
	OBFF	39.3	0.8	1.3	7.8
	OBNF	45.5	4.6	5.3	17.3

Table 2: Average values for Coverage $(A \succeq B)$

pairs of solutions sets, averaged over 30 replicate runs. Coverage $(A \succeq B)$ shows the proportion of points produced by algorithm B that are weakly dominated by points produced by algorithm A. Each of the 30 results files obtained by running algorithm A is paired with exactly one of the results files generated by algorithm B, and the coverage, $(A \succeq B)$ and $(B \succeq A)$, evaluated for each of the 30 pairs of files. The table records the averages of these values. (Note: statistical significance tests have not been attempted for Coverage at this stage, because of the large quantity of correlated data pairs involved. This issue is addressed, however, to some extent at the end of this section, when tests are carried out to compare just two instead of five approaches.)

From Table 2 the repair mechanism that uses the average profit/weight ratio produces much better results than the one that uses the maximum profit/weight ratio, except in the case of the smallest problems, kn100.2. For the order-based representations the first fit decoder would appear to work much better than the next fit decoder. The penalty method performs worst of all.

In general, the $\mathcal{S}$ and $\mathcal{C}$ results appear to reinforce each other. Both metrics indicate that the repair method based on the average profit/weight ratio is better than the one based on the maximum profit/weight ratio, and also suggest that when an order-based representation is used, a decoder using a first fit heuristic is preferable to a decoder based on a next fit decoder. $\mathcal{S}$ and $\mathcal{C}$ metrics show that repair methods and decoders work better than the penalty function in these experiments.

When the results are examined more closely, however, it is not easy to determine whether it is better to use an order-based approach or a binary string with repair. Order-based methods produce better values for $\mathcal{S}$, and thus would appear to have a better spread of points. When coverage $\mathcal{C}$, is examined though, it would appear that the order-based approaches are not so good when faced with the larger problems, such as kn500.2 and kn750.2.

Figure 3 gives a pictorial summary of the results produced by three of the approaches: the binary string representation with penalty function, the binary string representation with repair based on the average profit/weight ratios, and the order-based representation with the first fit decoder. For each algorithm all 30 results files have been combined and the non-dominated vectors extracted. Figure 3 clearly indicates that, although the order-based approach produces a much better spread of results than the other algorithms, the binary string approach with repair achieves higher quality solutions (albeit far fewer of them) on the larger problem instances. One could speculate that a larger population may allow the order-based technique to better exploit its greater spread of approximated Pareto points.

To test the hypothesis that larger populations may be required if order-based approaches are to deal effectively with larger problems, some extra experiments were performed. Figure 4 summarizes the results for sets of 30 replicate runs using populations of 500 and 1000 for kn500.2 and kn750.2, respectively. Clearly the order-based results now look much better than the results using the binary string and repair. The coverage results also favor the order-based approach over the binary string approach: for kn500.2 Coverage $(OBFF \succeq R_AVE) = 40.0$ and Coverage $(R_AVE \succeq OBFF) = 24.9$, for kn750.2 Coverage $(OBFF \succeq R_AVE) = 87.2$ and Coverage $(R_AVE \succeq OBFF) = 2.8$. Student t-tests (*ttest* from MATLAB) on the distribution of the difference, $(OBFF \succeq R_AVE) - (R_AVE \succeq OBFF)$, bear this out by showing that the mean of this distribution differs significantly from zero at the 0.001 level.

7 Conclusion and Future work

Using a simple Pareto-based evolutionary algorithm, this paper explores various representations and recombination operators for the 0/1 multiple knapsack problem. The approaches tested are adapted from the single objective version of the problem, and cover binary strings and order-based representations as well as various techniques for dealing with capacity constraints.

The findings generally favor order-based approaches, using cycle crossover, over binary string representations, although binary string techniques with repair do rather better than those that use a penalty function. The order-based approaches appear to converge more slowly and generate a

