European Journal of Operational Research 206 (2010) 417–425

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Decision Support

Interactive evolutionary multi-objective optimization for quasi-concave preference functions

John W. Fowler [a], Esma S. Gel [a], Murat M. Köksalan [b], Pekka Korhonen [c], Jon L. Marquis [a], Jyrki Wallenius [c,*]

[a] Arizona State University, P.O. Box 875906, Tempe, AZ 85287-5906, USA
[b] Middle East Technical University, 06531 Ankara, Turkey
[c] Aalto University School of Economics, P.O. Box 21210, 00076 AALTO, Finland

ARTICLE INFO

Article history:
Received 7 August 2009
Accepted 19 February 2010
Available online 4 March 2010

Keywords:
Interactive optimization
Multi-objective optimization
Evolutionary optimization
Knapsack problem

ABSTRACT

We present a new hybrid approach to interactive evolutionary multi-objective optimization that uses a partial preference order to act as the fitness function in a customized genetic algorithm. We periodically send solutions to the decision maker (DM) for her evaluation and use the resulting preference information to form preference cones consisting of inferior solutions. The cones allow us to implicitly rank solutions that the DM has not considered. This technique avoids assuming an exact form for the preference function, but does assume that the preference function is quasi-concave. This paper describes the genetic algorithm and demonstrates its performance on the multi-objective knapsack problem.

1. Introduction

The field of combinatorial optimization, with its variety of NP-hard problems, has turned to heuristics to provide nearly optimal solutions to previously intractable problems. Frequently, however, a DM is faced with a combinatorial optimization problem with several different objectives. These multi-objective combinatorial optimization (MOCO) problems are even more difficult to solve optimally since they involve both NP-hard problems and multiple objectives.

While it is sometimes possible to create a single objective problem by combining the objective functions into a single preference function, the way a DM combines conflicting objectives is often difficult to capture. In many cases, the DM cannot quantify how the objectives should be combined into a preference function that can then be optimized using single objective optimization techniques. This complication is often severe enough to make *a priori* approaches impractical. For a more complete discussion of the advantages and shortcomings of these approaches, the interested reader is referred to Dyer et al. (1992) and Wallenius et al. (2008).

An alternative approach is to generate the set of Pareto optimal solutions (or approximate Pareto optimal solutions), and present them to the DM. The DM then selects a solution *a posteriori*. Unfortunately, this set is difficult to visualize for problems having three or more objectives, and is very time consuming to generate since it may be comprised of thousands of solutions, even in the case of linear constraints and objectives (Kondakci et al., 1996).

Interactive decision making addresses many of the challenges associated with *a priori* and *a posteriori* approaches. It combines the process of obtaining information from the DM with the process of generating solutions to the problem. Seminal papers in the area include Geoffrion et al. (1972) and Zionts and Wallenius (1976). Since these approaches generate solutions that the DM must evaluate in order to guide the progression of the algorithm, research in this area is concerned with both the generation and the representation of solutions. There are software packages that perform both activities. See Caballero et al. (2002) and Poles et al. (2006) for a comprehensive list of software descriptions. For more information on generating solutions and representing them to a DM, the interested reader is referred to Miettinen (1999).

Evolutionary optimization and computing has emerged as a new field with strong ties to Multiple Criteria Decision Making/Multiattribute Utility Theory (Deb, 2001). The first evolutionary multi-objective optimization algorithm is due to Schaffer (1984). However, it was not until about 10 years later that three working evolutionary algorithms were suggested almost at the same time: MOGA by Fonseca and Fleming (1993), NSGA by Srinivas and Deb (1994), and NPGA by Horn et al. (1995). The main thrust in all these algorithms was to generate an approximation of the Pareto optimal frontier. An interesting paper is Fonseca and Fleming (1998), who demonstrate the need for preference articulation in cases where many objectives lead to a non-dominated set too large to effectively sample. A survey of methods developed before 2000

* Corresponding author.
E-mail address: Jyrki.Wallenius@hse.fi (J. Wallenius).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.02.027

attempting to handle user's preferences is provided by Coello Coello (2000). More recent attempts to incorporate user's preferences into a multi-objective evolutionary framework include Cvetkovic and Parmee (2002), Phelps and Köksalan (2003), Branke and Deb (2004), Deb et al. (2005), Hanne (2005), Kamalian et al. (2004), Molina et al. (2009), and Parmee et al. (2001). Cvetkovic and Parmee (2002) assign weights for the objectives and additionally require a minimum level for dominance. NSGA-II (Deb et al., 2002) elaborates on two methods; the guided dominance principle, and biased crowding distance to incorporate vague user preferences. The method in Deb et al. (2005) is based on the idea of using reference direction projections as part of the fitness function. Hanne (2005) discusses interactive decision support based on evolutionary principles. Parmee et al. (2001) and Kamalian et al. (2004) discuss interactive evolutionary systems for multi-objective design. Molina et al. (2009) develop a reference point-based method where they favor solutions that dominate a reference point or that are dominated by this reference point over all other solutions. Phelps and Köksalan (2003) develop and demonstrate an interactive genetic algorithm on multi-objective knapsack and minimum spanning tree problems. Finally, Köksalan and Phelps (2007) developed a multi-objective evolutionary algorithm to concentrate on a desired part of the efficient frontier using partial information on the preferences of the DM.

To the best of our knowledge, Phelps and Köksalan (2003)'s and our approach are the only approaches that guarantee correct partial orders of the populations provided that the DM's preferences are consistent with the assumed utility function forms. Phelps and Köksalan (2003) use a linear utility function and make corrections to the partial order whenever the DM's expressed preferences are not consistent with such a function. We assume a more general quasi-concave function that is considered to represent human preferences well. To our knowledge, this is the only evolutionary algorithm that incorporates the properties of an implicit quasi-concave utility function into the algorithm. Utilizing the theory developed for quasi-concave functions we guarantee to produce partial orders that are consistent with preferences derived from such functions.

