Multiobjective Prototype Optimization with Evolved Improvement Steps

Jiri Kubalik[1], Richard Mordinyi[2], and Stefan Biffl[3]

[1] Department of Cybernetics
Czech Technical University in Prague
Technicka 2, 166 27 Prague 6, Czech Republic
kubalik@labe.felk.cvut.cz
[2] Space-Based Computing Group
Institute of Computer Languages
Vienna University of Technology
Argentinierstr. 8, A-1040 Vienna, Austria
richard@complang.tuwien.ac.at
[3] Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstr. 9/188, A-1040 Vienna, Austria
Stefan.Biffl@tuwien.ac.at

Abstract. Recently, a new iterative optimization framework utilizing an evolutionary algorithm called "Prototype Optimization with Evolved iMprovement Steps" (POEMS) was introduced, which showed good performance on hard optimization problems - large instances of TSP and real-valued optimization problems. Especially, on discrete optimization problems such as the TSP the algorithm exhibited much better search capabilities than the standard evolutionary approaches. In many real-world optimization problems a solution is sought for multiple (conflicting) optimization criteria. This paper proposes a multiobjective version of the POEMS algorithm (mPOEMS), which was experimentally evaluated on the multiobjective 0/1 knapsack problem with alternative multiobjective evolutionary algorithms. Major result of the experiments was that the proposed algorithm performed comparable to or better than the alternative algorithms.

Keywords: multiobjective optimization, evolutionary algorithms, multiobjective 0/1 knapsack problem.

1 Introduction

In many real-world optimization problems a solution is sought that is optimal with respect to multiple (often conflicting) optimization criteria. Multiple objectives specify quality measures of solutions that typically do not result in a single optimal solution. Instead there is a set of alternative solutions that are optimal in a sense that (*i*) none of them is superior to the others and (*ii*) there is no superior solution in the search space that to these optimal solutions considering

J. van Hemert and C. Cotta (Eds.): EvoCOP 2008, LNCS 4972, pp. 218–229, 2008.

all objectives. Thus, a good multi-objective optimization technique must be able to search for a set of optimal solutions concurrently in a single run.

For this purpose, evolutionary algorithms seem to be well suited because they evolve a population of diverse solutions in parallel. Many evolutionary-based approaches for solving multiobjective optimization problems have been proposed in the last 25 years.

The recently introduced POEMS optimization framework proved to be efficient for solving hard optimization problems - the Traveling Salesman Problem (TSP)[4], a binary string optimization problem [4], a real-valued parameter optimization problem [5], and a network flow optimization problem [6]. This paper introduces an extension of the basic POEMS algorithm for solving multiobjective optimizations.

For an experimental evaluation of the presented approach a multiobjective 0/1 knapsack problem was used, which is a well-known NP hard combinatorial optimization problem, the particular formulation of the problem is given in Section 5. Results achieved by our approach were analyzed and compared to several evolutionary-based approaches presented in [9]. First results indicate that the proposed multiobjective POEMS algorithm performs very well on the test problem. It also scales well as it outperforms the alternative algorithms even on the largest instances of the problem.

This paper is structured as follows. First, a short overview of multiobjective optimization techniques is given with a focus on evolutionary-based approaches. Section 3 briefly describes the single-objective POEMS algorithm. In section 4, the multiobjective version mPOEMS is introduced. Section 5 describes the test problem, test datasets, and the configuration of the multiobjective POEMS algorithm used in the experiments. Results achieved with our approach are analyzed in section 6. Section 7 concludes and suggests directions for analyzing and improving the proposed approach.

