Genetic local search for multiple objective combinatorial optimization

Andrzej Jaszkiewicz[*]

Working paper
RA-014/98

December 1998

[*] Institute of Computing Science
Poznań University of Technology
ul. Piotrowo 3a, 60-965 Poznań, Poland
Jaszkiewicz@cs.put.poznan.pl
www.cs.put.poznan.pl/~and_j

Abstract

A new multiple objective metaheuristic algorithm based on genetic local search is presented and tested on travelling salesperson problem. The goal of the algorithm is to generate in a short time a set of solutions being a good approximation to the whole non-dominated set of a multiple objective combinatorial optimization problem. The algorithm in each iteration draws at random a utility function and constructs a temporary genetic population selecting a number of best solutions from the set of previously generated solutions. Randomly selected pairs of solutions from the temporary genetic population are recombined. Local search procedure is applied to each offspring. Results of the experiments show that the algorithm outperforms other multiple objective methods based on GLS on TSP instances.

1 Introduction

Multiple objective optimization of combinatorial problems is a real challenge nowadays. Many results obtained for multiple objective linear problems cannot be directly used in the case of multiple objective combinatorial optimization (MOCO) because of computational complexity of this kind of problems. The use of metaheuristcs seems to be the most promising approach to MOCO because of their efficiency, generality and relative simplicity of implementation.

Several authors have proposed multiple objective metaheuristic procedures that aim at the effective generation of approximations of the non-dominated set. The methods are usually based on classical single objective metaheuristics. For example, the methods of Schaffer (1985), Fonseca and Fleming (1993), Horn, Nafpliotis and Goldberg (1994), Srinivas and Deb (1995) are based on genetic algorithms, the methods of Serafini (1994), Ulungu et al. (1994), Czyzak and Jaszkiewicz (1995) are based on simulated annealing, and the methods of Gandibleux et. al. (1996) and Hansen (1997) are based on tabu search. In recent years we are observing, however, growing interest in hybrid single objective meteheuristic that combine elements of various methods. Typical example are genetic local search (GLS) methods combining genetic algorithms with locals search. Such methods often prove to be extremely efficient in combinatorial optimization (see e.g. Merz, Freisleben, 1997; Gorges-Schleuter, 1997).

We use the term “multiple objective metaheuristic” to characterize methods that generate approximations to the whole or parts of the non-dominated set, i.e. generate samples of approximately efficient solutions in a single run. Single objective metaheuristics can also be used in multiple objective context, for example to optimize a scalarizing function. In result a single approximately efficient solution is obtained.

The paper presents a new multiple objective genetic local search metaheuristic, called random directions multiple objective genetic local search (RD-MOGLS). The method is based on the idea of simultaneous optimization of all possible utility functions. In each iteration of the method a weighted Tchebycheff or weighted linear utility function is drawn at random. Then, a sample of best solutions is selected from the set of prior generated solutions. The sample is treated as a temporary genetic population. A number of randomly selected pairs of solutions from the temporary population are recombined. After recombination a local search is performed.

Multiple objective metaheuristics can be applied to both large scale MOCO problems and strongly non-linear continuous optimization problems. The aim of the proposed method is to work effectively on MOCO problems. Furthermore, the development of the method is

motivated by property of global convexity exhibited by many combinatorial optimization problems. Strongly non-linear continuous optimization problems and not globally convex MOCO may require different approaches.

A MOGLS method was proposed by Ishibuchi and Murata. Their method is discussed in section 3.3. Results of computational experiments reported in section 6 show that our method performs significantly better in the case of multiple objective TSP problems.

The paper is organized in the following way. In the next section some basic definitions are given. In the third section existing multiple objective genetic algorithms are discussed. Background of the proposed algorithm is presented in the fourth section. In the fifth section proposed multiple objective genetic local search algorithm is described. Computational experiments with the proposed algorithms are reported in the sixth section. In the last section conclusions and directions for further research are summarized.

2 Problem statement and basic definitions

The general multiple objective combinatorial optimization (MOCO) problem is formulated as:

$$\max\left\{f_1(\mathbf{x})=z_1,\ldots,f_J(\mathbf{x})=z_J\right\} \qquad \text{(P1)}$$

s.t.

$$\mathbf{x}\in D,$$

where: *solution* $\mathbf{x}=[x_1,\ldots,x_I]$ is a vector of discrete *decision variables*, D is the set of feasible solutions.

The image of a solution $\mathbf{x}$ in the objective space is a *point* $\mathbf{z}^{\mathbf{x}}=[z_1^{\mathbf{x}},\ldots,z_J^{\mathbf{x}}]$, such that $z_j^{\mathbf{x}}=f_j(\mathbf{x})$, j=1,..,J. The image of the set D in the criterion space is a set Z composed of *attainable* points, i.e. points being images of feasible solutions.

Point $\mathbf{z}\in Z$ *dominates* $\mathbf{z}'\in Z$, $\mathbf{z}\succ\mathbf{z}'$, if $z_j\geq z_j'$ $\forall j$, and $z_i>z_i'$ for at least one i.

Point $\mathbf{z}'\in Z$ is *non-dominated* if there is no $\mathbf{z}\in Z$ that dominates $\mathbf{z}'$. The set N of all non-dominated points is called the *non-dominated set*. A solution $\mathbf{x}$ is *efficient* if its image in the objective space is non-dominated.

The point $\mathbf{z}^*$ composed of the best attainable objective function values is called the *ideal point*:

$$z_j^*=\max\left\{z_j\,|\,\mathbf{z}\in Z\right\} \qquad j=1,\ldots,J.$$

The point $\mathbf{z}_*$ composed of the worst attainable objective function values in the nondominated set is called the *nadir point*.

Range equalization factors (Steuer, 1986, sec. 8.4.2) are defined in the following way:

$$\pi_j=\frac{1}{R_j},\ j=1,\ \ldots,\ J \qquad (1)$$

where R_j is the range of objective j in the set N or D. Objective function values multiplied by range equalization factors are called *normalized objective function values*.

A *utility function* $u:\Re^J \rightarrow \Re$, is a model of the DM's preferences that maps each point in the objective space into a value of utility. It is assumed that the goal of the DM is to maximize the utility.

A utility function u is *compatible with the dominance relation* if and only if $\forall \mathbf{z}^1, \mathbf{z}^2 \in \Re^J \quad \mathbf{z}^1 \succ \mathbf{z}^2 \Rightarrow u(\mathbf{z}^1) \geq u(\mathbf{z}^2)$. The set of all utility functions that are compatible with the dominance relation is denoted by U_c.

Weighted Tchebycheff utility functions are defined in the following way:

$$u_\infty(\mathbf{z}, \mathbf{z}^*, \Lambda) = -\max_j \{\lambda_i (z_j^* - z_j)\}. \tag{2}$$

Weighted linear utility functions are defined in the following way:

$$u_1(\mathbf{z}, \Lambda) = \sum_{j=1}^{J} \lambda_i z_j . \tag{3}$$

Weight vectors than meet the following conditions:

$$\forall j \ \lambda_j \geq 0, \sum_{j=1}^{J} \lambda_j = 1,$$

are called *normalized weight vectors*.

A finite set $A \subseteq Z$ is called an *approximation* of set N if $\forall \mathbf{z}^1, \mathbf{z}^2 \in A \quad \mathbf{z}^1 \not\succ \mathbf{z}^2 \wedge \mathbf{z}^2 \not\succ \mathbf{z}^1$, i.e. if it is composed of mutually non-dominated attainable points.

3 Existing multiple objective genetic methods

3.1 Pareto ranking based multiple objective genetic algorithms

Clearly, majority of research in the field of multiple objective metaheuristics concentrates on methods based on genetic algorithms (see Fonseca and Fleming, 1995, for review). It is often claimed that as GAs work with population (set) of solutions they are especially well suited for multiple objective optimization were the goal is to find a set of approximately efficient solutions. In all MOGAs, we are aware of, a single population of solutions is expected to approach and disperse over the whole (or, in some cases, interesting region of) the non-dominated set. We argue below that adaptation of GAs using a single population to multiple objective case, although possible, is not so natural and easy as is often claimed.

The first multiple objective genetic algorithm was proposed by Schaffer (1985). At present probably most often used are MOGAs based on Pareto ranking. In classical single objective GAs fitness of a solution, and thus probability of reproduction and/or survival, depends on value of the single objective. In Pareto ranking based MOGAs the fitness depends on ranking induced by dominance relation. For the first time this idea was proposed by Goldberg (1989). He proposed to assign the first rank to all non-dominated solutions in the current population, then the efficient solutions were removed, the second rank was assigned to efficient solutions in the remaining set, and so on. Fonseca and Fleming (1993) proposed a slightly different approach. In their method rank of a given solution corresponds to the number of solutions that dominate it. Pareto ranking based algorithms are clearly the most popular MOGAs now.

