Comparing Algorithms, Representations and Operators for the Multi-Objective Knapsack Problem

Gualtiero Colombo
School of Computer Science
Cardiff University
United Kingdom
G.Colombo@cs.cardiff.ac.uk

Christine L. Mumford
School of Computer Science
Cardiff University
United Kingdom
C.L.Mumford@cs.cardiff.ac.uk

Abstract- This paper compares the performance of three evolutionary multi-objective algorithms on the multi-objective knapsack problem. The three algorithms are SPEA2 (strength Pareto evolutionary algorithm, version 2), MOGLS (multi objective genetic local search) and SEAMO2 (simple evolutionary algorithm for multi-objective optimization, version 2). For each algorithm, we try two representations: bit-string and order-based. Our results suggest that a bit-string representation works best for MOGLS, but that SPEA2 and SEAMO2 perform better with an order-based approach. Although MOGLS outperforms the other algorithms in terms of solution quality, SEAMO2 runs much faster than its competitors and produces results of a similar standard to SPEA2.

1 Introduction

The multi-objective knapsack problem (MKP) is a popular test-bed with researchers developing evolutionary multi-objective algorithms (EMOs). The present paper compares the performance of three EMOs on some large instances of this problem, trying two different representations on each EMO. The EMOs chosen for this study are SPEA2 (strength Pareto evolutionary algorithm, version 2) [17], MOGLS (multi-objective genetic local search) [5] and SEAMO2 (simple evolutionary algorithm for multi-objective optimization, version 2) [13]. The study builds on earlier work in which Mumford [12] used the SEAMO algorithm [14, 16] (the precursor of SEAMO2) as a framework to explore a number of different representations and operators for the 0-1 multi-objective knapsack problem. The approaches tested by Mumford on the SEAMO framework, covered bit-string and order-based representations, with various penalty functions, repair mechanisms and decoders, all adapted from the single objective case [3, 4, 10, 11]. Results published in [12] favor an order-based approach with a first fit decoder over the bit-string representations. Notwithstanding the apparent success of the order-based approach in this particular study, the favorite method of representation for the MKP is a bit-string chromosome with a greedy repair mechanism [5, 8, 17, 18]. However, the implications drawn from the comparative study must remain tentative, given the experiments were limited to the SEAMO algorithm. Bit-string representations may indeed produce better results for other EMOs.

The present study takes the most promising approaches established for SEAMO [12], and tries them on the three above mentioned EMOs. In addition, our paper assesses the relative merits of the EMOs in terms of solution quality and run time.

The remainder of the paper is organized as follows. Sections 2 to 5 cover the essential background to our study, including the MKP and an outline of the representations, operators and algorithms. In Section 6 we describe our experimental method, and in Section 7 we present our results. Finally, we summarize the paper and suggest future work in Section 8.

2 The 0-1 Multi-Objective Knapsack Problem

The 0-1 multi-objective knapsack problem (MKP) is a generalization of the 0-1 simple knapsack problem, and is a well known member of the NP-hard class of problems. In the simple knapsack problem, a set of objects, $O = \{o_1, o_2, o_3, ..., o_n\}$, and a knapsack of capacity C are given. 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 weight sum over the objects in S does not exceed the knapsack capacity and yields a maximum profit. 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 simultaneously, although neither the weight of an object nor its profit is fixed, and will probably have different values in each knapsack. The present study covers problems with between two and four objectives (i.e. knapsacks).

3 Representations for the Multiple Objective Problem

For our experiments we have selected the two best performing approaches from [12]: a bit-string representation with greedy repair [18], and an order-based representation with a first-fit decoder [12]. More details of these approaches are given below.

3.1 The Bit-String Representation with Greedy Repair

For the bit-string representation, all n objects of a given instance of the MKP are fixed in an arbitrary sequence, $[o_1, o_2, \ldots, o_n]$, and each object, o_i, is mapped to bit i of an n bit chromosome. Bit i is set if and only if the i^{th} object from the list is included in the knapsack, otherwise bit $i = 0$. Thus, for example, a bit pattern (0 1 1 0 1 0) for a

six object problem would mean that objects 2, 3 and 5 are packed.

Unfortunately, the act of randomly assigning bits to objects, cannot guarantee a feasible solution, and overfull knapsacks are produced when too many bits are set. Processing and storing large numbers of infeasible solutions can seriously reduce the effectiveness of an EA. The most popular method of dealing with this problem in the MKP is to use a greedy repair mechanism.