2 Multiobjective Optimization Techniques

There are many evolutionary approaches for solving multiobjective optimization problems. The most distinguishing features are (*i*) the fitness assignment strategy for evaluating the potential solutions, (*ii*) the evolutionary model with a specific selection and replacement strategy, and (*iii*) how the diversity of the evolved population is preserved. Note the last issue is extremely important as the desired outcome of the algorithm is a set of optimal solutions that is as diverse as possible. One of the early approaches is Schaffer's Vector Evaluated Genetic Algorithm (VEGA) [7] that does not make the use of a single fitness value when selecting solutions to a mating pool. Instead, it carries out selection for each objective separately. Then, crossover and mutation are used in a standard way. Another approaches make the use of a weighted-sum aggregation of objectives in order to assign a scalar fitness value to solutions, see [1]. However, such methods are highly sensitive to the weight vector used in the scalarization process.

Perhaps the most widespread and successful are multiobjective evolutionary algorithms that use a concept of *dominance* for ranking of solutions. By definition [1], a solution x dominates the other solution y, if the solution x is no worse than y in all objectives and the solution x is strictly better than y in at least one objective. Naturally, if solution x dominates solution y then x is considered better than y in the context of multiobjective optimization. However, many times there are two different solutions such that neither of them can be said to be better than the other with respect to all objectives. When this happens between two solutions, they are called *non-dominated* solutions.

The concept of dominance can be used to divide a finite set S of solutions chosen from the search space into two non-overlapping sets, the *non-dominated set* S_1 and the *dominated set* S_2. The set S_1 contains all solutions that do not dominate each other. The set S_2, which is a complement of S_1, contains solutions that are dominated by at least one solution of S_1. If the set S is the whole feasible search space then the set S_1 is a set of optimal solutions called *Pareto-optimal* solutions and the curve formed by joining these solutions is called a Pareto-optimal front. Note that in the absence of any higher-level information, all Pareto-optimal solutions are equally important [1]. That is why the goal in a multiobjective optimization is to find a set of solutions that is (i) as close as possible to the Pareto-optimal front and (ii) as diverse as possible so that the solutions are uniformly distributed along the whole Pareto-optimal front.

Of the Pareto-based approaches, perhaps the most well-known are Pareto Archived Evolution Strategy (PAES) [3], Non-dominated Sorting GA (NSGA and NSGA-II) and Strength Pareto Evolutionary Algorithm (SPEA and SPEA2). We just briefly describe the NSGA-II [2] and SPEA2 [9] algorithms, because we chose them for as alternative approaches for the empirical comparison with our approach.

SPEA2 uses a regular population and an archive (a set of constant size of best solutions found so far). An archive truncation method guarantees that the boundary solutions are preserved. Fitness assignment scheme takes for each individual into account how many individuals it dominates and it is dominated by which is further refined by the incorporation of density information. In order to maintain a good spread of solutions, NSGA-II uses a density estimation metric called *crowding distance*. The crowding distance of a given solution is defined as the largest cuboid enclosing the solution without including any other solution in the population. Then, so called *crowding comparison operator* guides the selection process towards solutions of the best non-domination rank and with crowding distance. In each generation, a population Q_t of offspring solutions is generated from the current population of solutions P_t. The two populations are merged together resulting in the temporary population R_t of size $2 \cdot N$, where N is the population size. From this population a better half of solutions is chosen in the following way to constitute a new population P_{t+1}. First, the population R_t is sorted according to non-domination. Then the solutions are taken starting from the best non-domination level and are put to the new population P_{t+1}. If a set of solutions of currently processed non-domination level is bigger than the

remaining empty space in the population P_{t+1}, then the best solutions in terms of the crowding distance are used only.

3 Singleobjective POEMS

Standard evolutionary algorithms (EAs) typically evolve a population of candidate solutions to a given problem. Each of the candidate solutions encodes a complete solution, e.g., a complete set of the problem parameters in parameter optimizations, a complete schedule in the case of scheduling problems, or a complete tour for the traveling salesman problem. This implies, especially for large instances of the solved problem, that the EA operates with very big and complex structures.

In POEMS [4], the evolutionary algorithm does not operate on a population of complete solutions to the problem to be solved. Instead, one candidate solution, called the *prototype*, is generated at the beginning and then it is iteratively improved with the best-performing modification of the current prototype provided by an EA, see Figure 1.