A clear advantage of Pareto ranking is its independence on any monotonic transformation of objective functions. Note, however, that this kind of fitness assignment may promote regions

with higher density of solutions. In such regions a given solution dominate many others, even if differences on objective values are very low.

Pareto ranking alone does not guaranties that the population will disperse over all regions of the non-dominated set. Fonseca and Fleming (1995) consider fitness landscapes induced by very large, uniformly distributed populations to model their algorithm. In this case all efficient solutions have highest fitness, i.e. define plateau of the fitness landscape. This situation is similar to optimization of single function having global optima at a plateau. It is well known that in this case finite populations converge to a single optimum. This phenomenon is called genetic drift (Goldberg and Segrest, 1987). In the multiple objective case, genetic drift means that finite populations tend to converge to small regions of the non-dominated set. Fonseca and Fleming (1993) and Srinivas and Deb (1994) propose the use of fitness sharing to prevent genetic drift. The idea of this technique is to penalize (decrease fitness) of solutions being too close, either in objective or in decision space, to some other solutions in the current population. They propose also some special approaches to set parameters of the fitness sharing.

In classical single objective GAs all solutions from the current population may be mated (recombined) to produce offsprings. This may be, however, very ineffective in multiple objective case. To explain it we need to analyze mechanism that make GAs work. Single objective GAs recombine good features of different solutions from the current population. As the probability of reproduction depends on objective function value, the population is expected to contain relatively good solutions. This works if good solutions have some similarities in the decision space, i.e. if some features appear often in good solutions. In the case of standard binary coding such features are called schemas and correspond to some patterns of zeros and ones (see schema theorem, Holland, 1975).

The question arises, if in multiple objective case good solutions, i.e. solutions close to the non-dominated set, have some similarities. In general, there is no reason to expect such similarities. Non-dominated set of a multiple objective problem includes, among others, optima of particular objectives. If the objectives are not positively correlated their optima will, in general, be completely different. The question of similarities between approximately efficient solutions was experimentally studied by Borges and Hansen (1998) for multiple objective travelling salesperson problem (TSP). Significant similarities (measured by the number of common edges) between good solutions of single objective TSP was observed earlier in experiments of Mühlenbein (1991) and Boese et al. (1994). This property of a problem is often called *global convexity*. Borges and Hansen in their study found significant similarities between approximately efficient solutions placed close in the objective space. The greater distance between two solutions in the objective space the less similar these solutions. This suggests that recombination of approximately efficient solutions distant in the objective space is very unlikely to yield good offsprings. Fonseca and Fleming (1993) realize this fact and propose the use of mating restrictions to avoid not promising recombinations, i.e. they propose to ban mating of distant solutions.

Observe, that lack of mating restriction does not deteriorates results of a MOGA, as close solutions are also mated with probability greater than zero. It decreases, however, efficiency of the algorithm, because the time spend on recombining distant solutions is most likely lost.

Note also, that fitness sharing and mating restrictions do in some sense opposite jobs. Fitness sharing penalizes close solutions while mating restrictions ban mating of distant solutions. Thus, parameters of these techniques should be carefully set.

Fitness sharing and mating restrictions are known from single objective GAs, but are not standard techniques. The fact that they are required in Pareto based MOGAs contradicts in

our opinion the statement that GAs are especially well suited for multiple objective optimization.

In our opinion the main disadvantage of Pareto ranking based MOGAs is, however, at the very heart of the algorithms - the lack of the objective function to be optimized. Taking into account recent trends in combinatorial optimization, Pareto ranking based fitness assignment may be considered a step back. As was mentioned before GLS methods often prove to be the most efficient for single objective combinatorial problems. Such methods combine genetic algorithms with locals search. Offsprings are starting points for local optimization of the objective function. In other words such methods can be treated as GAs working on the set of local optima of the objective function. Methods of this kind are also called hybrid genetic algorithms or memetic algorithms. There are significant similarities between GLS and scatter search (Glover, 1977; see also Glover, 1995, for discussion of the similarities).

Pareto ranking is not well suited for local optimization. Change of the rank of a given solution may require significant changes of the objectives, so, many local moves will not influence the rank. In the case of solutions having rank 1, no local improvement is possible, even, if the solutions are far from the non-dominated set. Furthermore, evaluation of local moves depends on other solutions in the population. It may significantly increase computational time needed for local optimization.

3.2 Ishibuchi's and Murata's multiple objective genetic local search

Murata et al. (1996) proposed a multi-objective genetic algorithm that is not based on Pareto ranking. Ishibuchi and Murata (1998) developed on this basis a multiple objective genetic local search algorithm. Their algorithm uses single genetic population of constant size. At each iteration a weighted linear utility function is drawn at random. Two solutions (parents) are selected randomly from the population with probability that promotes solutions good on the current utility function. The two solutions are recombined and local search is applied to the offspring. Ishibuchi and Murata (1998) report promising results on multiple objective flowshop scheduling problems.

Ishibuchi and Murata do not use explicit mating restrictions. Their parents selection procedure increases probability of recombining solutions being good on an utility function but still all solutions are allowed to recombine.

3.3 MOSA-like multiple objective genetic local search

Ulungu et al. (1995) proposed a methods based on simulated annealing called MOSA. In MOSA a number of weighted linear functions are selected. Each of them is optimized sequentially or in parallel by independent simulated annealing process. The outcome of this algorithm are not only best solutions obtained for each of the optimized functions, but all potentially efficient solutions found during the optimization. The idea of the method is very general and can easily be used with any other metaheuristic, e.g. GLS. Such algorithm will be called MOSA-like MOGLS. The fact that the algorithm uses a number separate populations may be interpreted as a form of strict mating restrictions.

4 Background of the proposed algorithm

4.1 The goal of multiple objective metaheuristics

Multiple objective metaheuristics should generate good approximations to the non-dominated set in a short time. What does it mean, however, “a good approximation”? Most authors of

multiple objective metaheuristics assumes that answer to this question is intuitively clear and needs no further explanation or they describe good approximation as close to the real non-dominated set and well dispersed over this set.

Hansen and Jaszkiewicz (1998) discuss the issue of evaluating approximations to the non-dominated sets in more details. The main idea of their framework is described below.

In the case of a MOCO problem, the overall goal of the decision-maker (DM) is to select the single solution, which is most consistent with his or her preferences, the so-called best compromise. For example, if the DM's preferences correspond to a utility function, the best compromise is the solution that maximizes utility. Generating an approximation to the non-dominated set is only the first phase in solving the problem. In the second phase, the DM selects the best compromise solution from the approximation.

Generation of approximations to the non-dominated set makes sense if the DM's preferences are unknown or known only partially. In this case the approximation should contain solutions good for all possible DM's preferences. It is often assumed that DM's preferences may be modeled with a utility function compatible with the dominance relation. In this case, good approximation should give high utility for any possible utility function of this kind. Thus, the goal of multiple objective metaheuristics may be formulated as simultaneous optimization of all possible utility functions.

Note, that each utility function compatible with the dominance relation has a global optimum at the non-dominated set. Thus, approximation containing all efficient solutions optimizes all utility functions of this kind.

The way in which one states the goal of multiple objective metaheuristics influence the way the methods are designed. For example, authors of Pareto ranking based MOGAs made significant efforts to avoid optimization of any function. Our method is clearly motivated by the by the idea of simultaneous optimization of all utility functions.

4.2 Intrinsic parallelism of metaheuristic optimization of utility functions

The set of all utility function compatible with the dominance relation is obviously intractable. Thus, we will use the sets of all weighted Tchebycheff utility functions or all weighted linear utility functions with normalized weight vectors. Of course, these sets still contain infinite numbers of functions.

Consider optimization of a single weighted linear utility function for a bi-objective TSP with 50 cities by a genetic algorithm. The weight vector Λ is equal to [0.5, 0.5]. In figure 1 we present images of randomly selected 10% of all the solutions generated by the algorithm. Observe that the solutions are placed in relatively wide region in the objective space. If the weights are only slightly changed the picture would be very similar. The GA will generate solutions in more or less the same region in the objective space. This suggests that optimizing a single utility function we can find good solutions for other similar utility function.