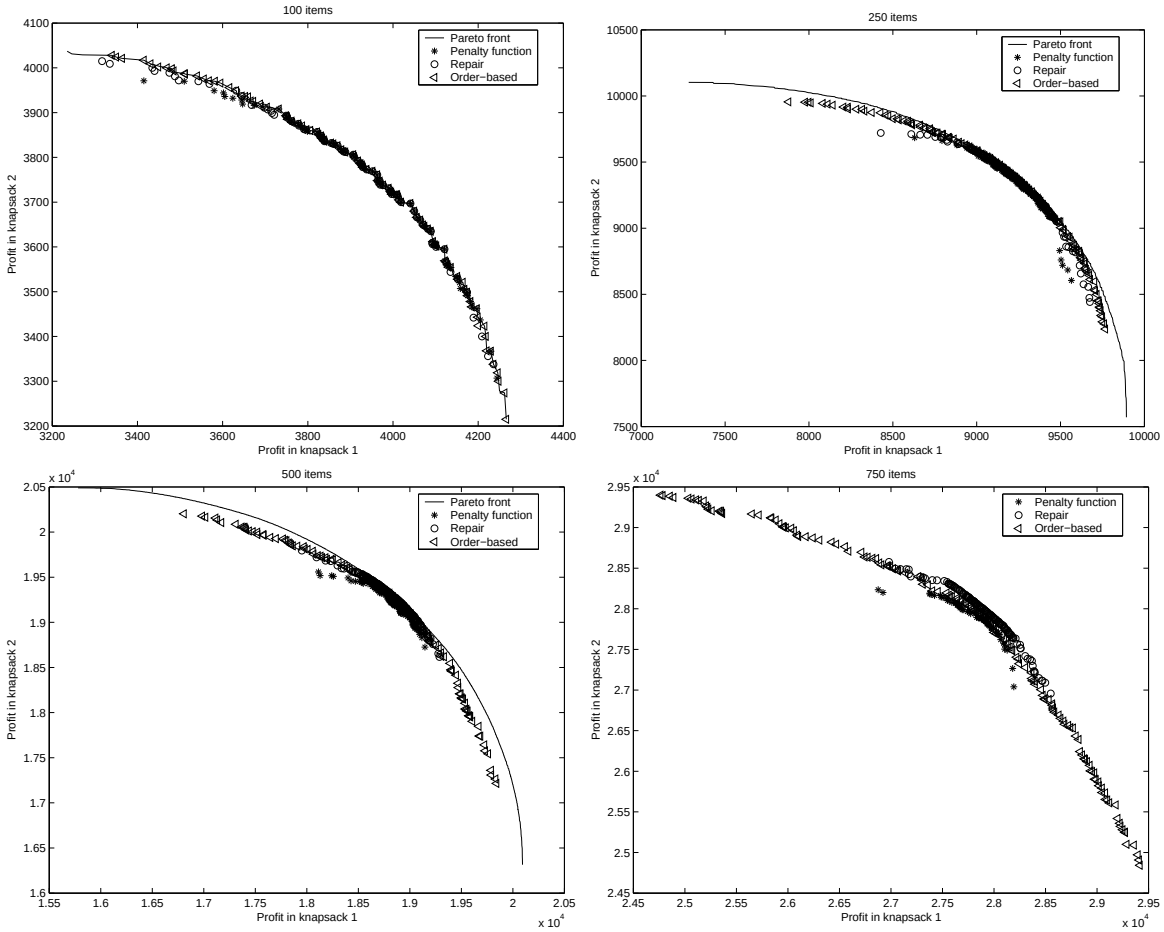

Figure 3: Comparing various representations, with populations of 250

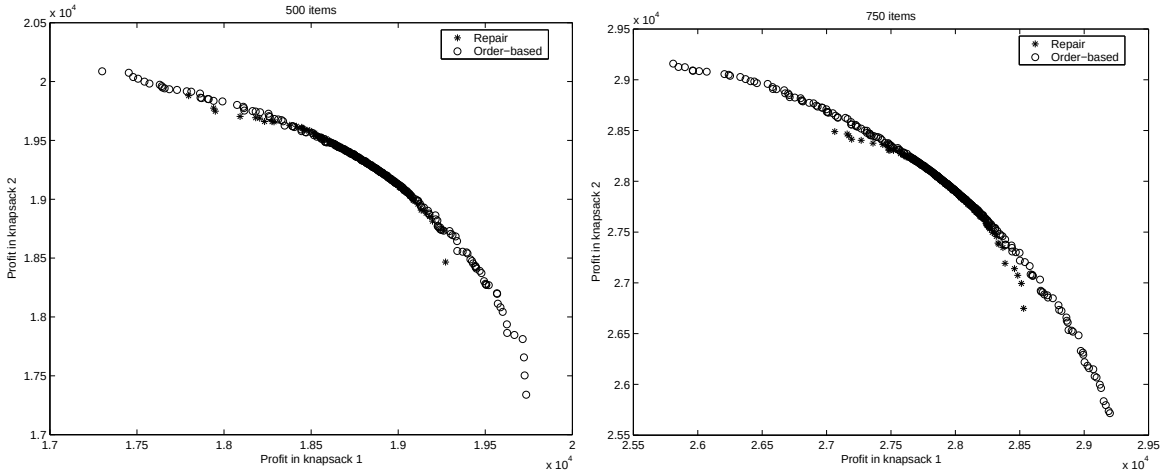

Figure 4: Comparing different approaches using larger populations

more widely spread set of solutions than binary string representations are able to produce, although an adequate size of population is essential, for those solutions to be of really high quality. Binary string methods can produce a narrow set of reasonably high quality solutions fairly quickly, even when a small population is used. They seem incapable of producing much in the way of improvement, however, regardless of how large the population, or how long the run time. There are indications that order-based approaches may also produce poorly spread results, depending on the choice of genetic operators. Sample runs suggest that cycle crossover works better than other order-based recombination operators, at least in conjunction with the SEAMO algorithm.