The prototype modifications are represented as a sequence of primitive actions/operations, defined specifically for the problem at hand. The evaluation of action sequences is based on how well/badly they modify the current prototype, which is passed as an input parameter to the EA. Moreover, sequences that do not change the prototype at all are penalized in order to avoid generating useless trivial solutions. After the EA finishes, it is checked whether the best evolved sequence improves the current prototype or not. If an improvement is achieved, then the sequence is applied to the current prototype and resulting in the new prototype. Otherwise the current prototype remains unchanged for the next iteration. The process of iterative prototype improvement stops when the termination condition is fulfilled. A common termination condition is the number of fitness evaluations performed in the run.

The following paragraphs briefly discuss POEMS implementation issues.

Representation of action sequences. The EA evolves linear chromosomes of length *MaxGenes*, where each gene represents an instance of a certain action cho-

```
1 generate(Prototype)
2 repeat
3   BestSequence ← run_EA(Prototype)
4   Candidate ← apply(BestSequence,Prototype)
5   if(Candidate is_better_than Prototype)
6       Prototype ← Candidate
7 until(POEMS termination condition)
8 return Prototype
```

Fig. 1. An outline of the single-objective POEMS algorithm

sen from a set of elementary actions defined for the given problem. Each action is represented by a record, with an attribute *action_type* followed by parameters of the action. Besides actions that truly modify the prototype, there is also a special type of action called *nop* (no operation). Any action with $action\_type = nop$ is interpreted as a void action with no effect on the prototype, regardless of the values of its parameters. A chromosome can contain one or more instances of the *nop* operation. This way a variable effective length of chromosomes is implemented.

Operators. The representation of action sequences allows to use a variety of possible recombination and mutation operators such as standard 1-point, 2-point or uniform crossover and a simple gene-modifying mutation. In [4] a generalized uniform crossover was used, that forms a valid offspring as an arbitrary combination of parental genes. Both parents have the same probability of contributing their genes to the child, and each gene can be used only once. The mutation operator changes either the *action_type* (activates or inactivates the action) or the parameters of the action.

Evolutionary model. In general, the EA is expected to be executed many times during the POEMS run. Thus, it must be designed and configured to converge fast in order to get the output in short time. As the EA is evolving sequences of actions to improve the solution prototype, not the complete solution, the maximal length of chromosomes *MaxGenes* can be short compared to the size of the problem. For example *MaxGenes* would be much smaller than the size of the chromosome in case of binary string optimization or much smaller than the number of cities in case of the TSP problem [4]. The relaxed requirement on the expected EA output and the small size of evolved chromosomes enables to setup the EA so that it converges within a few generations. Examples of typical configurations can be found in [4], [5] and [6].

It is important to note, that the evolved improving alterations of the prototype do not represent just local moves around the prototype. In fact, long phenotypical as well as genotypical distances between the prototype and its modification can be observed if the algorithm possesses a sufficient explorative ability. The space of possible modifications of the current prototype is determined by the set of elementary actions and the maximum allowed length of evolved action sequences *MaxGenes*, see [4]. If the actions are less explorative and the sequences are shorter, the system searches in a prototype neighborhood only and is more prone to get stuck in a local optimum early. And vice versa, if larger alterations of the prototype can be evolved using the primitive actions, the exploration capability the algorithm allows to converge to better and hopefully globally optimal solutions.

4 Multiobjective POEMS

The multiobjective "Prototype Optimization with Evolved iMprovement Steps" (mPOEMS) belongs to a class of multiobjective optimization algorithms that

```
1 generate(SolutionBase)
2 repeat
3   Prototype ← choose_prototype(SolutionBase)
4   ActionSequences ← MOEA(Prototype, SolutionBase)
5   NewSolutions ← apply_to(ActionSequences, Prototype)
6   SolutionBase ← merge(NewSolutions,SolutionBase)
7 until(termination condition is fulfilled)
8 return SolutionBase
```