Our research uses a similar interactive genetic algorithm framework to that of Phelps and Köksalan (2003), and we use a similar experimental framework to test our algorithm. They estimate and use a linear utility function to order the population. They make corrections to this ordering when the DM's expressed preferences are in conflict with this order. However, our technique is more general since it is designed to support any quasi-concave function of the objectives. Incorporating DM preferences in the metaheuristic via DM interaction guides the search to the most preferred region of the solution space. This helps avoid both the computational burden of generating the full efficient frontier and the problems inherent in attempting to quantify the DM's preference function. More specifically, we propose an evolutionary metaheuristic that evaluates (i.e., partially rank orders) solutions using the convex preference cones developed in Korhonen et al. (1984). These cones are generated based on pairwise comparisons of solutions, making it possible to preference order solutions that the DM has never directly evaluated and provide a stronger ordering than previous dominance-based techniques. The preference order is used to evaluate the fitness of the population members. As in Phelps and Köksalan (2003), we assume that the individual objectives are known, and that the user's preference function is unknown.

The results of our method are evaluated in a manner similar to that of Phelps and Köksalan (2003). We compare our results to the best solution found by the genetic algorithm per the DM's true preferences to assess the ability of our algorithm to correctly select the DM's most preferred solution. Due to the difficulty of accurately solving large problems to optimality, we present a comparison of our result to the optimal linear programming (LP) relaxation of our problem instances. We report the average value of these metrics over several runs of each tested configuration of the algorithm.

In the following sections, we detail the development of the evolutionary metaheuristic and present computational results for the multi-objective 0–1 knapsack problem (MOKP). Without loss of generality, we assume that each objective is to be maximized.

2. Steps of the evolutionary metaheuristic

While all genetic algorithms share certain common features (Michalewicz, 1996), they must be adapted to the specific problem at hand in order to provide good solutions efficiently (Phelps and Köksalan, 2003). In this case, the algorithm must generate the population of solutions, obtain preference information from the DM, and partially order the population members, based on the preference information in order to find which members of the population to select for breeding and which to replace. The standard functions of our genetic algorithm, such as the random number generator, are based on GAlib (Wall, 1995).

The first problem to consider in a genetic algorithm is how to represent the solution. For the multi-objective knapsack, the simplest approach is to create a boolean vector of length J, where J is the number of candidate items for inclusion in the knapsack. Since the items can be indexed from 1 to J, the jth entry in the vector will be set to one if the jth item is to be included. It remains zero otherwise.

A high level overview of the steps of the evolutionary metaheuristic is given below.

1. Create the population
2. Send a sample, S', of the population to the DM
3. Preference order the population using feasibility, Pareto dominance, and preference cone dominance (most preferred to least preferred)
4. Replace the last (worst) $r\%$ of the population
5. Select parents
   (a) Probabilistically select the first parent based on preference
   (b) Randomly select the second parent
6. Breed the parents and mutate the offspring
   (a) Breed using a crossover operator
   (b) With probability p_m, mutate the offspring
7. Update the population by adding the new population members
   (a) $\frac{2}{3}r\%$ are bred using the crossover operator
   (b) The rest are generated randomly
8. Update the best solution and repeat for the required number of generations

The following subsections describe each step in detail.

2.1. Create the population

Creating the initial population is known to be important to how quickly a genetic algorithm reaches a good solution (Phelps and Köksalan, 2003). Genetic algorithms provide better solutions when the initial solutions are Pareto optimal, or nearly so, and have a reasonable amount of diversity in the population. For this reason, the initial population is composed of two groups: a "random" group and an "efficient" group. The random group consists of solutions whose chromosomes are generated randomly with an equal probability of all possible values. This group composes 1/3 of the

initial population, and the solutions are not constrained to be feasible. Instead, infeasible solutions will be penalized, and thus tend to be replaced as the algorithm progresses. The primary purpose of this group is to provide genetic diversity. The efficient group is based on assigning weights to the objectives to produce a linear combination of the objectives. This linear combination acts as a single objective, so a greedy heuristic is then applied to the problem to produce a good solution. The weights are randomly generated and normalized to sum to unity. This process fills the remaining 2/3 of the population with solutions near the efficient frontier of the multi-objective problem with a linear objective function, and constitutes a means of providing a good initial population. This generation technique is more favorable to linear preference functions since it uses a linear combination of the objectives, but it has also shown to be effective for Chebyshev preference functions. This suggests that providing nearly efficient solutions is a good starting point for a variety of preference functions.

2.2. Send a sample of the population to the DM

Once an initial population has been created, the DM is asked to evaluate a set S' of $|S'|$ members of the population, and to specify the best and worst in the set. (Due to human cognition limits explained in Miller (1956), $|S'|$ is restricted to be strictly less than eight.) The best and worst solutions in the set (called sample) are used to create convex preference cones in the objective space of the problem, which are used to define inferior solutions. The precise method of using the cones is explained in the next section. Theorem 1 of Korhonen et al. (1984) explains the role of the cones in ranking solutions. It states, "Assume a quasi-concave and nondecreasing function $f(x)$ defined in a p-dimensional Euclidean space R^p. Consider distinct points $x_i \in R^p,\ i = 1,\ldots,M$ and any point $x^* \in R^p$ and assume that $f(x_k) < f(x_i),\ i \neq k$. Then, if $\epsilon \geqslant 0$ in the following linear programming problem:

$$\begin{aligned} &\text{Max} && \epsilon, \\ &\text{Subject to :} && \sum_{i=1, i\neq k}^{M} \mu_i (x_k - x_i) - \epsilon \geqslant x^* - x_k, \\ & && \mu_i \geqslant 0 \quad \forall i, \end{aligned} \tag{1}$$

it follows that $f(x_k) \geqslant f(x^*)$". Given that x_k is the vertex of the cone and x^* is shown to be less preferred than that point, we will refer to the point x^* as being cone dominated.