Initially, a greedy repair routine is presented with knapsacks packed with all the objects that have their bits set. Repair proceeds by the sequential removal of objects from the knapsacks, until all the capacity constraints are satisfied. In the repair method of Zitzler and Thiele [18], used in SPEA2, the order in which the items are removed is determined by the maximum profit/weight ratio per object (taken over all the knapsacks), with the object which is least profitable, per unit weight, being the first to be removed. The repair mechanism of [12] used in SEAMO2 is similar to this but here the removal of objects from the knapsacks is sequenced on *average* profit to weight ratio taken over all the knapsacks. Our implementations of SPEA2 and SEAMO2 do not write back the repaired chromosomes into the population.

The repair mechanism for MOGLS, although similar to the approach described for SPEA2 and SEAMO2, takes account of the *weighted scalarized functions* that MOGLS uses to fuel the *local search* phase of the algorithm, [5]. In this case the replacement of repaired chromosomes into the population forms a constituent part of the algorithm.

One-point crossover and point mutation (i.e. bit flips) are used for all our bit-string experiments.

3.2 The Order-Based Representation

In the order-based representation the genes represent the objects themselves and the chromosomes consist of orderings of all the objects. Because every object is included on each chromosome, a decoder is required to produce legal solutions. Hinterding [3] used a first fit heuristic for his order-based representation in the single objective case. Starting with an empty knapsack, he selected objects in sequence from a permutation list, starting with the first object on the list, then working through to the second, then the third and so on. Whenever inclusion of an object from the list would result in a constraint violation, that object was skipped over and the next object tried. Assume we are given an order-based sequence of (1, 5, 2, 4, 3, 6) for a six object problem in the single objective case. A first-fit decoder will attempt to pack object 1, then object 5, then object 2 etc., until it reaches an object that, if packed, would exceed the capacity constraint. If this occurs, for example, when object 4 is tried, the first-fit decoder would skip over object 4 and try object 3, then 6.

Adapting a decoder based on the first fit algorithm for the multi-objective knapsack problem simply requires that the constraints are tested for all the knapsacks each time an object is considered for inclusion in the solution, and this is the approach adopted here.

Cycle crossover, CX, [15] is used as the recombination operator for all 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 described in [12].

4 The Evolutionary Multi-Objective Algorithms

We justify our choice of SPEA2, MOGLS and SEAMO2 as follows:

1. SPEA and SPEA2 are successful and widely implemented,
2. MOGLS produces particularly impressive results and uses a different approach to most other EMOs: linear scalarizing functions, rather than Pareto-based fitness assignment
3. SEAMO and SEAMO2 seem to be fast and effective, but need testing against other state-of-the-art EMOs.

Below we outline the main features of our chosen EMOs. Interested readers should refer to the original papers for more details, [5, 13, 14, 16, 17, 18].

4.1 SPEA2

Procedure *SPEA2*
Parameters: N – *population size*, $\bar{N}$ – *archive size*, *stopping condition*
Begin
Initialization:
Generate an initial population, of N random individuals
Create an empty archive, $\bar{P}_0 \leftarrow \emptyset$
Main loop
Repeat
Fitness assignment:
for each individual in P_t and $\bar{P}_t$
Environmental selection:
from P_t and $\bar{P}_t$ creating new archive, $\bar{P}_{t+1}$
Mating selection:
Perform binary tournament selection
with replacement on $\bar{P}_{t+1}$
in order to fill the mating pool, M_{t+1}
Apply recombination and mutation to M_{t+1}
Set $P_{t+1} \leftarrow M_{t+1}$
Increment generation counter, $t \leftarrow t + 1$
Until stopping condition is satisfied
Print all non-dominated solutions in the final population and archive
End

Figure 1: Algorithm 1 A basic framework for SPEA2

SPEA2 uses two populations: a regular population, P_t and an archive, $\bar{P}_t$. The archive is used to accumulate non-dominated solutions and it is from here that individuals are selected to form a mating pool. Following the application of the genetic operators to the mating pool, the resulting offspring become the new regular population, P_{t+1}. The new archive, $\bar{P}_{t+1}$, is mainly constructed from the non-dominated solutions in $P_t + \bar{P}_t$. Binary tournament selection is used to fill the mating pool, and fitness assignment consists of two components: one based on the concept of Pareto

dominance and the other on density information. The purpose of the density component is to encourage a more even spread of solutions across the Pareto front. SPEA2 is outlined in Figure 1.

In more detail, the procedure for fitness assignment can be expressed as follows:

$$F(i) = R(i) + D(i)$$

Where $F(i)$ is the total fitness value, $R(i)$ represents the raw fitness component, and $D(i)$ the density component.