To study it more precisely, we performed the following experiment for a bi-objective TSP with 100 cities. We optimized with a GLS algorithm (details of this algorithm may be found in section 4.3) a number of weighted linear utility functions. The weight of the first objective λ_1 was set equal to 0.5, 0.505, 0.51,...,0.65. The weight of the second objective was equal 1-λ_1. Thus, 31 different objective functions were optimized. Each of the functions was optimized 1000 times. The utility function with weight vector [0.5,0.5] was treated as the basic function. We estimated probability that optimizing a given function with weight vector [λ_1,1-λ_1] we find a better solution of the basic function than that found by optimizing this

basic function (we excluded cases when the same values were found). If exactly the same basic function is optimized by the same algorithm the probability would be equal to 0.5. The results are presented in figure 2. We can see that when we optimize a function with weight vector close to [0.5,0.5] the probability is very close to 0.5. The probability goes to zero when the difference between weight vectors increases. The results suggest that it is virtually irrelevant if we optimize weighted utility function with weight vector [0.5,0.5] or [0.515, 0.485].

The above observations suggest that optimizing with a GLS a single utility functions we always simultaneously optimize other close utility functions. In other words, it is just impossible to optimize only a single utility function with a GLS. We expect that the same holds for other metaheuristics. We call this phenomenon intrinsic parallelism of metaheuristic optimization of utility functions.

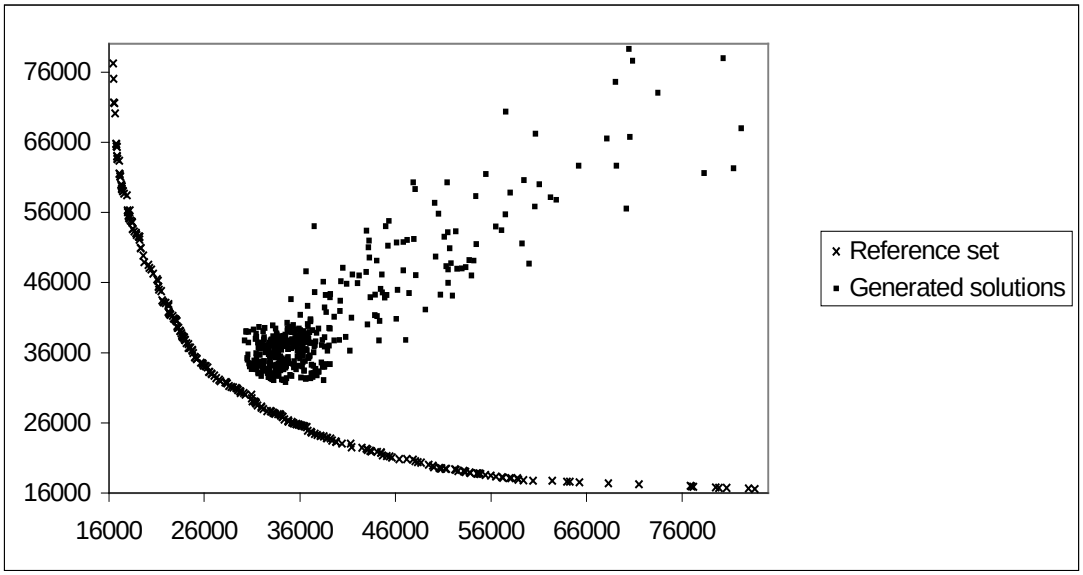

Figure 1. Solution generated by a GA optimizing a single weighted linear utility function for a bi-objective TSP with 50 cities

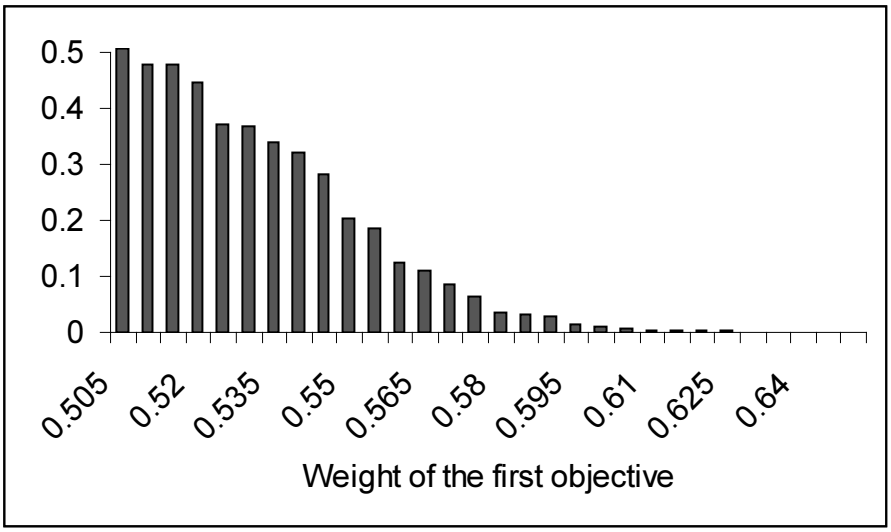

Figure 2. Estimated probability that optimizing linear weighted sum with given weight of the first objective GLS finds better solution of linear weighted sum with weights [0.5, 0.5] than GLS optimizing this the second function

4.3 Basic genetic local search

Alike other multiple objective metaheuristic our MOGLS is based on a single objective algorithm. Below details of this algorithm are given.

Parameters: N – size of the current population, stopping criterion
Initialization:
Current population P:=∅
repeat N times
    Construct randomly a new feasible solution **x**
    Optimize locally the objective function starting from solution **x** obtaining **x**'
    Add **x**' to P
Main loop:
repeat
    Draw at random with uniform probability two solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ from P.
    Recombine $\mathbf{x}_1$ and $\mathbf{x}_2$ obtaining $\mathbf{x}_3$
    Optimize locally the objective function starting from solution $\mathbf{x}_3$ obtaining $\mathbf{x}_3$'
    if $\mathbf{x}_3$' is better than the worst solution in P and different to all solutions in P **then**
        Add $\mathbf{x}_3$' to P and delete from P the worst solution
until the stopping criterion is met

For TSP problems of the size similar to those used in our experiments the population relatively quickly converges to a number of similar local optima, such that no other better local optima can be found in result of recombination and local search. Thus, optimization is stopped if in N successive iterations current population is not changed.

In the above algorithm mutation operator is not explicitly used. The recombination operator used for TSP problems introduces, however, some elements of randomness. In other cases explicit mutation operators may be used.

5 The algorithm of random directions multiple objective genetic local search

Alike in algorithm of Ishibuchi and Murata at each iteration a utility function is randomly selected. Then a temporary population is created composed of best known solutions on the current utility function. The details of the algorithm are given below.

Parameters: N – size of the temporary population, S - number of initial solutions, stopping criterion
Initialization:
The set of potentially efficient solutions PE:=∅
The current set of solutions CS:=∅
repeat S times
    Draw at random a utility function u
    Construct randomly a new feasible solution **x**
    Optimize locally the objective function u starting from solution **x** obtaining **x**'
    Add **x**' to the current set of solutions CS
    Update set PE with **x**'
Main loop:

repeat
Draw at random a utility function u
From CS select N different solutions being the best on utility function u forming temporary population TP
repeat N times
Draw at random with uniform probability two solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ from TP.
Recombine $\mathbf{x}_1$ and $\mathbf{x}_2$ obtaining $\mathbf{x}_3$
Optimize locally the utility function u starting from solution $\mathbf{x}_3$ obtaining $\mathbf{x}_3$'
if $\mathbf{x}$' is better than the worst solution in TP and different to all solutions in TP **then**
Add $\mathbf{x}_3$' to the current set of solutions CS
Add $\mathbf{x}_3$' to TP and delete from TP the worst solution
Update set PE with $\mathbf{x}_3$'
until the stopping criterion is met

The original idea was to keep in the current set of solutions CS all generated solutions. Storing and handling all solutions would, however, be very time and memory consuming. Thus, the following approach is used. Set CS is organized as a queue of size $N \times S$. At each iteration the new generated solution is added to the beginning of the queue if it is better than the worst solution in the temporary population and different to all solutions in the temporary population. If the size of the queue is bigger than $N \times S$ the last solution from the queue is removed. According to our experiments this approach does not introduces any significant deterioration of result with respect to version that uses all solutions.

The way temporary populations are formed may be interpreted as a form of mating restrictions. The solutions that are allowed to recombine must both be good on the same utility function. If the problem has the property of global convexity it also assures that the solutions selected to a temporary population are relatively similar in the decision space.

In the case of RD-MOGLS we did not observe this kind of convergence noticed in GLS (see section 4.3). Even after a large number of iterations temporary populations allow to obtain new good solutions. This happens because random selection of utility functions introduces additional diversification mechanism. Thus, the stopping criterion is defined by maximum number if iterations.