Sending a sample of six solutions to the DM, and using the DM provided preference information, it is possible to construct five 2-point convex preference cones by observing that the best member is a point and each of the other five members of the set is a vertex. We also form a single 6-point preference cone with the worst point in the sample at the vertex. This process produces five unique preference cones per call to the DM since the 2-point cone formed between the best and the worst is redundant to the 6-point cone.

The convex cones are formed from the objective function values of the population members selected for evaluation (and exist in objective space), hence they are independent of the population in that the information in the cones is not dependent on the continuing existence of the population members that created them. The information they contain is dependent solely upon the true preference function. Due to this characteristic, all preference cones are retained throughout all the genetic algorithm's generations, whether or not the members of the population from which they were derived are still surviving. Note that these cones are valid for the underlying preference function regardless of which solutions are included in the cones as proven in Korhonen et al. (1984).

2.3. Preference order the population

Placing the population in preference order is divided into two steps. First, the algorithm selects members of the population for DM evaluation so the algorithm can elicit preference information. Second, the algorithm uses the preference information to sort the population. These two steps are explained in the sections below.

2.3.1. Selecting members for DM Evaluation

The members selected for review by the DM will impact the power of the cones created as well as the regions of the objective space that fall under the cone. For this reason, we select solutions to send to the DM from the set of nearly feasible non-dominated solutions. First, we add solutions that cannot be ranked using the already existing preference cones. If this sample is too small, we randomly select members from the set of feasible non-dominated solutions to supplement the sample. Including these additional members provides an opportunity to create cones powerful enough to provide information about new solutions. Pseudo code for this algorithm is provided below. Note that a better solution at the vertex of a cone results in a more powerful cone, so not sending dominated solutions to the DM actually improves the ability of the algorithm to evaluate solutions.

```
for all members of the population
  if population[0] preferred to population[i] by cone or direct
   information
  else
    sample = sample + population[i]
  end if
end for
while size(sample) < 6
  index = (int) rand( ) * number of feasible non-dominated
   solutions + 1
  sample = sample + population[index]
end while
```

The selected members are sent to the DM, who tells us the best and worst of the sample. The objective values of the worst solution define the coordinates of the vertex, and the other solutions define the coordinates of the other points in the cone. Note that the best solution from the most recent DM interaction is retained independent of the population as we discuss in Section 2.8.

2.3.2. Preference order the population

Any time new convex preference cones or population members are formed, the next step is to put the population into preference order. (This occurs immediately after the first DM interaction, hence it is explained here.) The most preferred solution is placed first, the next best solution is placed second, and so on. This is a partial ordering of the population since we do not know the DM's true preference function and, therefore, may not always know whether or not one solution is preferred to another. The three steps used to efficiently sort the population are listed below and illustrated in Fig. 1. Note that 1, 2, 3 in Fig. 1 implies a partial order and is not representative of a fixed number of solutions in the cluster.

1. Sort by feasibility (F indicates feasible, I indicates infeasible) (if applicable)
2. Sort by Pareto dominance (ND indicates not dominated, D indicates dominated)
3. Sort with cone dominance and information obtained from DM interaction

The first step in creating the partial order is to divide the population into feasible (F) and infeasible (I) clusters in problems where

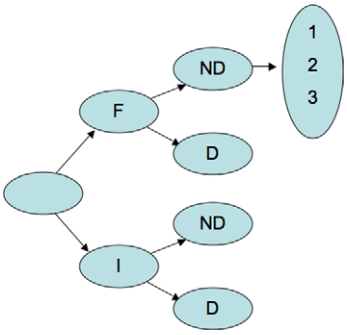

Fig. 1. Population member clustering diagram. Key: F = feasible solution, I = infeasible solution, ND = non-dominated solution, D = dominated solution, 1 2 3 = partial order.

infeasible population members can be generated (Deb et al., 2002). Alternatively, the genetic encoding may be designed not to permit the creation of infeasible population members or a repair operator may be used. Whether using a repair operator or penalizing infeasibility is more preferred depends on the problem being investigated and the genetic representation of the solution in the algorithm. If infeasible solutions cannot be created, sorting by feasibility is skipped and we proceed directly to considering Pareto dominance.

The second sort is conducted on Pareto dominance. Since the true preference function of the objectives and the set of efficient solutions are unknown, this sorting checks whether or not another member of the population is at least as good in all objectives and better at least in one objective as the population member in question. If this condition is met, the member is considered dominated (D), and that member is placed in a less preferred cluster than a non-dominated (ND) member (Deb et al., 2002).

For computational efficiency, the final sorting is performed only on the most preferred cluster, the feasible non-dominated members. In this sort, the top cluster is placed in preference order using the convex preference cones and DM preference information elicited in step 2. Specifically, if solution i is under cone K and solution j is within cone K's polyhedron (spanned by points 4, 5, and 6 in Fig. 2), solution j will appear before solution i in the partial order. In terms of the cones, this means that, if a solution is under a cone (like point 1 in Fig. 2), it will appear after the vertex of the cone (point 4 in Fig. 2), which will appear after all the solutions in the cone's polyhedron (like point 2 in Fig. 2). This is repeated for all the cones generated. The DM preference information is applied by placing solution j before solution i if j is within the cone's polyhedron and i is the vertex of the cone. All this follows from the quasi-concavity of the underlying value function. (Note that while any point in the cone's polyhedron is preferred to the vertex, the point designated as the best in the cone is not necessarily preferred to every other point in the polyhedron since we allow nonlinear preference functions. In relation to Fig. 2, imagine that the upper two white points are equally preferred by the DM, but that a point directly in between them is more preferred. This would occur if the isopreference lines in the objective space were upward-opening parabolas.)