In order to calculate the raw fitness, each individual in P_t and $\bar{P}_t$ is assigned a *strength* value, $S(i)$, representing the number of solutions it dominates. The raw fitness, $R(i)$, for a particular individual, is then determined by adding together the strengths of all members of P_t and $\bar{P}_t$, that dominate that individual. The density value, $D(i)$, is based on the inverse of the Euclidean distance (in objective space) of the k^{th} nearest neighbor from individual i:

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

where σ_i^k denotes the Euclidean distance sought. We set $k = \sqrt{\mathtt{N} + \bar{\mathtt{N}}}$, following advice in [17]. In the denominator, two is added to avoid division by zero and ensure $D(i) < 1$.

Environmental selection essentially copies all the individuals in P_t and $\bar{P}_t$ that are non-dominated, to $\bar{P}_{t+1}$, but at the same time, it ensures that the size of the archive remains constant at $\bar{N}$. To achieve this, solutions are deleted if $\bar{N}$ is exceeded and added when the archive size would otherwise fall below $\bar{N}$. A truncation operator is used to reduce the population by deleting those members considered too close to another individual in objective space. When the population needs to be increased, the best dominated individuals are copied from the previous archive and population.

4.2 MOGLS

In [5] Jaskiewicz describes how MOGLS is able search for approximations to all the points in a non-dominated set simultaneously, by making a random choice of a scalarizing function at each iteration of the algorithm's main loop. This process simply requires the application of random numbers, $rand()$, in the range $(0, 1)$ to the formulae in (1):

$$\begin{aligned}
\lambda_1 &= 1 - \sqrt[m-1]{rand()} \\
&\dots \\
\lambda_j &= \left(1 - \sum_{l=1}^{j-1} \lambda_l\right)\left(1 - \sqrt[m-1-j]{rand()}\right) \\
&\dots \\
\lambda_m &= 1 - \sum_{l=1}^{m-1} \lambda_l
\end{aligned} \tag{1}$$

When $m = 2$ objectives, the weights applied to objectives 1 and 2 simply become, respectively, $\lambda_1 = 1 - r_1$ and $\lambda_2 = r_1$, where $r_1 = rand()$. When $m > 2$ it is necessary to generate more than one random number, $(r_1, r_2, \dots r_{m-1})$, using the function $rand()$. The linear scalarizing function in the is given by: $\lambda_1 P_1 + \lambda_2 P_2 + \dots + \lambda_m P_m$, where P_1, P_2 etc. are the total profits in knapsack 1, 2 and so on. (Note: For problems where objectives share approximately the same range, it is not necessary to normalize the scalarizing function, [5].) Outline pseudocode for MOGLS can be found in Figure 2.

Procedure *MOGLS*
Parameters: S – *number of initial solutions*
K – *size of the temporary elite population* (TEP),
stopping condition
Begin
Initialization:
The set of potentially non-dominated points $PP \leftarrow \emptyset$
The current set of solutions $CS \leftarrow \emptyset$
Generation of the initial set of solutions:
Repeat S **times**
Draw at random a weight vector, $(\lambda_1, \lambda_2, \dots, \lambda_m)$
Construct randomly a new feasible solution x
Apply local heuristic to solution x obtaining x'
Update set PP with x'
Main loop
Repeat
Draw at random a weight vector, $(\lambda_1, \lambda_2, \dots, \lambda_m)$
From CS select K solutions best on
scalarizing functions $(\lambda_1 P_1 + \lambda_2 P_2 + \dots + \lambda_m)$
forming temporary elite population TEP
Draw at random with uniform probability
two solutions x_1 and x_2 from TEP
Recombine x_1 and x_2 obtaining x_3
Mutate x_3 then apply local heuristic to solution x_3
obtaining x_3'
If x_3' is better on $(\lambda_1 P_1 + \lambda_2 P_2 + \dots + \lambda_m)$
than the worst solution in TEP
and different in the decision space
from all solutions in TEP
Then
Add x_3' to the current set of solutions CS
and update set PP with x_3'
Until stopping condition satisfied
Print PP
End

Figure 2: Algorithm 2 A basic framework for MOGLS

In addition to the current set, CS, MOGLS maintains two further populations: PP, the set of potentially non-dominated solutions, and TEP a temporary elite population. At the start of each iteration of the main loop, a new random weight vector, $(\lambda_1, \lambda_2, \dots, \lambda_m)$, is generated and this is used to select K different solutions from CS to make up TEP. The K solutions selected are the best on the current scalarizing function, $(\lambda_1 P_1 + \lambda_2 P_2 + \dots + \lambda_m P_m)$. Two solutions, x_1 and x_2, are next selected at random from TEP. These are recombined to produce an offspring, x_3. Mutation is then applied, followed by the greedy repair heuristic in the case of the bit-string representation, or the decoder applied to the order-based scheme, giving x_3'. x_3' is then added to CS if it is better on the linear scalarizing function than the worst solution in TEP and different in the de-

cision space (i.e., the chromosomes) from all the solutions in TEP. Finally PP, the set of potentially non-dominated solutions, is updated.