Fig. 2. An outline of the mPOEMS algorithm

uses the concept of dominance. In this section we describe the way the set of non-dominated solutions progressing towards the Pareto-optimal set is evolved in mPOEMS. Note that in multiobjective optimization the goal is to find a set of optimal solutions (as close as possible to the Pareto-optimal set) that are as diverse in both the variable space and the objective space as possible. Thus, the main differences between mPOEMS and POEMS are that

- mPOEMS maintains a set of best solutions found so far, called a *solution base*, not just one prototype solution that is maintained in POEMS. In each iteration of mPOEMS one solution from the set of non-dominated solutions in the solution base is chosen as the prototype for which the action sequences will be evolved by the EA.
- mPOEMS uses a kind of a multiobjective EA (MOEA) based on the dominance concept, not just a simple EA. The output of the MOEA is a set of action sequences (not just one action sequence) generating new solutions that are merged with the current solution base resulting in a new version of the solution base.

Figure 2 shows the main steps of the mPOEMS algorithm. It starts with generating the initial solutions of the solution base. The size of the solution base is denoted as $SBSize$ and stays constant through the whole mPOEMS run.

The first step of the main body of the iterative process is the selection of the prototype for the current iteration. The prototype is chosen among non-dominated solutions of the solution base in a way that guarantees that all parts of the non-dominated front of the evolved solution base are processed equally, see paragraph "prototype selection" below. The prototype is passed as an input parameter to the multiobjective EA, where the action sequences possibly altering the prototype towards the Pareto-optimal set are evolved. The other input parameter of MOEA is the current solution base that is used for evaluation purposes, see below. MOEA returns the final population of action sequences, which are then applied to the current prototype resulting in a set of new solutions.

Prototype selection. In each iteration a new prototype is chosen among non-dominated solutions of the solution base. The selection scheme is designed so that

all partitions of the non-domination set have as equal sampling rate as possible. In the first iteration a set S of n *candidate prototype* solutions is chosen according to the following procedure:

1. $S = \{\}$, $i = 1$, Choose a solution s_i by random
2. $i++$, choose a solution s_i so that its normalized Euclidean distance to the nearest solution in S is maximal,
   $S = S + s_i$,
3. Repeat Step 2 until $|S| = n$.

The Euclidean distance between two solutions i and j is calculated with the objective function values according to the following formula

$$d_{ij} = \sqrt{\sum_{k=1}^{m} (\frac{o_k^{(i)} - o_k^{(j)}}{u_k - l_k})^2},$$

where $o_k^{(i)}$ and $o_k^{(j)}$ are k-th objective values of solutions i and j, u_k and l_k are the upper and lower bounds for the k-th objective and m is the number of objectives. Each time a new prototype is to be chosen it is selected from the set S by random and removed from the set. Also if any solution in S becomes dominated by any solution in S it is removed from S. If the set is empty a new sample S of non-dominated solutions is selected according to the above described procedure.

The outline of the multiobjective EA used in mPOEMS is shown in Figure 3. First, it generates a starting population of action sequences of size $PopSize$. The action sequences are evaluated based on the quality of the solution that is produced by applying the given action sequences to the prototype. Then, the population of action sequences is evolved within a loop until some stopping condition is fulfilled. In the first step of the loop, a new population of action sequences is generated using standard operations of selection, crossover and mutation. The action sequences are evaluated and assigned fitness values. Finally, the new population and the old one are merged and $PopSize$ solutions of the best non-dominated fronts of that joint population are used to constitute the resulting population.