Updating the set of potentially efficient solutions PE with solution $\mathbf{x}$ consists of:

- adding $\mathbf{x}$ to PE if no solution in PE dominates $\mathbf{x}$,
- removing from PE all solutions dominated by $\mathbf{x}$.

Note, that the set of potentially efficient solutions is updated with local optima only. In general, other solutions generated during the local search may also be potentially efficient. This approach allows, however, for significant reduction of computational time. Furthermore, a data structure called quad tree allows for very effective implementation of this step (Finkel and Bentley, 1974; Habenicht, 1982).

In order to set parameter S - number of initial solutions, for a given temporary population size, the following heuristic is proposed. The idea is to find a set of solutions that for any utility function would contain N best solutions of the same average quality as average quality of local optima of this function.

Phase I
Select a number (e.g. 20) utility functions
for each of the selected utility functions

Generate a number (e.g. 10) of local optima of the utility function
Calculate average value of local optima over the generated local optima
Calculate *AV1* - average value of local optima over the selected utility functions

Phase II
The current set of solutions *CS*:=∅
repeat
Draw at random a utility function from the set of all possible functions
Find local optimum of the utility function and add it to *CS*
for each of the utility function selected in *Phase I* find *N* best solutions from *CS*
Calculate *AV2* - average value of *N* best solutions over the selected utility functions
while *AV2* is not worse that *AV1*
The size of *CS* defines parameter *S*

All local optimizations in the above algorithm start from random solutions.

The first phase of the above heuristic is equivalent to generation of a number of starting populations of the single objective GLS described in section 4.3. In the second phase a set of random local optima is obtained that contains *N* best solutions of the same average quality on the selected utility functions. Some results of this heuristic are presented in table 1 The results are averages over 10 randomly generated problems.

Table 1. Estimated numbers of starting solutions for Euclidean TSP problems

	bi-objective Euclidean TSPs with 50 cities	bi-objective Euclidean TSPs with 100 cities	three-objective Euclidean TSPs with 50 cities	three-objective Euclidean TSPs with 100 cities
N	*S*	*S*	*S*	*S*
4	25	38	95	187
8	41	67	184	337
12	63	97	279	533
16	87	133	362	692
32	169	257	713	1367

The algorithm uses either weighted linear or weighted Tchebycheff utility functions, so, drawing at random a function means drawing at random a weight vector form the set of normalized weight vectors. The weight vectors are drawn with uniform probability distribution $p(\Lambda)$, i.e. a distribution for which:

$$\forall \Psi' \subseteq \Psi \quad \int_{\Lambda \in \Psi'} p(\Lambda) d\Lambda \Big/ \int_{\Lambda \in \Psi} p(\Lambda) d\Lambda = V(\Psi') / V(\Psi)$$

where Ψ and Ψ' denote the set of normalized weights and a subset of it, $V(\Psi)$ and $V(\Psi')$ are Euclidean hyper-volumes of Ψ and Ψ', respectively. In other words, the probability of a weight vector belonging to Ψ' is proportional to the hyper-volume of Ψ'.

6 Computational experiments for multiple objective symmetric travelling salesperson problems

6.1 Multiple objective symmetric travelling salesperson problem

Single objective TSP is often used to test single objective metaheuristics. It is defined by a set of cities and cost (distance) of travel between each pair of cities. In symmetric TSP the cost

does not depend on direction of travel between two cities. The goal is to find the lowest cost hamiltonian cycle.

In *J*-objective TSP, *J* different costs are defined between each pair of cities. To construct a *J*-objective symmetric TSP we use *J* different single objective TSP. Thus, *j*-th cost, *j*=1,...,*J*, between a pair of cities comes from *j*-th single objective problem. Individual optima of particular objectives are equal to optima of corresponding single objective problems. In our case, the single objective problems are completely independent, so, also objectives are independent and therefore non-correlated. The same approach was used by Borges and Hansen (1998).

Also following Borges and Hansen (1998) we use some multiple objective problem instances based on TSPLIB library (Reinelt, 1991). For example, problem instance kroAB100 denotes a bi-objective problem with distances corresponding to the first objective taken from kroA100, and distances corresponding to the second objective taken from kroB100. kroABC100 denotes a three objective problem instance with distances taken from kroA100, kroB100 and kroC100 instances. In this way 10 different bi-objective problem instances and 10 three-objective problem instance were created. We used also 10 problem instances with 50 cities based on kroA100-kroE100 instances. They are obtained by using 50 beginning cities from the original instances.

6.2 Quality evaluation

The outcome of tested algorithms are sets of solutions being approximations to the non-dominated set. In order to measure quality of approximations we follow the approach proposed by Hansen and Jaszkiewicz (1998). The quality of approximation A is evaluated by expected value of weighted Tchebycheff utility function over the set of normalized weight vectors Λ:

$$E\left(u_{\infty}^{*}(A,\Lambda)\right)=\int_{u\in U} u_{\infty}^{*}(A,\Lambda)p(\Lambda)d\Lambda\,,$$

where $u_{\infty}^{*}(A,\Lambda)=\max_{\mathbf{z}\in A}\left\{u_{\infty}(\mathbf{z},\mathbf{z}^{*},\Lambda)\right\}$ is the best utility achieved by function $u_{\infty}(\mathbf{z},\mathbf{z}^{*},\Lambda)$ on approximation A.

The utility functions are normalized such that each of them achieve value equal to 1 at (approximation of) ideal point and value equal to 0 at (approximation of) nadir point. In order to estimate the expected value we use numerical integration. Details are given appendix.

6.3 Customization of genetic local search to travelling salesperson problem

This section describes the way both single and multiple objective GLS algorithms were customized to TSP.

The recombination operator we use is similar to the distance preserving crossover introduced by Freisleben and Merz (1996). An offspring is constructed in the following steps:

Step 1. Put in the offspring all edges common among both parents

Step 2. Try to complete the hamiltonian cycle with edges contained in one of the parents. If two or more edges may be placed select the shortest one.

Step 3. Complete the hamiltonian cycle with randomly selected edges.

No mutation operator is used, however, the third step of the recombination operator introduces some randomness.

Local optimization is performed in two phases. In the first phase local search does not take into account edges that were common to both parents. In the second phase all edges are considered. In many cases no improvement is found in the second phase. This two phase approach gives only slightly worse result than a single local search considering all edges, but is significantly faster.

The local search used is based on standard 2-edge exchange neighborhood. While constructing the initial population greedy local search is used. After recombination steepest local search is used. This combination was found to give the best results.

Note, that the local search is relatively simple and leaves space for many improvements. Merz and Freisleben (1997) describe a state of the art single objective GLS for single objective TSP that uses a number of techniques that significantly increase efficiency of local search.

6.4 The choice of the type of utility functions

In the case of discrete problems, weighted Tchebycheff utility functions seem to have an advantage over weighted linear utility functions. Each efficient-solution is global optimum of some weighted Tchebycheff utility functions, while weighted linear utility functions achieve global optima on a subset of efficient solutions only, called *supported efficient solutions*. Because of it, we use Tchebycheff in our quality measure (see section 6.2).

Our implementation of RD-MOGLS for TSP uses, however, weighted linear function. Several reasons for that are discussed below.

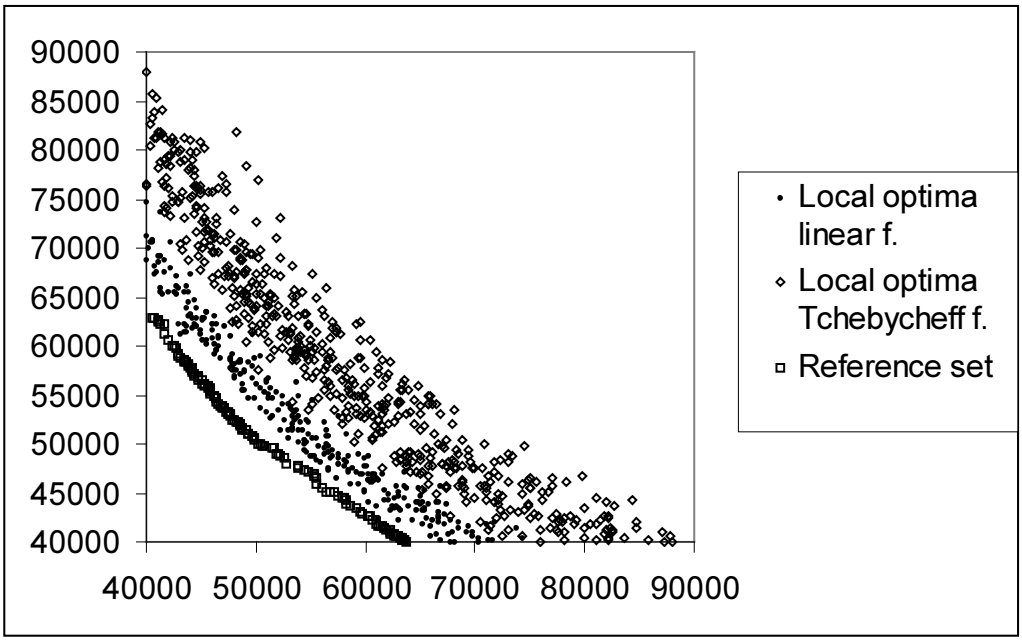

Figure 3. Local optima of weighted linear and weighted Tchebycheff functions with random weights