As already mentioned in Section 3.1, the local search heuristic used for the bit-string representation involves a form of greedy repair heuristic that takes into account the current value of the linear scalarizing function. In more detail, each new offspring, x, is checked for feasibility, and if the capacity is exceeded for one or more of the knapsacks, objects are removed (i.e. bits reset from "1" to "0") until the constraints are satisfied. The order in which the objects are removed corresponds with their profit to weight ratios amalgamated according to the random vector, $(\lambda_1, \lambda_2, \ldots, \lambda_m)$. Assuming that object i has profit to weight ratios, $p_{i1}/w_{i1}, p_{i2}/w_{i2} \ldots, p_{im}/w_{im}$, in knapsacks $1, 2, \ldots, m$ respectively, the amalgamated profit to weight ratio for object i will be $\lambda_1 p_{i1}/w_{i1} + \lambda_2 p_{i2}/w_{i2} + \ldots + \lambda_m p_{im}/w_{im}$. The local search heuristic will remove objects in non-decreasing sequence of these amalgamated values (thus removing the least promising objects first). In the case of the order-based representation, the decoder described in Section 3.2 will produce a feasible solution from any given permutation, making the local repair heuristic redundant. As mentioned previously, the lack of a weighted local search in MOGLS when using the order based representation, is a potential disadvantage (and we shall see later that this is borne out by our results).

CS is organized as a queue of maximum size $K \times S$, where K is the size of the temporary elite populations and S is the size of initial population. New solutions are added to the front of the queue, and old solutions deleted from the back of the queue to prevent CS exceeding its maximum size.

4.3 SEAMO2

The SEAMO2 framework, outlined in Figure 3, illustrates a simple steady-state approach, which sequentially selects every individual in the population to serve as the first parent once, and pairs it with a second parent that is selected at random (uniformly).

A single crossover is applied to produce each offspring, and this is followed by a single mutation. Each new offspring will either replace a parent, or another population member, or it will die, depending on the outcome of a number of tests. Essentially, an offspring will replace a parent if it is deemed to be better than that parent. This occurs if the offspring dominates one of the parents, or if produces a new global best profit in one of the knapsacks. If the offspring neither dominates either of its parents, nor is it dominated by either of them, the offspring will replace a random individual in the population, if one can be found that it dominates. Any offspring that duplicates any member of the population in its objective space will be deleted, regardless of its ability to dominate its parents.

Procedure *SEAMO2*
Parameters: *N – population size*, *stopping condition*
Begin
Generation of the initial set of solutions:
  Generate N random individuals
  Evaluate the objective vector for each individual
    and store it
  Record the bestSoFar for each objective function
Main loop
  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 offspring
      Evaluate the objective vector
        produced by the offspring
      If offspring objective's vector
        improves on any bestSoFar
        Then it replaces a parent
      Else if offspring is a duplicate
        Then it dies
      Else if offspring dominates either parent
        Then it replaces it
      Else if offspring is neither dominated by
        nor dominates either parent
        Then it replaces another individual
        that it dominates at random
      Otherwise it dies
    End for
  Until stopping condition is satisfied
  Print all non-dominated solutions in the final population
End

Figure 3: Algorithm 3 A basic framework for SEAMO2

5 Comparing the Performance of the Algorithms

Unfortunately there is no general agreement amongst researchers on how best to assess the quality of a non-dominated set of points produced by a particular multi-objective algorithm. Of the multitude of metrics from the literature which attempt to express the solution quality as a single number, the $\mathcal{S}$ metric of Zitzler and Thiele[18] has been recommended in [9] as a good all-round choice, provided the problem involves relatively few dimensions, and the non-dominated sets are not overlarge. For this reason we have adopted $\mathcal{S}$ as the main metric for the the present study. In addition, we use simple 2D plots, where appropriate, to give a visual interpretation and also consider the cardinality (i.e. size) of the non-dominated sets produced by the various algorithms and compare their run times.

5.1 The $\mathcal{S}$ Metric

The $\mathcal{S}$ metric is a measure of the size of the dominated space, measured in the number of dimensions determined by the number of objectives. 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 hyperplanes arising out of x_i, along with the axes. For each axis in the objective space, there exists a hyperplane perpendicular to that 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 $(f_1(x_i), f_2(x_i))$. In this study, the size of the dominated space is quoted as a percentage of the reference volume between the origin and an upper approximation of the ideal point taken from [5].

6 Experimental Method

The purpose of our experiments is to answer the following questions:

1. For each EMO, which representation is best: bit-string or order-based?
2. How do the EMOs compare with each other with respect to solution quality?
3. How do the EMOs compare with each other with respect to run time?