Fitness assignment schema. Since we are dealing with multiobjective optimization problems, each solution is assigned multiple objective values. The evaluation procedure uses a concept of dominance between solutions in order to find a single fitness value specifying the solution quality in terms of its non-domination level. In order to have more levels of non-domination that better distinguishes solutions the evaluated solutions are merged with solutions from the solution base resulting in a temporary set of solutions S (the prototype solution is included in the set S as well). The process of calculating the level of non-dominance starts with finding the non-dominated solutions among the whole set S. These solutions belong to the first level of non-domination front and are assigned a non-domination level $ND_{level} = 1$. Then they are temporarily disregarded from the set S and the set of non-dominated solutions is sought

```
input: Prototype, SolutionBase
output: Population of evolved action sequences

1 generate(OldPop)
2 evaluate(OldPop)
3 repeat
4     NewPop ← evolutionary_cycle(OldPop)
5     evaluate(NewPop)
6     OldPop ← merge(OldPop, NewPop)
7 until(EA termination condition is fulfilled)
8 return OldPop
```

Fig. 3. An outline of the multiobjective evolutionary algorithm used in mPOEMS

among the remaining solutions. These are the solutions of the second level of non-domination and are assigned a non-domination level $ND_{level} = 2$. The process goes on until there is no solution left in S, i.e. every solution has assigned its ND_{level} value. In the second phase of the evaluation procedure, the evaluated solutions are assigned their fitness value. Solutions that belong to a better than or the same level of non-domination as the prototype solution are assigned a fitness value equal to their ND_{level} value. Solutions with the ND_{level} higher than the prototype solution are assigned a fitness value equal to $ND_{level} + 0.5 * P_D$, where P_D is 1 if the given solution is dominated by the prototype, and 0 otherwise. Note that the smaller fitness the better solution. So, the selection pressure is towards the solutions that

1. belong to a better non-domination front than the prototype, if possible, and
2. are not dominated by the prototype solution.

Evolutionary model. New solutions produced by action sequences evolved by the MOEA are merged with the current solution base resulting in a temporary population of size $PopSize + SBSize$. From this population a new solution base of size $SBSize$ is selected according to the schema used in NSGA-II. First, the joint set is sorted based on the non-domination. Then the non-dominated fronts are added to the new solution base one by one, starting from the best non-dominated front. The non-dominated front that can not fit the whole into the remaining space in the new solution base is ranked according to the *crowding distance* value introduced in [2], and only the best solutions are added to the new solution base. This strategy together with the prototype selection scheme ensures that (i) the boundary solutions of the non-dominated front of the solution base will not get lost and (ii) the most unique solutions will retain in the solution base and (iii) the non-dominated front will be sampled uniformly.

5 Test Data and Experimental Setup

Test problem. For experimental evaluation we chose a well-known NP-hard Multiobjective 0/1 Knapsack Problem. We used the same formulation of the problem as was used in the comparative study by Zitzler and Thiele [8] and we compared results achieved by mPOEMS with alternative approaches presented there. As stated in [8], the multiobjective 0/1 knapsack problem is a good test problem, because its description is simple, yet the problem itself is difficult to solve and the problem is important in practice.

The multiobjective 0/1 knapsack problem considered in [8] is defined in the following way: Given a set of m items and a set of n knapsacks, with $p_{i,j}$ being profit of item j according to knapsack i, $w_{i,j}$ being weight of item j according to knapsack i, and c_i being capacity of knapsack i, find a vector $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in \{0,1\}^m$, such that

$$\forall i \in \{1, 2, \ldots, n\} : \sum_{j=1}^{m} w_{i,j} \cdot x_j \leq c_i$$

and for which $f(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ is maximum, where

$$f_i(\mathbf{x}) = \sum_{j=1}^{m} p_{i,j} \cdot x_j$$

and $x_j = 1$ iff item j is selected.

Since the solution of the problem is encoded as a binary string of length m, many codings do not represent a feasible solution (i.e. the capacity constraint of one or more knapsacks is violated). Thus, a greedy *heuristic repair algorithm*, which was also used in [8], was applied to every illegal solution. It removes items from the solution until all capacity constraints are fulfilled. Items with the least *profit/weight ratio* are removed first.

Compared algorithms. Out of the algorithms tested in [9] we chose the following two – SPEA2 and NSGA-II. We used the largest datasets with 750 items that were used in [9]. Both the datasets and the data of results presented in [9] are available on the web [10]. Thirty independent runs were carried out with mPOEMS on each test problem resulting in a similar set of values as in [9].