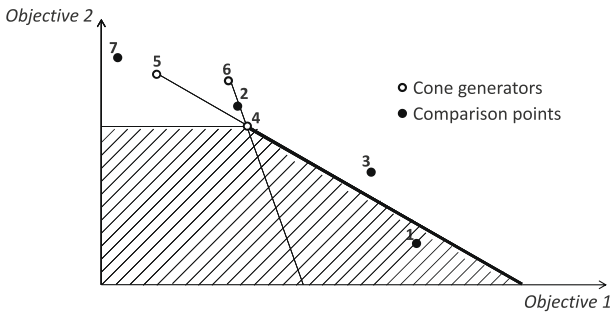

Fig. 2. Possible locations of a solution with respect to a cone.

The above steps are conducted for each member of the population, and a simple interchange sort exchanges the position of two members of the population if the earlier one is found to be less preferred than the later one. In this way, the best solution (and those members of the population that cannot be ranked against it) tends to rise to the top of the population. Feasible non-dominated solutions that are less preferred than other members of the population will come next, and the dominated and infeasible clusters of solutions appear last.

The preference cone (the shaded region in Fig. 2) is constructed from two binary comparisons: solution 5 is preferred to solution 4, and solution 6 is preferred to solution 4. As shown in Korhonen et al. (1984), it follows from the quasi-concavity of the underlying preference function that the shaded cone (and naturally all solutions which lie "under" the cone) contains solutions that are inferior to solution 4, as well as to solutions that lie in the cone's polyhedron (solutions spanned by 4, 5, and 6.)

To perform the above ordering, first note that any population member can have four possible locations relative to a convex preference cone. It can be: (1) under the cone, (2) in the cone's polyhedron, (3) and (7) outside the cone, or (4) the cone's vertex. These possible locations are shown for a 3-point cone in Fig. 2, where the white solutions define the cone and the black solutions are points being compared to the cone. Finding a member's location relative to a cone requires solving two linear programs. The formulation to determine if a member falls under a cone is presented in Korhonen et al. (1984), and is given above in Eq. (1). The formulation to determine if the member falls within the polyhedron follows directly from Lemma 1 of Korhonen et al. (1984), and is given in Eq. (2). Essentially, the linear program (LP) checks to see if a solution's objective values can be written as a convex combination of the objective values of the points that make up the cone. If the above condition is true (i.e., if there is a feasible solution to Eq. (2)), the solution lies within the cone's polyhedron.

Let X_j denote the jth solution, $z_i(X_j)$ denote the value of the ith objective of the jth solution, and K denote the set of points generating the cone. If a feasible solution exists to the following LP, the solution $X_j,\ j \notin K$ falls within the polyhedron of the cone.

$$\begin{aligned} &\text{Min} && 0, \\ &\text{Subject to :} && z_j(X_j) = \sum_{k\in K} \mu_k z_i(X_k), \\ & && \sum_{k\in K} \mu_k = 1, \\ & && \mu_k \geqslant 0 \quad \text{for all } k. \end{aligned} \tag{2}$$

2.4. Replace the worst r% of the population

Since the population is already clustered into four sets and ordered from most preferred to least preferred, the last members of the last set in population are the worst. We replace the worst ranked 20% of the population every generation by removing them from the end of the population listing.

2.5. Select parents

In order to replace the less fit members of the population, new members are bred from two parents. The first parent is selected probabilistically from the population. The probability of selection

is given in Eq. (3) below. The second parent is selected randomly (with equal probability) from the population.

Let w be the population size, and let i be the index of the solution in the (partially ordered) population. (Note that a smaller i implies higher preference, and that the first solution is stored with index 0.) Then the probability of selection is given by

$$P(\text{select}) = \frac{w-i}{\sum_{j=0}^{w}(w-j)}. \quad (3)$$

2.6. Breed the parents and mutate the offspring

A simple uniform crossover operator is used to combine the chromosomes of the two parents (Michalewicz, 1996). We set the crossover operator such that there is a 50% probability that the child's ith chromosome will come from the first parent (independent of any other chromosome selections.) Again, there is no feasibility check imposed on the offspring. The clustering and subsequent replacement of the worst members is used to remove infeasible solutions from the population. Approximately 2/3 of the total replacements are made in this way.

In order to preserve diversity in the population, the remaining population members selected for replacement are replaced with immigration (Ahuja et al., 2000). These immigrant solutions are generated by the same technique used to place random members in the initial population.

Regardless of whether the new members were generated by breeding or randomly, we set the mutation probability of the new member to 90% as in Phelps and Köksalan (2003). The mutation exchanges two randomly selected chromosomes in the new population member. The high mutation probability combined with the relatively gentle mutation operation slightly alters new population members and prevents premature convergence of the population. Since the best solution is maintained independently from the population as discussed in Section 2.8, the high mutation probability will not adversely affect solutions already in the population.

2.7. Update the population

Once breeding is complete, the new members are added to the population, their objective functions are calculated, and the partial order is updated. From here, the algorithm repeats from either step 2 or step 3 in order to keep the number of calls to the DM at a reasonable level. The best method (that is, frequency and pattern) of allocating calls to the DM is an ongoing research topic.

2.8. Update the best solution

The best solution (defined as the first member of the population sorted using information from the cones) is selected at the end of each generation. This solution is stored independent of the population, and is called the *incumbent*. Each time the population is updated, the new best solution is compared to the incumbent. The incumbent solution is replaced with the new best solution after each iteration, unless the new best solution is dominated by the incumbent. This replacement condition is imposed as a sanity check to ensure superior solutions are not discarded.