First of all, weighted Tchebycheff functions are more difficult to optimize than weighted linear functions for multiple objective TSP. Hansen (to appear) observed that both local search and tabu search give better final values of weighted Tchebycheff function for multiple objective TSP, when the search within neighborhood of the current solution is guided by weighted linear functions than when it is guided by weighted Tchebycheff functions. This result is somehow surprising, as it means that in order to achieve good values of an objective function some other function should be used locally. Our observations confirm this phenomenon. For example, figure 3 presents local optima of 1000 randomly selected

weighted Tchebycheff and local optima of 1000 randomly selected weighted linear functions obtained for kroAB100 bi-objective TSP with 100 cities. We present also a reference set, i.e. the best approximation we have obtained. One can see that local optima of weighted linear functions are much closer to the reference set.

Because of special structure of TSP local search is also faster in the case of weighted linear functions, because requires less arithmetical operations.

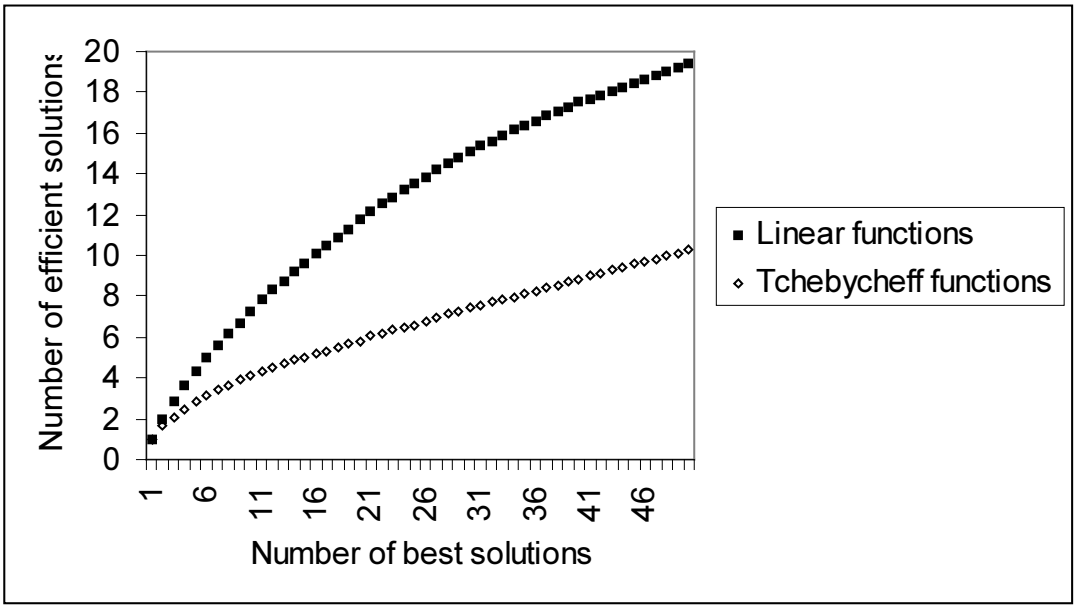

Figure 4. Numbers of efficient solutions among *n* best solutions of linear and weighted Tchebycheff functions

As was mentioned above global optima of weighted Tchebycheff functions contain all efficient solutions. As we work with heuristic algorithm and as the set of potentially efficient solutions is updated with every new generated local optimum, it is more interesting what is the chance of finding efficient solutions among good local optima of a function. In order to test it we used a set of 1000 local optima of randomly selected weighted linear functions (similar results were obtained when local optima of Tchebycheff functions were used). Among the 1000 solutions 130 were potentially efficient. Then, 1000 other randomly selected weighted linear functions and weighted Tchebycheff functions were used. For each of the functions samples of composed of 1, 2,...,50 best solutions were selected from the set of 1000 solutions. For each sample the number of potentially efficient solutions placed in the sample was counted. The average results are presented in figure 4 One can see that on average *n* best solution of a weighted linear function contains more potentially efficient solutions. For example, among 50 best solutions of a weighted linear function on average more than 19 were potentially efficient, while among 50 best solutions of a weighted Tchebycheff function on average less than 11 were potentially efficient. Intuitively, it can be explained by comparison of figures 5 and 6. One can see, that good solutions of weighted linear functions are dispersed over the efficient surface while good solutions of weighted Tchebycheff functions are more concentrated and go inside the feasible set. Another interesting observation made in this experiment, is that good solutions of weighted linear functions, although more dispersed in the objective space, are not more dispersed in the decision space than good solutions of weighted Tchebycheff functions.

