A. Jaszkiewicz (to appear). Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*.

Genetic local search for multi-objective combinatorial optimization

Andrzej Jaszkiewicz

Institute of Computing Science

Poznań University of Technology

ul. Piotrowo 3a, 60-965 Poznań, Poland

Jaszkiewicz@cs.put.poznan.pl

www-idss.cs.put.poznan.pl/~jaszkiewicz

Abstract

The paper presents a new genetic local search algorithm for multi-objective combinatorial optimization. The goal of the algorithm is to generate in a short time a set of approximately efficient solutions that will allow the decision maker to choose a good compromise solution. In each iteration, the algorithm draws at random a utility function and constructs a temporary population composed of a number of best solutions among the prior generated solutions. Then, a pair of solutions selected at random from the temporary population is recombined. Local search procedure is applied to each offspring. Results of the presented experiment indicate that the algorithm outperforms other multi-objective methods based on genetic local search and a Pareto ranking based multi-objective genetic algorithm on travelling salesperson problem.

Keywords Multi-objective combinatorial optimization, metaheuristics, genetic local search

1 Introduction

Combinatorial optimization finds applications in many areas, e.g. in production scheduling, project scheduling, staff scheduling, time tabling, production facilities design, vehicle routing, telecommunication routing, investment planning, location and many others (see e.g. Yu, 1998). Solutions of real-life combinatorial optimization problems usually have to be evaluated taking into account different points of view corresponding to multiple, often conflicting objectives.

The goal of multi-objective optimization is to find the single solution giving the *best compromise* between multiple objectives. Since usually there is no single solution that optimizes simultaneously all the objectives, selection of the best compromise solution requires taking into account preferences of the DM. Under very weak and generally accepted assumptions about the DM's preferences the best compromise solution belongs to the set of, so called, *efficient solutions* (Steuer, 1986, ch. 6.6-6.7). Thus, many multi-objective optimization methods reduce the search space to the set of efficient solutions. Note that this

approach is not valid if the DM searches for a sample of best solutions as the second best and other good solutions do not need to be efficient under the same assumptions about the DM's preferences

Because of computational complexity of many MOCO problems the use of metaheuristcs, e.g. genetic algorithms, simulated annealing or tabu search, seems to be the most promising approach to generation of approximately efficient solutions (Ulungu and Teghem, 1994). Metaheuristics have the advantages of being computationally efficient, general and relatively simple in implementation.

Below we use the term "multi-objective metaheuristic" to characterize methods that generate a set of approximately efficient solutions in a single run. Single objective metaheuristics can also be used in multi-objective context, for example to optimize a scalarizing function. In the latter case, however, a single approximately efficient solution is obtained in each run of the single objective method.

Several authors have proposed multi-objective metaheuristic procedures. 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), Czyżak and Jaszkiewicz (1998), Ulungu et al. (1999) are based on simulated annealing, and the methods of Gandibleux et. al. (1996) and Hansen (1998) are based on tabu search.

In recent years we are observing a growing interest in hybrid single objective metaheuristicc that combine elements of various methods. Typical example is genetic local search (GLS) method combining genetic algorithms with local search. Such methods often outperform other metaheuristics on combinatorial optimization problems (see e.g. Ulder et al., 1991; Murata and Ishibuchi, 1994; Merz, Freisleben, 1997; Gorges-Schleuter, 1997; Galinier and Hao, 1999). Thus, the construction of multi-objective genetic local search methods is a very promising direction for multi-objective combinatorial optimization.

The paper describes a new multi-objective genetic local search method. The goal of the method is to generate effectively a set of approximately efficient solutions that will allow the DM to choose a good compromise solution.

A multi-objective genetic local search method has been proposed by Ishibuchi and Murata (1998). Their method is discussed in section 4.2. Results of the computational experiments reported in section 7 demonstrate that our method performs significantly better in the case of multi-objective travelling salesperson problem (TSP).

The paper is organized in the following way. In the next section, some basic definitions are given. In the third section, the single objective genetic local search metaheuristic is described. Existing multi-objective genetic algorithms are discussed in the fourth section. In the fifth section, the basic single objective genetic local search algorithm is presented. The new multi-objective genetic local search algorithm is described in the sixth section. In the seventh section computational experiments with the proposed algorithms are reported. In the last section conclusions and directions for further research are summarized.

2 Problem statement and basic definitions

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

$$\text{maximize}\{f_1(\mathbf{x}) = z_1, ..., f_J(\mathbf{x}) = z_J\} \qquad \text{(P1)}$$

$$\text{s.t.} \quad \mathbf{x} \in D,$$

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.

A point $\mathbf{z}^1 \in Z$ dominates $\mathbf{z}^2 \in Z$, $\mathbf{z}^1 \succ \mathbf{z}^2$, if $\forall j\ z_j^1 \geq z_j^2$ and $z_j^1 > z_j^2$ for at least one j. Solution $\mathbf{x}^1$ dominates $\mathbf{x}^2$, $\mathbf{x}^1 \succ \mathbf{x}^2$, if the image of $\mathbf{x}^1$ dominates the image of $\mathbf{x}^2$. A solution $\mathbf{x} \in D$ is *efficient* (*Pareto-optimal*) if there is no $\mathbf{x}' \in D$ such that $\mathbf{x}' \succ \mathbf{x}$ Point being image of an efficient solution is called *nondominated*. The set of all efficient solutions is called *efficient set* and denoted by *N*. The image of the efficient space in the objective space is called *nondominated set.*

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

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

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

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

$$\pi_j = \frac{1}{R_j},\ j=1,\ \ldots,\ J \tag{1}$$

where R_j is the (approximate) range of objective z_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.