Performance measures and indicators. For all datasets the algorithms were compared on the basis of the following performance measures

- *Coverage of two sets* $C(X, Y)$, proposed in [8]. The measure is defined in the following way: Given the two sets of non-dominated solutions found by the compared algorithms, the measure $C(X, Y)$ returns a ratio of a number of solutions of Y that are dominated by or equal to any solution of X to the whole set Y. Thus, it returns values from the interval [0, 1]. The value $C(X, Y) = 1$ means that all solutions in Y are covered by solutions of the set X. And vice versa, the value $C(X, Y) = 0$ means that none of the solutions in Y are covered by the set X. For the problems with 2 knapsacks we

also present plots showing the tradeoff fronts constituted of non-dominated solutions of a union set of ten sets of non-dominated solutions obtained by each algorithm on the given dataset. The plots also show the Pareto-optimal front for the respective dataset if available.
- *Size of the space covered* $S(X)$, proposed in [8] and modified in [9]. A reference volume between the origin and an utopian objective vector (defined by the profit sums of all items in each objective) is taken into account. This measure is defined as a fraction of that volume that is not dominated by the final non-dominated solutions. So, the smaller the value of this measure the better the spread of solutions is, and vice versa.

Configuration of mPOEMS. In [9], the total number of solutions sampled (and evaluated) through the whole EA was 480.000 for 2 knapsacks, 576.000 for 3 knapsacks and 672 for 4 knapsacks. We used the same number of solution evaluations $N_{Evaluations}$ for each dataset. Parameters of mPOEMS were constant for all datasets as follows

- $PopulationSize = 70$. Size of the population of evolved action sequences.
- $SolutionBaseSize = 100$. The size of the solution base.
- $MaxGenes = 50$. The length of the evolved action sequences. Note, that it is much smaller than the solution size m (which is 750 in this study).
- $N_{Iterations}$ was 274, 330, and 384 for 2, 3, 4 knapsacks. The number of iterations in mPOEMS algorithm, see *repeat-until* cycle in Fig. 2.
- $N_{Generations=25}$. The number of generations carried out in the MOEA.
- $P_{Cross} = 0.8$, $P_{Mutate} = 0.2$. Probability of crossover and mutation.
- $Tournament = 3$. Parameter of the tournament selection used in MOEA.
- $n = 20$. A size of the set S of candidates for the prototype.
- Both the solution base as well as the starting population of action sequences in each iteration were initialized by random.

6 Results

Table 1 provides a comparison of mPOEMS and the other approaches on the basis of the coverage of two sets performance measure $C(X, Y)$. Each cell of the table is interpreted so that it indicates a proportion of non-dominated solutions obtained by the approach given in the corresponding column covered by the set of non-dominated solutions obtained by the approach given in the corresponding row. For example, for $n = 4$ we see that non-dominated solutions found by mPOEMS dominate 96.1% of non-dominated solutions found by NSGA-II and 97.1% of non-dominated solutions found by SPEA2, while only 0.1% of mPOEMS non-dominated solutions are dominated by solutions found by SPEA2 and none of them is dominated by solutions found by NSGA-II.

Table 2 shows the average values of the *size of the space covered* measure achieved by the compared algorithms. It shows that mPOEMS is no worse on any dataset and is significantly better (proved by t-test) on datasets with 2 and 4 knapsacks than the other two algorithms. This indicates, that mPOEMS finds a better spread of non-dominated solutions than NSGA-II and SPEA2.

Table 1. Comparison of mPOEMS, NSGA-II and SPEA2 using the *coverage of two sets* measure on datasets with 2, 3 and 4 knapsacks and 750 items. n is the number of knapsacks. Numbers in each cell show the fraction of non-dominated solutions obtained by the algorithm in the given column that are dominated by non-dominated solutions obtained by the algorithm in the given row.