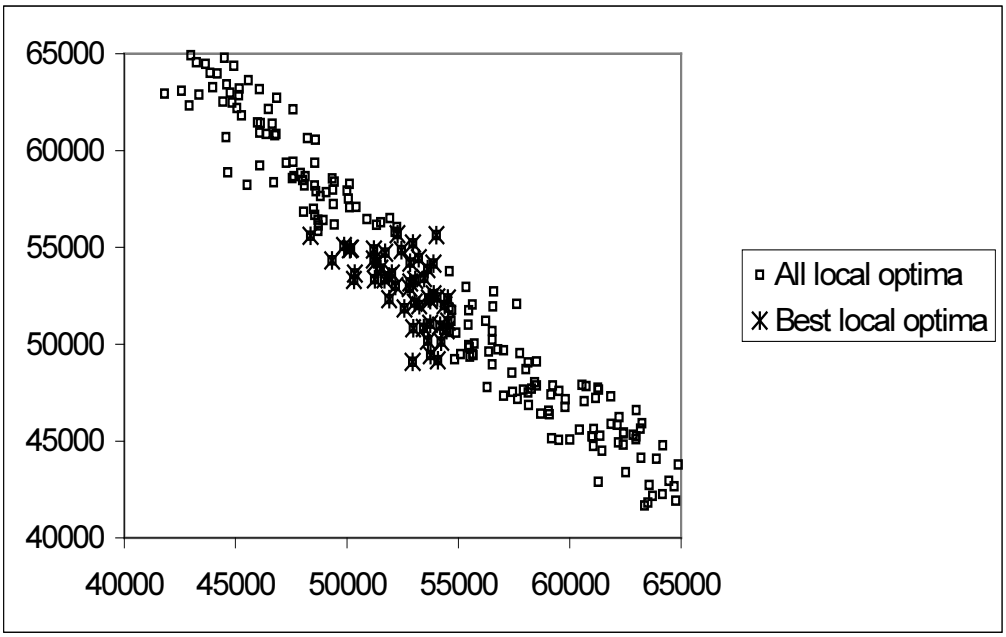

Figure 5. 50 best local optima of a weighted Tchenycheff function

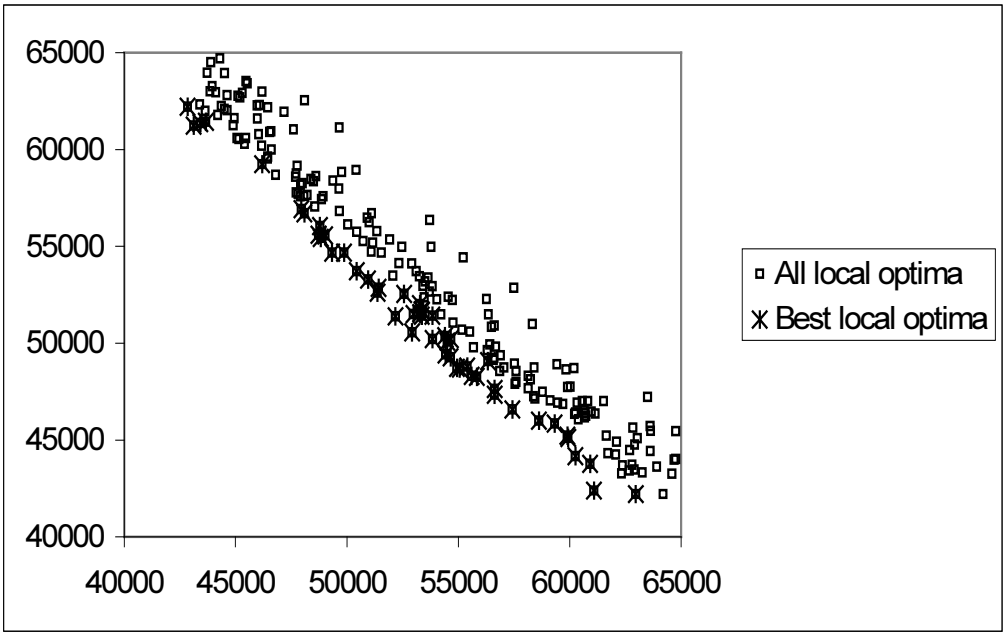

Figure 6. 50 best local optima of a weighted linear function

Summarizing, we have decided to use weighted linear functions in the implementation of RD-MOGLS for multiple objective TSP because such functions are easier to optimize and give higher chance for finding new potentially efficient solutions than weighted Tchebycheff functions. Note, however, that we do not claim that the two kind of functions have the same properties on other problems. The global shape of the non-dominated set in the case of multiple objective TSP is relatively smooth (see e.g. figure 7). Weighted Tchebycheff functions may be better for problems with more irregular non-dominated sets. Similar experiments on other MOCO problems should be performed.

6.5 Results

In the first experiment quality of our single objective GLS was tested. It was compared to multistart local search (MLS) and simulated annealing (SA). In MLS starting solutions were constructed randomly and greedy algorithm was used. SA used exactly the same type of neighborhood moves. Starting temperature was set equal to 100 and final temperature was equal to 1. An intensive experiment was performed in order to find good temperature settings for SA. After each temperature plateau the temperature was multiplied by 0.9. The number of moves on temperature plateau was constant in each run of the procedure.

The three methods were compared on five problem instances coming from TSPLIB library (Reinelt, 1991) – kroA100, kroB100,..., kroE100. The results presented in table 2 are averages of 25 runs of the algorithms (5 runs for each problem). Comparison of heuristics algorithms takes into account two criteria – number of functions evaluations (number of tested local moves) and quality of results. As for each of the 5 problems global optimum is known, the quality is measured by relative excess over the optimum value, i.e. $(f - f_{opt})/ f_{opt}$, where f is the obtained value of objective and f_{opt} is the optimum value.

One can see that GLS by far outperforms the other algorithms. In about 2 000 thousands of functions evaluations it gives better results than achieved by the other algorithms in more than 50 000 thousands of functions evaluations.

In this experiment we report number of iterations not CPU time. If CPU times were used the performance of simulated annealing and multistart local search would be almost the same. This is because, each iteration of simulated annealing requires random number generation and some floating point operations which increases CPU time.

Table 2. Comparison of three single objective algorithms

Multiple start local search (MLS)			Simulated annealing (SA)			Genetic local search (GLS)		
Number of local searches	Thousands of functions evaluations	Average excess	Moves on temperature plateau	Thousands of functions evaluations	Average excess	Population Size	Thousands of functions evaluations	Average excess
10	470	0.0458	10000	440	0.0388	4	753	0.0206
20	940	0.0401	20000	880	0.0310	12	2027	0.0061
40	1903	0.0339	40000	1760	0.0295	20	3293	0.0022
80	3805	0.0298	80000	3520	0.0251	28	4804	0.0019
160	7608	0.0248	160000	7040	0.0168	36	5801	0.0007
320	15241	0.0192	320000	14080	0.0159	44	7365	0.0006
640	30436	0.0156	640000	28160	0.0116	52	7890	0.0006
1280	60882	0.0136	1280000	56320	0.0084	68	10740	0.0005

The proposed RD-MOGLS algorithm was compared to the Ishibuchi's and Murata's MOGLS and with MOSA-like MOGLS. We used our own implementations of Ishibchi's and Murata's algorithm that shares most of the code with implementation of our algorithm. All the algorithms used the same recombination and local search operators. In the case of Ishibuchi's and Murata's algorithm the size of the population was set in the same way as the number of initial solutions in RD-MOGLS. Thus both algorithms were starting by generating the same number of random local optima. In both algorithms the random utility functions were generated in the same way. Also in both algorithms the calculation time was controlled by the

stopping criterion - maximum number of iterations. In the case of MOSA-like MOGLS the weight vectors defining the set of utility functions were generated with the algorithm described in the appendix. The calculation time was controlled by changing the number of utility functions. Optimization of each of the utility functions was continued till the population converged and no further improvement could be achieved as described in section 4.3. In the case of our algorithm the size of the temporary population was equal to 16 for bi-objective instances and equal to 12 for three-objective instances. In the same way the size of the populations were set in MOSA-like MOGLS.

Results of the experiments are presented in tables 3 to 5. For each problem size 10 different instances described in section 6.1 were used. On each instance a single run of each method was performed. The presented results are averages over the ten runs. As local search uses majority of CPU time the number of function evaluations is the main factor influencing calculations time. For example, on PC with Pentium 233 MHz about 1000000 function evaluations are performed in a second.

It can be observed that Ishibuchi's and Murata's MOGLS is by far least effective algorithm. Only on the smallest bi-objective instances with 50 cities it gives results comparable to other algorithms but requires more functions evaluations. On instances with 100 cities Ishibuchi's and Murata's MOGLS does not achieve quality given by RD-MOGLS even if it performs 4 times more recombinations and 20 times more function evaluations. Note also, that on average Ishibuchi's and Murata's MOGLS performs more function evaluations per recombination than the other algorithms, i.e. on average local search in this algorithm is significantly longer. This happens because in this algorithm all solutions are allowed to recombine. Thus, in general, solutions recombined in Ishibuchi's and Murata's MOGLS are worse on the current objective and less similar than in the other algorithms. In result, local search needs more function evaluations to reach local optimum.

The difference between MOSA-like MOGLS and our algorithm is less significant. In all cases, however, our algorithm outperforms MOSA-like MOGLS, i.e. gives better quality in shorter time. For example, MOSA-like MOGLS needs on average 8460 recombinations and 140315 thousands of function evaluations to achieve average quality equal to 0.90212 bi-objective instances with 100 cities, while our algorithm needs on average 6650 recombinations and 79557 thousands of function evaluations to achieve higher average quality equal to 0.90222. The difference is even more significant of 3-objective instances. Our algorithm on average performs also less function evaluations per recombination than MOSA-like MOGLS.

The quality measure used in the experiment allows comparison of different algorithms but does not give information about absolute quality of the approximations. Unfortunately, the exact non-dominated sets for the problems tested are not known. However, Borges and Hansen (1998) generated a large set of supported efficient solution for kroABC100 instance. From this one can extract supported solutions for kroAB100, kroAC100 and kroBC100 instances. Unfortunately, there is no warranty that the sets contain all supported solutions. We use the supported solutions to graphically present quality of obtained approximations. Figures 7 and 8 present supported solutions of kroAB100 instance and one of the approximations obtained with population of size 16 after 11970 recombinations. The approximation contains solutions very close to the supported ones and many of the supported solutions are contained in the approximation.

Table 3. Comparison of three MOGLS algorithms on bi-objective instances with 50 cities

MOSA-like MOGLS				Ishibuchi's and Murata's MOGLS			RD-MOGLS		
Number of utility functions	Number of recombinations	Thousands of functions evaluations	Average quality	Number of recombinations	Thousands of functions evaluations	Average quality	Number of recombinations	Thousands of functions evaluations	Average quality
8	1078	3868	0.88516	440	5099	0.88549	880	3205	0.88634
12	1424	5306	0.88639	880	9138	0.88642	1760	5105	0.88706
16	2057	7446	0.88693	1320	13146	0.88690	2640	6937	0.88735
20	2627	9362	0.88717	1760	17130	0.88713	3520	8762	0.88751
24	3078	10968	0.88728	2200	21211	0.88731	4400	10582	0.88760
28	3526	12719	0.88740	2640	25304	0.88742	5280	12391	0.88766
32	4102	14780	0.88751	3080	29288	0.88750	6160	14158	0.88769
36	4643	16804	0.88758	3520	33291	0.88756	7040	15942	0.88772
40	5251	18513	0.88759	3960	37314	0.88763	7920	17719	0.88775

Table 4. Comparison of three MOGLS algorithms on bi-objective instances with 100 cities