Five problems were selected from the test data taken from [18], kn250.2, kn500.2, kn750.2, kn750.3 and kn750.4. Problems kn250.2, kn500.2 and kn750.2 are two objective problems with 250, 500 and 750 objects, respectively, while kn750.3 and kn750.4 are problems with 750 objects and 3 and 4 objectives. SPEA2, MOGLS and SEAMO2 were each run 30 times, on the two objective problems, and just 5 times on kn750.3 and kn750.4 (because of the long run times). Each replicate run was seeded with a different random start, and population sizes were N = 150, 200 and 250 for kn250.2, kn500.2 and kn750.(2,3,4). On all problems, runs were extended for 5000 generations for SPEA2 and SEAMO2, and for $5000 \times S$ evaluations in the case of MOGLS. Each EMO performed exactly the same number of objective function evaluations on a given problem. For SPEA2 the regular population (P_t) and archive ($\bar{P}_t$) were set to the same size, $N = \bar{N}$. MOGLS was initialized with a current set, CS, of $S = N$, and we set the size of TEP to 20, the same as [5].

Crossover was applied at a rate of 100 % in MOGLS and SEAMO2 and at a rate of 80 % in SPEA2, and the mutation rate was 0.6 % in SPEA2, 1 % in MOGLS and one mutation per offspring in SEAMO2. Guidelines for crossover and mutation rates were based on those given in the original papers, validated by us in some pilot studies. To give each algorithm a fair chance in the contest, we set crossover and mutation rates favorable to each technique, rather than use a "one size fits all" approach.

In adapting MOGLS for an order-based implementation, we found it necessary to deviate slightly from the published version of the algorithm. The standard version of MOGLS checks the temporary elite population, TEP, for duplicates in the decision space. However, the order-based encoding has a very high level of redundancy (i.e. many orderings produce identical solutions) and deleting duplicates in the decision space tends to be ineffective and leaves too many individuals that are identical in the objective space. For this reason we delete duplicates in the objective space of CS, rather than the decision space of TEP. Much better results can be obtained in this way, with no extra burden on

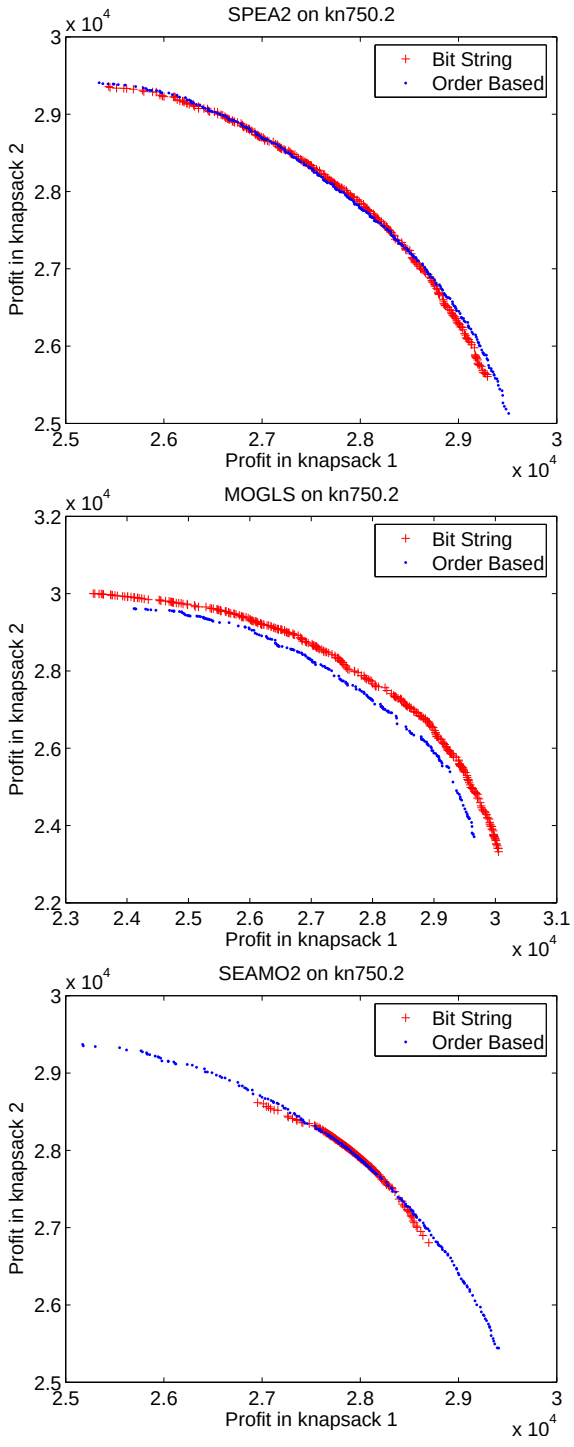

Figure 4: Non dominated solutions extracted from 5 runs of SPEA2 (top), MOGLS (center), and SEAMO2 (top)

the run time (pairwise comparisons in the decision space are very much more time consuming than comparisons in the objective space). We also discovered that better results for SPEA2 can be obtained if objective space duplicates are deleted, and not allowed to enter the archive. Furthermore, these improvements were noted for SPEA2 using bit-string as well order-based approaches. These findings are in accordance with previous reports on the same issue. Fitness duplicate elimination has been shown to be an important feature to avoid drift and improve performance in multi-objective evolutionary algorithms on a range of problems in [1].