In the final generation, the first four members in the partial order, the current incumbent solution, and the best solution from the most recent DM interaction are sent to the DM as a final six-member sample. The best of this sample is the reported solution.

3. Experimentation

Several design and implementation issues arose while developing this genetic algorithm. Like most genetic algorithms, we needed to decide on the best population size, the number of generations to run, and several other similar issues. We addressed this parameter tuning problem using an experimental design framework.

For all experiments, the DM was replaced with a 'function robot' that would evaluate the sample of solutions sent the DM and return the best and the worst of the sample based on randomly generated preference function coefficients. This allowed us to experiment on the algorithm without issues of DM fatigue or inconsistency. It also reduces the computation time of the algorithm to less than a minute per problem instance.

3.1. Problem

For testing purposes and to allow us to benchmark our results, we applied this algorithm to the multi-objective 0–1 knapsack problem, as in Phelps and Köksalan (2003). Since that approach is the first interactive approach for evolutionary multi-objective optimization (EMO) and since we build upon their approach we decided to use the same problems. Our results are based on replicated instances of different randomly generated problems. There are other test problems available for the knapsack problems in the literature. These are mostly two-objective problems. We consider problems with three and four-objectives in addition to two-objectives. This was an important part of our tests, since a major problem with existing EMO algorithms has been the fact that they have largely been developed for 2-objective cases. This problem can be formulated as

$$\begin{aligned} \text{Max} \quad & \sum_{j=1}^{J} c_j^k d_j, \quad k = 1, \ldots, n, \\ \text{Subject to :} \quad & \sum_{j=1}^{J} a_j d_j \leqslant b, \\ & d_j \in \{0, 1\}, \end{aligned} \quad (4)$$

where b is the knapsack capacity, J is the number of candidate items for the knapsack, a_j is the volume of the jth item, d_j is a boolean indicating if the jth item is included in the knapsack, n is the number of objectives, and c_j^k is the value of the jth item to the kth objective. Selecting a research portfolio can be regarded as a knapsack problem where the DM must decide whether or not to fund each project. The goal is to maximize the value of the portfolio subject to a budget constraint. In general, this means that the algorithm is faced with a set of items that have a specified value to each objective and a volume. The objective is to maximize the total value of the items included in the knapsack without exceeding the volume constraint. The 0–1 variant of the knapsack means that an item can either be included once (1) or not at all (0). The modifications to the algorithm that are made specifically to address the 0–1 knapsack problem within the framework described in Section 2 are explained below. The random generation of problem instances and the experimental inputs are explained in Section 3.2.

For the knapsack problem, we elected to allow infeasible members to be generated and added to the population rather than to employ a repair operator. To promote genetic diversity, the volume constraint of the knapsack is inflated by a factor ϕ in the early generations. This temporary relaxation allows nearly feasible solutions (referred to as quasi-feasible) to be bred, in the hope that they will eventually turn into good feasible solutions. This practice must be stopped before the population converges to an infeasible solution, hence the relaxation of the volume constraint is progressively tightened to the actual volume constraint over the first 10 generations.

Since we wish to breed the quasi-feasible solutions described above, they must not be sorted into the infeasible class described

in Section 2.3. Instead, the first sorting step splits the population into quasi-feasible (QF) (more preferred) and infeasible (I) (less preferred) clusters. Let V_{max} denote the knapsack volume constraint, V_{member} the volume of the population member, and g the number of generations that have passed so far. If $V_{member} \leqslant V_{max}(1+\phi(g))$, the member is declared quasi-feasible. Otherwise, it is declared infeasible. The function $\phi(g)$, which calculates the amount of allowable volume constraint violation at generation g, is given by

$$\phi(g) = \max\left\{\phi_{initial} - \frac{\phi_{initial} * g}{10}, 0\right\}. \quad (5)$$

In this implementation, we set $\phi_{initial}$ to 0.3. Note that the denominator is the generation at which the volume constraint is fully imposed. The remaining two sorting steps proceed exactly as described in Section 2.3.

3.2. Experimental inputs

Each of the runs in the set of designed experiments is a multi-objective knapsack problem instance. For each run, the experimenter provides the information necessary to set up the problem. These input values uniquely determine the full course of the run (because the random number seed is an input). If the same inputs are repeated for two runs, the results will be identical. The input variables (and their investigated ranges) are:

- Preference function type (categorical: linear or Chebyshev)
- Population size (50, 70, and 90)
- Number of candidate items for the knapsack (problem size: 100–200 items)
- Knapsack volume constraint as a function of the sum of the sizes of the candidate items (0.5)
- Mutation probability (90%)
- Number of objectives (2, 3, and 4)
- Number of generations (20–500)
- Number of calls to the DM (0–10 calls total)
- Random number seed (randomly selected)

A few of these inputs require some discussion. First, we used two different preference functions, linear and Chebyshev, because they represent the bounds of a wide set of indifference curves. The linear preference function has linear indifference curves (in the objective space) while the Chebyshev function has indifference curves that contain a 90-degree bend. A wide set of common indifference curves falls between these two curves, suggesting that a technique that works for both extremes can be reasonably inferred to work for the curves that fall in between.

Section 2.7 mentions that the best method of calling the DM is an open research topic. For the purposes of this experiment, we set a number of DM calls, and distribute them evenly over the generations. The number of calls allows us to estimate DM effort required to achieve a level of performance with this algorithm.

Many of the other values are chosen to allow easy comparison to the results in Phelps and Köksalan (2003), including the volume constraint being 50% of the volume of the knapsack, the mutation probability, and the number of objectives that are being considered.