MOSA-like MOGLS				Ishibuchi's and Murata's MOGLS			RD-MOGLS		
Number of utility functions	Number of recombinations	Thousands of functions evaluations	Average quality	Number of recombinations	Thousands of functions evaluations	Average quality	Number of recombinations	Thousands of functions evaluations	Average quality
8	1564	26006	0.89800	1330	124650	0.89935	1330	28321	0.90079
12	2488	41797	0.90067	2660	236907	0.90006	2660	42641	0.90159
16	3361	55706	0.90142	3990	349997	0.90035	3990	55506	0.90190
20	4217	69547	0.90173	5320	462286	0.90056	5320	67518	0.90209
24	5209	84660	0.90188	6650	574425	0.90069	6650	79557	0.90222
28	5961	98480	0.90199	7980	687129	0.90080	7980	91345	0.90229
32	6788	112550	0.90202	9310	798503	0.90090	9310	103138	0.90236
36	7428	123768	0.90207	10640	910441	0.90098	10640	114743	0.90239
40	8460	140315	0.90212	11970	1022391	0.90105	11970	126138	0.90242

Table 5. Comparison of three MOGLS algorithms on tree-objective instances with 100 cities

MOSA-like MOGLS				Ishibuchi's and Murata's MOGLS			RD-MOGLS		
Number of utility functions	Number of recombinations	Thousands of functions evaluations	Average quality	Number of recombinations	Thousands of functions evaluations	Average quality	Number of recombinations	Thousands of functions evaluations	Average quality
36	5080	93430	0.84754	5380	676425	0.85183	5380	128232	0.85356
78	11582	208822	0.85297	10760	1311525	0.85259	10760	197686	0.85502
136	19881	361206	0.85484	16140	1949205	0.85294	16140	260210	0.85558
210	31117	564525	0.85575	21520	2589548	0.85319	21520	319153	0.85597
300	44271	807819	0.85620	26900	3228486	0.85341	26900	375662	0.85625
406	60355	1095729	0.85650	32280	3863447	0.85356	32280	431314	0.85643
528	78766	1432356	0.85675	37660	4499499	0.85369	37660	486133	0.85658
666	98412	1790407	0.85690	43040	5136402	0.85379	43040	539705	0.85672
820	122317	2223911	0.85704	48420	5776767	0.85390	48420	593616	0.85681

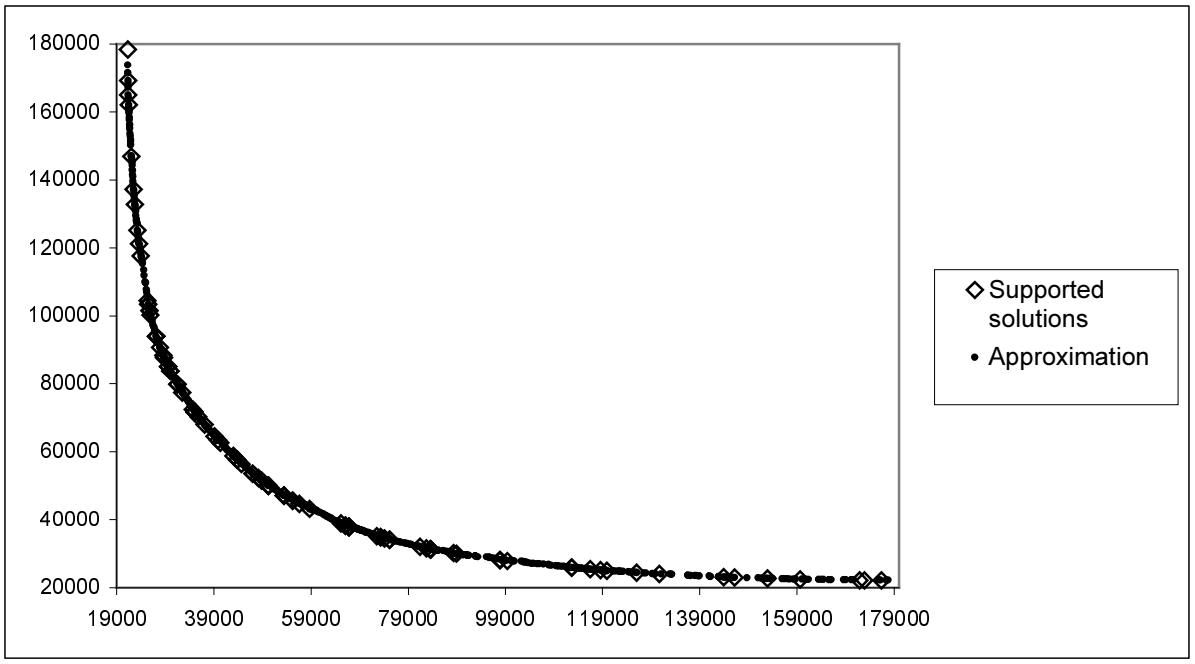

Figure 7. Supported solutions of kroAB100 instance and one of the approximations obtained with population of size 16 after 11970 recombinations

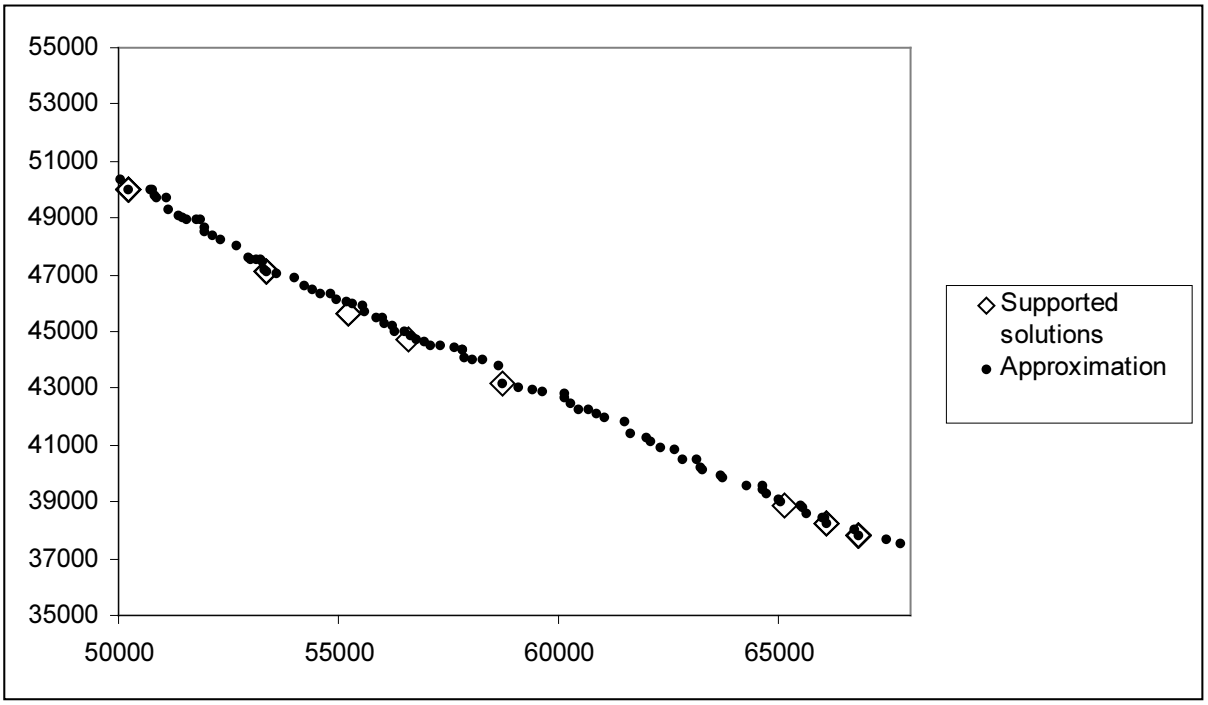

Figure 8. Supported solutions of kroAB100 instance and one of the approximations obtained with population of size 16 after 11970 recombinations – zoomed view

7 Conclusions and directions for further research

A new MOGLS algorithm was described. Results of the presented experiments indicate that the proposed algorithm can generate good approximation to relatively large instance of multiple objective combinatorial problems. The algorithm significantly outperformed other algorithms on TSP instances.

The outcome of the paper is also a method for setting the size of the starting sample. It can be used in other multiple objective methods that start by generation of set of random local optima. In fact, it was applied in our experiments in Ishibuchi's and Murata's MOGLS.

Another result presented in the paper is proposition of a MOGLS method based the idea of MOSA method (Ulungu et al., 1995). The method performs worse than our algorithm but significantly outperforms Ishibuchi's and Murata's MOGLS.