Future work will concentrate on testing the various approaches on a wider range of knapsack problems, with more objectives and more restricted capacities, for example. Other plans include the enhancement of the repair heuristics, along the lines suggested by Jaszkiewicz in [6] - after taking items out of an overfull knapsack he suggested trying to put some small ones back. It would also be very interesting to see how the various representations and operators explored here would perform on the MKP if multi-objective EAs other than SEAMO are tried.

Bibliography

[1] L. Davis, Applying adaptive algorithms to epistatic domains, *Proceedings of the Joint International Conference on Artificial Intelligence*, pp. 162–164, 1985.

[2] L. Davis, Order-based genetic algorithms and the graph coloring problem, *Handbook of Genetic Algorithms* pp. 72–90, Van Nostrand Reinhold, New York, 1991.

[3] D. E. Goldberg and R. Lingle, Alleles, loci and the TSP, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, pp. 154–159, 1985.

[4] R. Hinterding, Mapping, order-independent genes and the knapsack problem, *Proceedings of the first IEEE Conference on Evolutionary Computation*, Orlando, Florida, 1994 pp. 13–17.

[5] R. Hinterding, Representation, constraint satisfaction and the knapsack problem, *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, Vol. 2, pp. 1286 – 92, 1999.

[6] A. Jaskiewicz, On the performance of multiple objective genetic local search on the 0/1 knapsack problem - a comparative experiment, *IEEE Transactions on Evolutionary Computation*, 6(4), pp. 402–412, 2002.

[7] S. Khuri, T. Bäck and J. Heitkötter, The zero/one multiple knapsack problem and genetic algorithms, *Proc. of the 1994 ACM Symposium on Applied Computing*, pp. 188–193, ACM Press, 1994.

[8] J. D. Knowles and D. W. Corne, M-PAES: a memetic algorithm for multiobjective optimization, *Congress on Evolutionary Computation (CEC)*, $12–17^{th}$, Vol. 1, pp. 325–332, 2000.

[9] Z. Michalewicz and J. Arabas, Genetic Algorithms for the 0/1 knapsack problem, *Methologies for Intelligent Systems, (ISMIS'94)*, Lecture Notes in Computer Science, Vol. 869, pp. 134–143, 1994.

[10] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs*, Third, revised and extended edition, Springer, 1996.

[11] H. Mühlenbein, M. Gorges-schleuter and O. Krämer, Evolution Algorithms in Combinatorial Optimization, *Parallel Computing*, Volume 7, pp. 65–85, 1988.

[12] I. M. Oliver, D. J. Smith and J.R.C. Holland, A study of permutation crossover operators on the traveling salesman problem, *Genetic Algorithms and their Applications:Proceedings of the Second International Conference on Genetic Algorithms*, pp. 224–230, 1987.

[13] G. R. Raidl, an improved genetic algorithm for the multicontrained 0-1 knapsack problem, *Congress on Evolutionary Computation (CEC)*, pp. 207–211, May 1998.

[14] A. L. Olsen, Penalty functions and the knapsack problem, *Proceedings of the first IEEE Conference on Evolutionary Computation* Orlando, Florida, 1994 pp. 554–558.

[15] C.L. Valenzuela and A.J. Jones, Evolutionary divide and conquer (I): a novel genetic approach to the TSP, *Evolutionary Computation*, 1(4), pp. 313–333, 1994.

[16] C.L. Valenzuela, A study of permutation operators for minimum span frequency assignment using an order based representation, *Journal of Heuristics*, 7(1), pp. 5–22, 2001.

[17] C. L. Valenzuela, A simple evolutionary algorithm for multiobjective optimization (SEAMO), *Congress on Evolutionary Computation (CEC)*, Honolulu, Hawaii, $12–17^{th}$, Vol. 1, pp. 717–722, May 2002.

[18] C.L. Valenzuela and P.Y. Wang, VLSI placement and area optimization using a genetic algorithms to breed normalized postfix expressions, *IEEE Transactions on Evolutionary Computation*, 6(4), pp. 390–401, 2002.

[19] E. Zitzler and L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, *IEEE Transactions on Evolutionary Computation*, 3(4), pp. 257–271, 1999.

[20] E. Zitzler M. Laumanns and L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, TIK-Report 103, Department of Electrical Engineering, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, {zitzler, laumanns, thiele}@tik.ee.ethz.ch, 2001.