Given the inputs listed above, the specific parameters of each problem instance are created randomly in a fashion similar to that used in Phelps and Köksalan (2003). The three types of random parameters are: (i) the value of each item to each objective function if the object is included in the knapsack; (ii) the volume of each item; and (iii) the coefficients used in the preference function (shown in Eqs. (6) and (7) as α). The objective function coefficients and volume of each candidate item are uniformly distributed integers between 60 and 100. The random coefficients of each objective in the preference function are random numbers normalized such that the sum of all the coefficients is unity. Eqs. (6) and (7) give the general form of the linear and Chebyshev preference functions, respectively.

Let α_i be the normalized random coefficient of the ith objective function, $z_i(X)$ be the value of the ith objective of solution X, B_i be the ideal value of the ith objective, n be the number of objectives, Z be the value of the preference function, and S be the set of feasible solutions. Then Z is found by (6) for the linear preference function, and by (7) for the Chebyshev preference function.

$$\text{Max} \quad Z = \sum_{i=1}^{n} \alpha_i z_i(X_i) \quad (6)$$

$$\text{Subject to :} \quad X \in S$$

$$\text{Min} \quad Z = \text{Max}_i \alpha_i (z_i(X_i) - B_i) \quad (7)$$

$$\text{Subjectto :} \quad X \in S$$

If the underlying preference function is known, it is possible to rewrite the multi-objective knapsack problem as a single objective problem where the overall objective function is the preference function of the objectives. If the problem's set of non-dominated solutions is known, it is then possible to evaluate the preference function at each vector of objective function values corresponding to points in the non-dominated set. The best value is the optimal solution to the problem. (As pointed out above, the underlying preference function and the full efficient frontier are hard to find in practice, but are useful for testing our approach.)

It is worth noting that efficient solutions to the single objective problem described above tend to be very close to the optimal solution. For one problem instance with 200 items in the knapsack and two objectives, the full efficient frontier was generated and both linear and Chebyshev preference functions were applied with a range of coefficients. The maximum difference in preference function value between the best and worst efficient solution was less than 4% and the median deviation was 0.5% of the total value of the preference function. Considering these results, we expect the percentage difference between the best found and the output solution to be small in all cases since we only consider solutions that are non-dominated with respect to the genetic algorithm's population.

3.3. Experimental evaluation criteria

Since multi-objective knapsack problem instances are inherently difficult to solve, comparing the genetic algorithm's solution to the optimal solution is impractical. Instead, we establish a set of outputs that will allow comparison of the solutions without having to optimally solve instances of NP-hard problems. Let z_{bad} be the preference function value of a bad solution, z_{out} be the preference function value of the reported output solution, z_{best} be the preference function value of the best solution in the algorithm population (when searching with knowledge of the true preference function), and z_{lp} be the value of the preference function of the LP relaxation of the knapsack problem. The evaluation criteria are listed below.

- The percent deviation from the best found solution is given by

  $$\frac{z_{best} - z_{out}}{z_{best} - z_{bad}}$$

  and referred to as output versus best found,
- Percent deviation from the LP relaxation of the knapsack problem is given by

$$\frac{z_{lp} - z_{out}}{z_{lp} - z_{bad}}$$

and referred to as output versus LP relaxation,

- The preference function value of the reported solution (that would be output to the DM), z_{out}.

A measure of the deviation between the output solution and the best found with perfect information provides a measure of the algorithm's ability to sort the solutions generated by the genetic algorithm. (As noted earlier, we retain the best solution in the population with knowledge of the true preference function, but use that information only to create this performance measure.) The percent deviation from the LP relaxation provides a comparison between the algorithm's output and an easily generated bound on the optimal solution. Finally, the value of the solution's preference function is useful for comparing different options in the algorithm (such as the number of generations or the number of calls to the DM) for a given problem instance.

3.4. Experimental results

We designed experiments to determine which factors significantly affect the difference between the reported solution and the best solution found using the true preference function. The results (averaged over 10 replications) are shown in Table 1 for both two and four-objective function cases. It is clear that raising the number of objectives decreases the average performance of the algorithm (all other factors remaining the same). It is also clear that even for the modest number of generations in these runs, there are no effects for the two-objective case, since the algorithm returned the best found regardless of the factor levels.

A more complete experimental table containing averages from 10 replications of each input combination is shown below for both linear and Chebyshev preference functions. Tables 2 and 3 show the intuitive result that the deviation from the best found decreases with more calls to the DM using a population size of 70 and a volume fraction of 0.5. There are two main reasons for this. First, the algorithm is better able to sort the solutions in the population, and second, the algorithm is better guided so the population contains more preferred solutions. It also shows that the deviation from the best found increases with increased problem complexity (as measured by number of objectives) for both preference functions.

Table 1
Genetic algorithm factor screening experiment with a linear preference function.

Population size	Objects	Volume fraction	Number of DM calls	Deviation from best found	
				2 Objective cases	4 Objective cases (%)
50	100	0.3	15	0	0.11
90	100	0.3	5	0	0.00
50	200	0.3	5	0	0.28
90	200	0.3	15	0	0.00
50	100	0.5	5	0	0.00
90	100	0.5	15	0	0.26
50	200	0.5	15	0	0.00
90	200	0.5	5	0	0.72
50	100	0.3	5	0	0.00
90	100	0.3	15	0	0.00
50	200	0.3	15	0	0.00
90	200	0.3	5	0	0.08
50	100	0.5	15	0	0.00
90	100	0.5	5	0	0.11
50	200	0.5	5	0	0.00
90	200	0.5	15	0	0.00

Table 2
Linear preference function results.