The presented MOGLS algorithm generates approximations of the whole non-dominated set. In many cases, however, some partial information about decision maker's preferences may be known, and the search should be focused on some subregions of the non-dominated set only. In our algorithm it can be achieved by constraining the set of possible weight vectors. It is especially easy if the preferences are expressed in terms of limits on trade-offs between objectives. For example, statement that a unite of the first objective is more important than a unit of the second objective can be easily expressed in this way. In other cases, preferences may be expressed in other forms, e.g. as limits on objective values or as reference/reservation points. In this situation there are two possible solutions, either to transform the preference information to the form of limits on objective trade-offs or to modify the way the utility functions are randomly generated.

In the presented experiments all objectives are of the same type. In practice, one should rather expect problems with objectives of different type. Some of them may be more difficult to optimize than others. In result, some regions of the non-dominated set may require more computational time to achieve good results. The question arises if it is possible to design a method that would automatically discover differences of difficulty in different regions.

The algorithm uses random generation of weights. A deterministic scheme that would assure uniform sampling of the weight space could have positive influence on the quality of results.

In current version of our algorithm we use uniformly distributed normalized weigh vectors. There is, however, no reason to assume that this distribution is appropriate for all problems. In fact one reason to change the distribution was already discussed above, i.e. different difficulty of a problem in some regions of the non-dominated set. Furthermore, the overall shape of the non-dominated set may also influence the best distribution of weights for a given problem.

In general, the normalized weight vectors should be applied to normalized objective values (see section 2), especially if the objectives have significantly different ranges. This requires knowledge about ranges of range equalization factors. An estimation of range equalization factors may be obtained by running a simple optimization (e.g. local search) of particular objectives. Then, the values may be updated during the run of MOGLS on the basis of the current set of potentially efficient solutions.

According to presented experiments weighted linear functions gives better results on TSP that weighted Tchebycheff functions. In general, however, weighted Tchebycheff functions seem to have some advantages. For example, they should be more robust if the shape of the non-dominated set is more complicated. Another possibility would be to use some combined functions being weighted sum of linear and Tchebycheff functions.

Acknowledgement

The work of the author was supported by State Committee for Scientific Research, KBN research grant no. 8T11C 013 13 and by CRIT 2 - Esprit Project no. 20288.

I would like to thank my colleagues Michael Hansen and Pedro Borges for fruitful discussions.

Bibliography

Boese K., Kahng A., Muddu S. (1994). A new adaptive multistart technique for combinatorial global optimization. *Operations Research Letters*, **16**, 101-113.

Borges P.C., Hansen P.H. (1998), A basis for future successes in multiobjective combinatorial optimization. *Technical Report, Department of Mathematical Modelling, Technical University of Denmark*, IMM-REP-1998-8.

Czyzak P., Jaszkiewicz A. (1998), Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7, 34-47.

Finkel R.A. and Bentley J.L. (1974), Quad Trees, A data structure for retrieval on composite keys. *Acta Informatica*, **4**, 1-9.

Fonseca C.M., Fleming P.J. (1995), An overview of evolutionary algorithms in multiobjctive optimization. *Evolutionary Computation*, **3**, 1, 1-16.

Fonseca C.M., Fleming P.J. (1993), Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest (Ed.), *Genetic Algorithms: Proceedings of 5th International Conference*, San Mateo, CA, Morgan Kaufmann, 416-423.

Freisleben B., Merz P. (1996), A genetic local search algorithm for travelling salesman problem. In H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (eds.), *Proceedings of the 4th Conference on Parallel Problem Solving fram Nature- PPSN IV*, 890-900.

Gandibleux, X., Mezdaoui N., Fréville A. (1996). A tabu search procedure to solve multiobjective combinatorial optimization problems, In R. Caballero, R. Steuer (Eds.), *Proceedings volume of MOPGP '96*, , Springer-Verlag.

Glover F. (1977). Heuristic for integer programming using surrogate constraints. *Decision Science*, **8**, 156-166.

Glover F. (1995). Scatter search and star-paths: beyond the genetic metaphor. *OR Spektrum*, **17**, 125-137.

Goldberg D. E., Segrest P. (1987). Finite Markov chain analysis of genetic algorithms. In J.J. Grefenstette (Ed.), *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, Hillsdale, NJ, 1-8.

Gorges-Schleuter M. (1997). On the power of evolutionary optimization at the example of ATSP and large TSP Problems, In *European Conference on Artificial Life `97*, Brighton, U.K.

Habenicht W. (1982), Quad Trees, A datastructure for discrete vector optimization problems. *Lecture Notes in Economics and Mathematical Systems*, **209**, 136-145.

Hansen M. (1997), Tabu search for multiobjective optimization: MOTS, presented at the 13th MCDM conference, Cape Town, South Africa, January 6-10.

Hansen M. (to appear), Experiments on the optimization of the Tchebycheff program, *Journal of Heuristics*.

Hansen P.H, Jaszkiewicz A. (1998). Evaluating quality of approximations to the non-dominated set. *Technical Report, Department of Mathematical Modelling, Technical University of Denmark*, IMM-REP-1998-7.

Holland J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

Horn. J., Nafpliotis N. (1994). A niched Pareto genetic algorithm for multiobjective optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, vol. 1, IEEE, New York, 82-87.

Ishibuchi H. Murata T (1998). Multi-Objective Genetic Local Search Algorithm and Its Application to Flowship Scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, **28**, 3, 392-403.

P. Merz and B. Freisleben, Genetic Local Search for the TSP: New Results, In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, IEEE Press, 159-164, 1997.

Murata T., Ishibuchi H., Tanaka H. (1996). Multi-objective genetic algorithm and its application to flowshop scheduling. *Computers ind. Engng*, **30**, 4, 957-968.

Mühlenbein H. (1991). Evolution in time and space – the parallel genetic algorithm. In G.J.E. Rawlins (ed.), *Foundations of genetic algorithms*, Morgan Kaufman Publishers, 316-337.

Reinelt G. (1991). TSPLIB – a traveling salesman problem library. *ORSA Journal of Computing*, **3**, 4, 376-384.

Schaffer J.D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In: J.J. Grefenstette (ed.), *Genetic Algorithms and Their Applications: Proceedings of the Third International Conference on Genetic Algorithms*, Lawrence Erlbaum, Hillsdale, NJ, 93-100.

Serafini P. (1994). Simulated annealing for multiple objective optimization problems. In: Tzeng G.H., Wang H.F., Wen V.P., Yu P.L. (eds), *Multiple Criteria Decision Making. Expand and Enrich the Domains of Thinking and Application*, Springer Verlag, 283-292.

Srinivas N., Deb K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2, 2, 221-248.

Steuer R.E. (1986). *Multiple Criteria Optimization - Theory, Computation and Application*, Wiley, New York.

Ulungu E.L., Teghem J., Fortemps Ph. (1995). Heuristics for multi-objective combinatorial optimization by simulated annealing. *Multiple Criteria Decision Making: Theory and Applications. Proceedings of the 6th National Conference on Multiple Criteria Decision Making*, Beijing (China), 228-238.

Appendix.

Following proposition of Hansen and Jaszkiewicz (1998) we use the following approach in order to generate uniformly distributed normalized weight vectors. We use all weight vectors in which each individual weight takes on one of the following values: $\{l/k, l = 0,...,k\}$, where k is a sampling parameter defining the number of weight levels. With a combinatorial argument, we notice that this produces $\binom{k+J-1}{J-1}$ weight vectors. For example, for k=3 and J=3, we obtain the following set of 10 vectors: {[0,0,1], [0,1/3,2/3], [0,2/3,1/3], [0,1,0], [1/3,0,2/3], [1/3,1/3,1/3], [1/3,2/3,0], [2/3,0,1/3], [2/3,1/3,0], [1,0,0]}. This approach is used to obtain set of weight vectors defining the set of utility functions optimized in MOSA-like MOGLS, as well as, to calculate estimate values of the approximation's quality measure. All the quality evaluation results reported in the paper were obtained with parameter k equal to 100.

In order to estimate the quality measure, i.e. expected value of weighted Tchebycheff utility function over the set of normalized weight vectors Λ on approximation A (see section 6.2), the

uniformly distributed normalized weight vectors were applied to objective values multipled by range equalization factors (1). We use the ranges over set *N*. This requires the knowledge of ideal and nadir point. In the case of 100 cities instances the ideal point is known, because each individual objective corresponds to a single objective problem coming from TSPLib library. 50-cities instances are relatively easy to solve optimally. As the objectives are independent the elements of the nadir points were estimated on the basis of expected objectives values for random solutions. In the case of instances with 50 cities they were all set equal to 80000, while in the case of instances with 100 cities they were all set equal to 180000.

In order to assure that each of the utility functions achieve value equal to 1 at ideal point and value equal to 0 at approximation of nadir point, a constant 1 is added to each of the functions.