The performances of the algorithms are assessed using the $\mathcal{S}$ metric described in [18] and outlined in Section 5 of the present paper. In addition, 2D graphical plots, run time measurements and the sizes of non-dominated sets provide additional information. Experiments were conducted using Java 2, version 1.4.2 (J2SE) on a PC with 2.99 GHz Intel Pentium 4 and 1 GB RAM.

7 Results

We shall now attempt to answer the three questions posed in the previous section, beginning with the effect of representation. A visual impression of the algorithms' performance on kn750.2 can be obtained from Figure 4, which plots non-dominated solutions extracted from five replicate runs, comparing bit-string and order-based representations for each EMO. The traces in Figure 4 indicate that MOGLS performs better with a bit-string representation but that the order-based representation works best for SEAMO2. The results appear less clear cut for SPEA2, although the order-based approach seems to give a better spread of results.

Figure 5 shows box plots for the $\mathcal{S}$ metric on kn250.2, kn500.2 and kn750.2. Each plot gives the distribution of the dominated space for 30 replicate runs, covering a particular EMO, representation and knapsack instance. The plots provide strong supporting evidence in favor of bit-strings for MOGLS and order-based representations for both SEAMO2 and SPEA2. The box plots in Figure 6 for kn750.3 and kn750.4 add further support in favor of a bit-string approach for MOGLS, and furthermore, the gap between the two approaches appears to grow quite considerably for MOGLS, as the number of objectives increases from two to four. This is consistent with expectations, considering the absence of the local search phase in the order-based approach. As the number of objectives increase, so the problems get harder to solve, and it is possible that the role of local search becomes increasingly important. The order-based approach performs consistently better than the bit-string for SEAMO2. However, the case is not quite so clear cut for SPEA2, which still favors an order-based approach for kn750.3 but performs slightly better with a bit-string representation on kn750.4.

We shall now try to answer the second question we posed, and compare the three EMOs with respect to solution quality. Examination of Figures 5 and 6, indicate that MOGLS is the best performer, using a bit-string representation. Figures 5 and 6 further suggest that SPEA2 is marginally better than SEAMO2 on problems with two objectives, but that the situation changes when more objectives are used, with SEAMO2 outperforming SPEA2 on kn750.3 and kn750.4. Figure 7 compares the three EMOs: SPEA2, MOGLS and SEAMO2 on kn750.2, using the most appropriate representation in each case. (The results are similar for kn250.2 and kn500.2 but the sets of points are closer together and rather difficult to distinguish.) As before, non dominated solutions are extracted from 5 replicate runs. Clearly, MOGLS is able to achieve a much broader spread of results than the other EMOs. MOGLS also tends to produce larger sets of non-dominated points than the other EMOs, especially for problems with more objectives. Table 1 bears this out.