Generations	Objectives	0 Calls	3 Calls	6 Calls	11 Calls
Output versus best found					
30	2	2.4%	0.0%	0.0%	0.0%
30	3	3.3%	2.1%	0.0%	0.0%
30	4	13.6%	3.0%	0.6%	0.0%
Output versus LP relaxation					
30	2	17.0%	15.1%	15.1%	15.1%
30	3	12.8%	11.6%	9.8%	9.8%
30	4	22.7%	13.2%	11.1%	10.5%
Average output preference function value					
30	2	8964	8993	8993	8993
30	3	8878	8958	8966	8966
30	4	8849	8942	8968	8972

Table 3
Chebyshev preference function results.

Generations	Objectives	0 Calls	3 Calls	6 Calls	11 Calls
Output versus best found					
30	2	6.1%	0.0%	0.0%	0.0%
30	3	21.1%	18.5%	0.0%	0.0%
30	4	36.0%	10.9%	1.4%	0.0%
Output versus LP relaxation					
30	2	23.3%	18.7%	18.7%	18.7%
30	3	26.0%	23.6%	5.7%	5.7%
30	4	43.4%	21.1%	12.7%	11.5%
Average output preference function value					
30	2	−4904	−4848	−4848	−4848
30	3	−3429	−3330	−3258	−3258
30	4	−2830	−2719	−2680	−2677

Zero deviation from the best found can be created by increasing the effort expended in the four-objective case. This is a fortunate result, since our response (with its minimum value of zero) violates the model assumptions for a standard designed experiment even after the response is transformed using the Box–Cox method (Myers and Montgomery, 2002).

More interestingly, we can state that the performance of the algorithm is affected by problem complexity. We infer that similar effects would result from applying the technique to other problem types. For this reason, we recommend running at least 50 generations and terminating the algorithm when the DM is satisfied with the solution presented rather than setting a fixed budget of interactions. Our experiments have shown that satisfactory performance can be achieved using a reasonable number of DM interactions.

The performance of the algorithm appears to be robust to variations in population size and the number of generations executed between calls to the DM within the ranges presented in this section. Since the computer only requires a few seconds between DM calls for the ranges considered, the number of generations between calls is far less relevant to algorithm performance than the total number of DM calls. Both the dependence on problem complexity and the robustness to a reasonable range of genetic algorithm parameters is consistent with the conclusions presented in Phelps and Köksalan (2003) for a similar application. The granularity in the number of DM calls required is adequate to show the trends in the data, and the problem complexity dependence reinforces the conclusion that DM satisfaction is a better stopping criterion than a count of the number of DM calls.

We believe the most compelling conclusion in this data is the rapid convergence of the output solution to the best found solution. The proposed framework can be applied to genetic algorithms that have been tuned to specific problems. Since GA tuning

methods for specific problems is widely available in the literature, we did not attempt to tune this algorithm to the types of knapsack problem instances being evaluated. Hence the top part of Tables 2 and 3 is more important than the comparison to the LP relaxations. The results show that we can do as well as any genetic algorithm. The rest is up to tuning the GA.

3.5. Effectiveness of the convex preference cones

An important question that remains is an analysis of the usefulness of the cones when sorting solutions.

There are three obvious approaches to answering this question. Since the population is sorted using a simple exchange sort, it is logical to ask what percentage of these exchanges require information from the cones. For the cases presented above, on average, 67% of the exchanges required cone information beyond what is provided directly by the DM.

The second approach is to determine how much difference there is in the output solution if the cone evaluation is turned off. While there are few conclusive results available from this method, it is interesting to note that the output solutions were uniformly worse for this algorithm than the algorithm that uses the cones when three calls are made to the DM. These differences tend to diminish once six calls are made, but these results suggest that the cones help the algorithm make better use of sparse information.

Finally, we can ask how many calls would be required to fully sort the population, and how this perfect sort would affect the solution the algorithm returns. (Note that we still sort the population based on feasibility and dominance as outlined above and the DM is only called to sort the feasible non-dominated population members.) The maximum improvement over the approach using convex cones is a 2% increase in the preference function, and the minimum number of DM calls required in the approach is over 8000. The average number of calls required is nearly 40,000, whereas our algorithm only requires six to 10 calls to achieve similar performance.

All of the evaluation techniques show that the cones play a major role in the functioning of the genetic algorithm. In the first evaluation, the cones are shown to be important to sorting the population. The second evaluation technique shows that the addition of the cones improves the output solution in cases with fewer DM interactions (and having fewer interactions is preferred in interactive algorithms). The final approach shows that even a perfect sorting of the population provides very little additional information over that provided by the cones, and the amount of DM effort required to achieve these results is staggering.

4. Conclusions and future work

We have developed and tested a new evolutionary approach to interactive multi-objective optimization by showing how convex preference cones can be used to sort solutions to multi-objective combinatorial optimization problems. In turn, this guided search reduces the efficient solutions to the region of the solution space most preferred by the DM. This guided search can also reduce the number of efficient solutions that must be generated for the DM to evaluate. Applying this technique requires only that the underlying preference function be quasi-concave for the convex preference cones to be valid.

The computational experiments show it is possible to obtain solutions with a reasonable number of DM interactions that are very near to or equal to the best found by a similar algorithm that is operating with perfect knowledge of the user's preference function. We also present a reasonable range of parameters for the genetic algorithm to achieve these results. Clearly, if a more complex problem instance must be solved, additional effort should be expended.

Most existing preference-based multi-objective evolutionary algorithms are ad hoc in the sense that they are not backed with theory in converging to the preferred regions. Many use a reference point or a reference direction to guide the algorithm. We believe that our approach will open up a new research direction in multi-objective evolutionary research. We expect other researchers to pursue developing approaches that formally use the properties of the DM's underlying preferences. The idea of convex preference cones derived from quasi-concave utility functions have been widely used in the multi-criteria decision making literature. They can be incorporated into other multi-objective evolutionary algorithms and other heuristic search techniques as well. The mechanisms and details of such approaches and testing their performances await future research.