		NSGA-II	SPEA2	mPOEMS
n=2	NSGA-II	-	0.604	0.178
	SPEA2	0.283	-	0.047
	mPOEMS	0.708	0.966	-
n=3	NSGA-II	-	0.02	0.0
	SPEA2	0.887	-	0.323
	mPOEMS	0.955	0.431	-
n=4	NSGA-II	-	0.006	0.0
	SPEA2	0.844	-	0.001
	mPOEMS	0.961	0.971	-

Table 2. Comparison of mPOEMS, NSGA-II and SPEA2 using the *size of the space covered* measure on datasets with 2, 3 and 4 knapsacks and 750 items

	NSGA-II	SPEA2	mPOEMS
$n = 2$	0.497	0.492	**0.490**
$n = 3$	0.689	0.689	0.688
$n = 4$	0.8195	0.822	**0.8180**

7 Conclusions and Future Work

This paper proposed a new multiobjective optimization algorithm mPOEMS based on the single-objective POEMS, recently introduced in [4]. mPOEMS extends the single-objective version of POEMS so that it (*i*) maintains a set of best solutions found so far called *solution base*, (*ii*) uses multiobjective EA (MOEA) instead of a simple EA to evolve a population alterations of the current prototype, and (*iii*) employs a strategy for proper selecting the prototype in each iteration.

mPOEMS was evaluated on a multiobjective 0/1 knapsack problem with 2 and 4 knapsacks (objectives) and 750 items. Results obtained by mPOEMS were compared to the results achieved by two state-of-the-art multiobjective evolutionary algorithms - NSGA-II and SPEA2, presented in [9]. The approaches were compared using a performance measure checking the mutual dominance of their outcomes and the size of the volume covered by the found non-dominated solutions. Performance of mPOEMS is at least as good or even better as the compared algorithms on all datasets. This is a very promising observation because the NSGA-II and SPEA2 were the best performing algorithms among the algorithms analyzed in [9].

As this is the first study on the mPOEMS algorithm, there are many open issues that should be investigated in the future:

- Computational complexity of the algorithm and its sensitivity to parameters' setting should be investigated.
- The performance of mPOEMS should be evaluated on other test problems with different characteristics, such as the problems with discontinuous Pareto-optimal front.
- Since the mPOEMS is in fact a local search approach which uses an EA to choose the next move to perform so it should be compared with other local search approaches as well.

Acknowledgement

Research described in the paper has been supported by the research program No. MSM 6840770012 "Transdisciplinary Research in Biomedical Engineering II" of the CTU in Prague.

References

1. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Ltd., New York (2002)
2. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation 6(2), 182–197 (2002)
3. Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the Pareto archived evolution strategy. Evolutionary Computation 8(2), 149–172 (2000)
4. Kubalik, J., Faigl, J.: Iterative Prototype Optimisation with Evolved Improvement Steps. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 154–165. Springer, Heidelberg (2006)
5. Kubalik, J.: Real-Parameter Optimization by Iterative Prototype Optimization with Evolved Improvement Steps. In: 2006 IEEE Congress on Evolutionary Computation, pp. 6823–6829. IEEE Computer Society, Los Alamitos (2006) [CD-ROM]
6. Kubalik, J., Mordinyi, R.: Optimizing Events Traffic in Event-based Systems by means of Evolutionary Algorithms. In: Event-Based IT Systems (EBITS 2007) organized in conjunction with the Second International Conference on Availability, Reliability and Security (ARES 2007), Vienna, April 10-13 (2007)
7. Schaffer, J.D.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. Genetic Algorithms and Their Applications. In: Proceedings of the First International Conference on Genetic Algorithms, pp. 93–100 (1985)
8. Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation 3(4), 257–271 (1999)
9. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm For Multiobjective Optimization. In: Evolutionary Methods for Design, Optimisation, and Control, Barcelona, Spain, pp. 19–26 (2002)
10. Zitzler, E., Laumanns, M.: Test Problem Suite: Test Problems and Test Data for Multiobjective Optimizers, `http://www.tik.ee.ethz.ch/~zitzler/testdata.html`