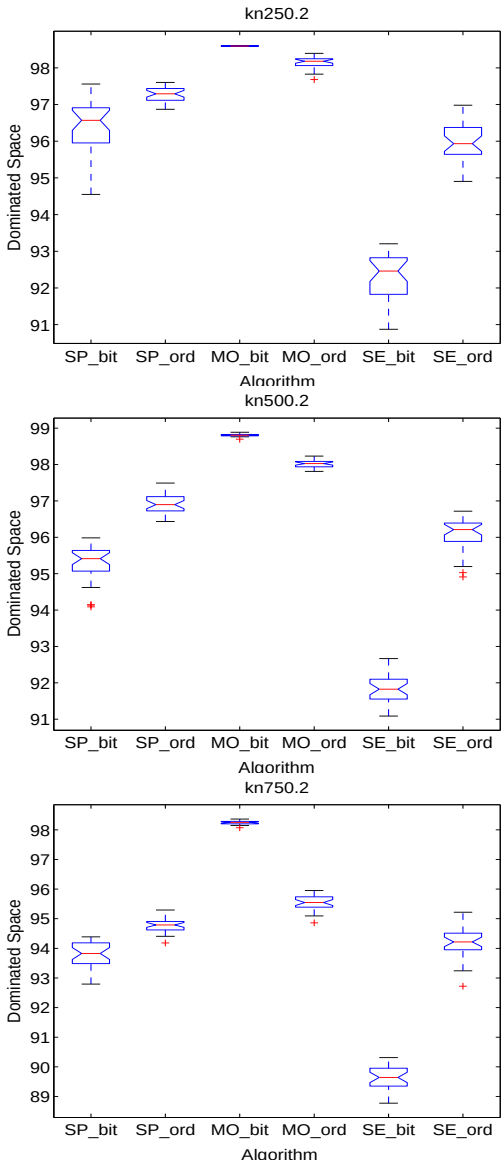

Figure 5: Dominated space for 30 runs of 5,000 generations on kn250.2 (top), kn500.2 (middle) and kn750.2 (bottom), SP: SPEA2, MO: MOGLS SE: SEAMO2

Finally, we answer question 3 and compare the run times of the three algorithms. Table 2 shows the results, in seconds, for the three EMOs on kn250.2, kn500.2 and kn750.2,

Table 1: Average sizes of final non-dominated sets, $|z|$

Algorithm	Test problems				
	kn250.2	kn500.2	kn750.2	kn750.3	kn750.4
SPEA2 BIT	155	201	242	264	301
SPEA2 ORD	164	209	245	270	300
MOGLS BIT	212	212	190	1492	3087
MOGLS ORD	190	175	145	850	1504
SEAMO2 BIT	68	80	107	204	228
SEAMO2 ORD	97	136	172	224	239

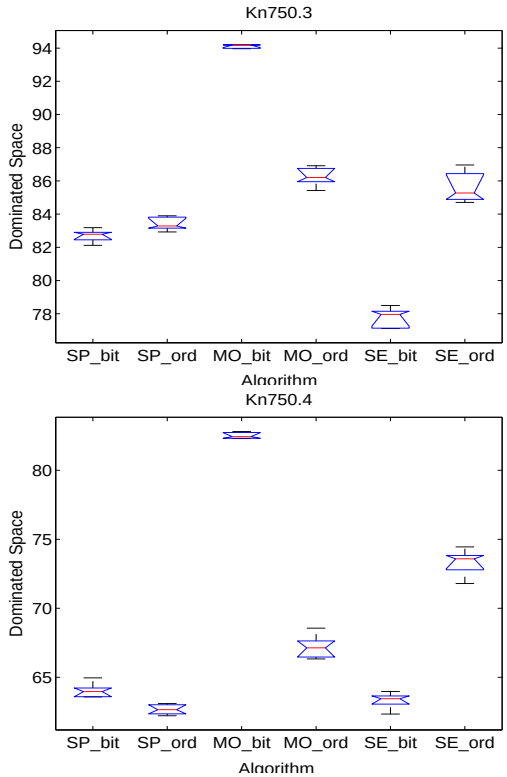

Figure 6: Dominated space for 5 runs of 5,000 generations on kn250.3 (top) and kn750.4 (bottom), SP: SPEA2, MO: MOGLS SE: SEAMO2

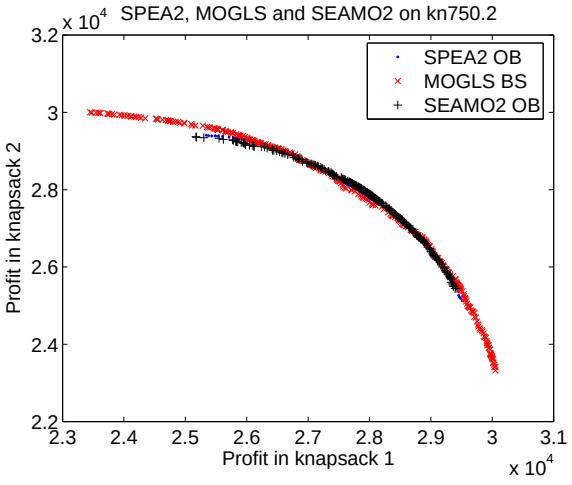

Figure 7: Comparing the performance of MOEs: OB = order-based, BS = bit-string

using the same parameters as previously, with each EMO performing exactly the same number of objective function evaluations on a given problem. Averages are taken of three replicate runs, starting with different random seeds, and excluding the calculation of any metrics. Clearly, SEAMO2 is at least an order of magnitude faster than its competitors on the two objective problems. Unfortunately, we did not record run times on kn750.3 and kn750.4 because the work was carried out at a much later date and on a different platform, making comparisons unreliable.