There are several other future research ideas suggested by our results. Investigating the effect of DM inconsistencies on the algorithm is a logical next step. Also, using sequential sampling to select points to go to the DM in order to maximize the information provided by each cone can improve the information provided by each interaction. Finally, altering the sample size sent to the DM has the potential to improve the power of the cones and reduce DM burden.

References

Ahuja, Ravindra, Orlin, James, Tiwari, Ashish, 2000. A greedy genetic algorithm for the quadratic assignment problem. Computers and Operations Research 27, 917–934.

Branke, J., Deb, Kalyanmoy, 2004. Integrating user preferences into evolutionary multi-objective optimization. KanGal Report, Indian Institute of Technology, Kanpur, India.

Caballero, R., Luque, M., Molina, M., Ruiz, F., 2002. Promoin: An interactive system for multiobjective programming. International Journal of Information Technology and Decision Making 1, 635–656.

Coello Coello, Carlos A., 2000. Handling preferences in evolutionary multiobjective optimization: A survey. In: Proceedings of the 2000 Congress on Evolutionary Computation, pp. 30–37.

Cvetkovic, Dragan, Parmee, Ian C., 2002. Preferences and their application in evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation 6, 42–57.

Deb, Kalyanmoy, 2001. Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester.

Deb, Kalyanmoy, Pratap, Amrit, Agarwal, Sameer, Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6, 182–197.

Deb, Kalyanmoy, Sundar, J., Uday, B.R.N., 2005. Reference point based multi-objective optimization using evolutionary algorithms. KanGal Report, Indian Institute of Technology, Kanpur, India.

Dyer, James, Fishburn, Peter, Steuer, Ralph, Wallenius, Jyrki, Zionts, Stanley, 1992. Multiple criteria decision making, multiattribute utility theory: The next ten years. Management Science 38, 645–654.

Fonseca, C.M., Fleming, P.J., 1993. Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416–423.

Fonseca, C.M., Fleming, P.J., 1998. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. Part II: Application example. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 28, 38–47.

Geoffrion, A.M., Dyer, J.S., Feinberg, A., 1972. An interactive approach for multicriterion optimization, with an application to the operation of an academic department. Management Science 19, 683–694.

Hanne, T., 2005. Interactive decision support based on multiobjective evolutionary algorithms. Operations Research Proceedings (GOR), 761–766.

Horn, J., Nafploitis, N., Goldberg, D., 1995. A niched pareto genetic algorithm for multi-objective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 82–87.

Kamalian, R., Takagi, H., Agogino, A., 2004. Optimized design of MEMS by evolutionary multi-objective optimization with interactive evolutionary computation. In: Genetic and Evolutionary Computation Conference (GECCO), pp. 1030–1041.

Köksalan, Murat, Phelps, Selcen (Pamuk), 2007. An evolutionary metaheuristic for approximating preference-nondominated solutions. INFORMS Journal on Computing 19, 291–301.

Kondakci, S., Azizoglu, M., Köksalan, M., 1996. Note: Bicriteria scheduling minimizing flowtime and maximum tardiness. Naval Research Logistics 43, 929–936.

Korhonen, Pekka, Wallenius, Jyrki, Zionts, Stanley, 1984. Solving the discrete multiple criteria problem using convex cones. Management Science 30, 1336–1345.

Michalewicz, Zbigniew, 1996. Genetic Algorithms + Data Structures = Evolution Algorithms, third ed. Springer, Berlin, Heidelberg, and New York.

Miettinen, Kaisa, 1999. Nonlinear Multiobjective Optimization. Kluwer Academic Publishers., Boston, London, and Dordrecht.

Miller, George A., 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. The Psychological Review 63, 81–97.

Molina, J., Santana, L.V., Hernandez-Diaz, A.G., Coello Coello, C.A., Caballero, R., 2009. G-dominance: Reference point based dominance for multiobjective metaheuristics. European Journal of Operational Research 197, 685–692.

Myers, Raymond, Montgomery, Douglas, 2002. Response Surface Methodology: Process and Product Optimization Using Designed Experiments, second ed. Wiley, New York.

Parmee, I.C., Cvetkovic, D., Bonham, C.R., Packahm, I.S., 2001. Introducing prototype interactive evolutionary systems for ill-defined multi-objective design environments. Advances in Engineering Software 32, 429–441.

Phelps, Selcen (Pamuk), Köksalan, Murat, 2003. An interactive evolutionary metaheuristic for multiobjective combinatorial optimization. Management Science 49, 1726–1738.

Poles, S., Vassileva, M., Sasaki, D., 2006. Multiobjective optimization software. In: Branke, J., Deb, K., Miettinen, K., Slowinski, R. (Eds.), Published as a Chapter in Springer State-of-the-Art Survey LNCS 5252 Multiobjective Optimization: Interactive and Evolutionary Approaches.

Schaffer, J.D., 1984. Some experiments in machine learning using vector evaluated genetic algorithms. Ph.D. Thesis, Vanderbilt University, Nashville, Tennessee.

Srinivas, N., Deb, K., 1994. Multi-objective function optimization using non-dominated sorting genetic algorithms. Evolutionary Computation Journal 2, 221–248.

Wall, Matthew, 1995. Massachusetts Institute of Technology's GAlib Library. <http://web.mit.edu/galib/www/GAlib.html> (accessed 15.02.06).

Wallenius, J., Dyer, J.S., Fishburn, P.C., Steuer, R.E., Zionts, S., Deb, K., 2008. Multiple criteria decision making/multiattribute utility theory: Recent accomplishments and what lies ahead. Management Science 54, 1336–1349.

Zionts, S., Wallenius, J., 1976. An interactive programming method for solving the multiple criteria problem. Management Science 22, 652–663.