Table 2: Comparing run times of EMOs (in seconds)

Algorithm	kn250.2	kn500.2	kn750.2
SPEA2 BIT	1169	2180	3341
SPEA2 ORD	1205	2287	3593
MOGLS BIT	1078	3400	7890
MOGLS ORD	497	1100	1710
SEAMO2 BIT	23	50	98
SEAMO2 ORD	34	78	138

8 Conclusion and Discussion

This paper compares the performance of the three EMOs: SPEA2, MOGLS and SEAMO2, using the multi-objective knapsack problem as a test-bed. For each algorithm, we try two representation schemes: bit-string and order-based. Our results provide strong evidence that a bit-string representation works best for MOGLS, but that SEAMO2 performs better if an order-based approach is used. The relative performance of the two approaches is less well defined for SPEA2, however. MOGLS appears to benefit from the bit-string representation because it is able to apply its weighted scalarizing function at the repair stage, thus biasing the search towards the chosen region of the Pareto front. Furthermore, it would seem that the relative benefit of applying the bit-string approach to MOGLS increases as the problems get harder with more objectives.

Over all, MOGLS produces better results than either SPEA2 or SEAMO2. SPEA2 and SEAMO2 produce rather similar results, with SPEA2 performing slightly better than SEAMO2 on the instances with two objective problems, and SEAMO2 doing better on kn750.3 and kn750.4. The EMOs are compared on the basis of equal numbers of objective function evaluations. However, as MOGLS and SPEA2 both take very much longer to run than SEAMO2 on the knapsack test problems, it is clear that an evaluation for evaluation comparison may not be entirely fair.

Future plans include extending our comparative studies (covering representations, solution quality and run time) to more EMOs, for example PESA (Pareto envelope-based selection algorithm) [7], NSGA II (non-dominated sorting algorithm II) [2], and PMA (Pareto memetic algorithm) [6]. PMA is of particular interest because it operates in a similar fashion to MOGLS (using scalarizing functions), and thus promises high quality results. Yet, because it uses a simple tournament selection instead of TEP used by MOGLS, it promises to be much faster.

Bibliography

[1] H. Aguirre and K. Tanaka. Selection, drift, recombination, and mutation in multiobjective algorithms on scalable mnk-landscapes. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, number 3410 in LNCS, pages 355–369. Springer, 2005.

[2] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature – PPSN VI*, number 1917 in LNCS, pages 849–858, Berlin, 2000. Springer.

[3] R. Hinterding. Mapping, order-independent genes and the knapsack problem. In *Proceedings of the first IEEE Congress on Evolutionary Computation*, pages 13–17, Orlando florida, 1994.

[4] R. Hinterding. Representation, constraint satisfaction and the knapsack problem. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC99)*, volume 2, pages 1286–1292, 1999.

[5] 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):402–412, 2002.

[6] A. Jaskiewicz. On the computational efficiency of multiple objective metaheuristics. the knapsack problem case study. *European Journal of Operational Research*, 158:418–433, 2004.

[7] J. Knowles and M. Oates. The pareto envelope-based selection algorithm for multiobjective optimisation. In *Parallel Problem Solving from Nature – PPSN VI*, number 1917 in LNCS, pages 839–848, Berlin, 2000. Springer.

[8] J. D. Knowles and D. W. Corne. M-paes: a memetic algorithm for multiobjective optimization. In *Proceedings of the 2000 IEEE Congress on Evolutionary Computation (CEC2000)*, volume 1, pages 325–332, 2000.

[9] J. D. Knowles and D. W. Corne. On metrics for comparing non-dominated sets. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC2002)*, pages 711–716, Honolulu, Hawaii, 2002.

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

[11] Z. Michalewicz and J. Arabas. Genetic algorithms for the 0/1 knapsack problem. In *Methologies for Intelligent Systems, (ISMIS'94)*, number 869 in LNCS, pages 134–143. Springer, 1994.

[12] C. L. Mumford. Comparing representations and recombination operators for the multi-objective 0/1 knapsack problem. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC2003)*, pages 854–861, Canberra, Australia, 2003.

[13] C. L. Mumford. Simple population replacement strategies for a steady-state multi-objective evolutionary algorithm. In *Proceedings of the 2004 Genetic an Evolutionary Computation Conference (GECCO)*, pages 1389–1400, Seattle, Washington, USA, 2004.

[14] C. L. Mumford-Valenzuela. A simple approach to evolutionary multiobjective optimization. In A. Abraham, L. Jain, and R. Goldberg, editors, *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, chapter 4, pages 55–79. Springer, London, 2005.

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

[16] C. L. Valenzuela. A simple evolutionary algorithm for multi-objective optimization (seamo). In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC2002)*, pages 717–722, Honolulu, Hawaii, 2002.

[17] E. Zitzler, M. Laumanns, and L. Thiele. Spea2: Improving the strength Pareto evolutionary algorithm. Technical Report 103, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

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