Weighted Tchebycheff utility functions are defined in the following way:

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

where $\Lambda = [\lambda_1,\ldots,\lambda_j], \forall j\ \lambda_j \geq 0$, is a weight vector. Each utility function of this type has at least one global optimum belonging to the set of efficient solutions. For each efficient solution $\mathbf{x}$ there exists a weighted Tchebycheff utility function such that $\mathbf{x}$ is a global optimum of u (Steuer, 1986, ch. 14.8).

Weighted linear utility functions are defined in the following way:

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

An efficient solution $\mathbf{x}$ is *supported* if there exists a vector of non-negative weights $\Lambda = [\lambda_1,\ldots,\lambda_j]$ such that $\mathbf{x}$ is the unique global optimum of the following problem:

$$\text{maximize } u_1(\mathbf{z}, \Lambda)$$

s.t. $\mathbf{x} \in D$.

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*.

3 Genetic local search metaheuristic

In recent years, the development of hybrid genetic algorithms is one of the most significant trends in the field of metaheuristics. Methods of this kind hybridize recombination operators with local heuristics, e.g. with local search. Other frequently used names are memetic algorithms or genetic local search (GLS). It is quite difficult to track the single origin of GLS. To our knowledge, the first description of GLS was published by Ackley (1987), but similar algorithms were developed probably completely independently by several authors. As it was mentioned in section 1 genetic local search algorithms have proved in recent years to be a very effective class of methods for combinatorial optimization. The methods tend to achieve synergy of recombination operators and local heuristics. In some cases very simple local heuristics are used while other implementations use extensions of local search. For example, Radcliffe and Surry (1994) consider an algorithm in which a single iteration of local search is applied to each offspring while Taillard (1995) applies tabu search to each offspring.

From the genetic algorithms perspective, GLS may be interpreted as a standard genetic/evolutionary algorithm working on a reduced set of solutions, e.g. on a set of local optima. From this point of view, local heuristic is just a part of the recombination operator. The efficiency of GLS may be explained by the fact that in the case of many problems local optima constitute a relatively small part of the search space and the local optima can be achieved in an efficient way.

GLS may be also interpreted as a modification of multiple start local search with random starting solutions. In GLS, starting solutions are constructed in an intelligent way by combination of the properties of other good solutions. If the recombination operator is well designed, starting solutions obtained by recombination should constitute better starting points for local improvement than random solutions. The efficiency of GLS in comparison with multiple start local search can be explained by the fact that local search when started from good starting solutions usually yields better solutions and often requires less time.

4 Existing multi-objective genetic methods

4.1 Pareto ranking based multi-objective genetic algorithms

Clearly, majority of research in the field of multi-objective metaheuristics concentrates on genetic algorithms (see Fonseca and Fleming, 1995, for review). It is often claimed that since GAs work with population (set) of solutions they are especially well suited for multi-objective optimization where the goal is to find a set of approximately efficient solutions (see e.g. Fonseca and Flemming, 1995; Van Veldhuizen, 1999, ch. 2.5). 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, over an interesting region of) the efficient set.

At present probably most often used are MOGAs based on Pareto ranking (compare Van Veldhuizen, 1999, ch. 3.3.2.2 and 3.3.2.3). In classical single objective GAs fitness of a solution depends on its score on the single objective. In Pareto ranking based MOGAs the

fitness depends primarily on a ranking induced by the dominance relation. For the first time this idea was introduced by Goldberg (1988). The general idea of Pareto ranking has been implemented by various authors in slightly different ways. In the MOGA of Fonseca and Fleming (1993) the rank of a given solution is equal to the number of solutions that dominate it.

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.

Pareto ranking alone does not guarantee, however, that the population will disperse over all regions of the efficient set. Fonseca and Fleming (1995) consider fitness landscapes induced by very large, uniformly distributed populations. In the case of Pareto ranking-based selection schemes, all efficient solutions have the highest fitness, i.e. define a plateau of the fitness landscape. This situation is similar to optimization of a 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 multi-objective case, genetic drift means that finite populations tend to converge to small regions of the efficient set. Fonseca and Fleming (1993) and Srinivas and Deb (1994) propose the use of fitness sharing to prevent the 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.

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 multi-objective case. Single objective GAs construct new solutions by recombination of properties of two good solutions. The idea is based on (usually implicit) assumption that good solutions have some similarities in the decision space, i.e. that 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). In multi-objective case, in general, there is no reason to expect such similarities even if they are observed in corresponding single objective problems (compare the study of Borges and Hansen, 1998). Efficient set of a multi-objective problem includes, among others, optima of particular objectives. If the objectives are not positively correlated their optima will be, in general, completely different. This suggests that recombination of approximately efficient solutions distant in the objective space is very unlikely to yield good offsprings. Fonseca and Fleming (1993) propose the use of mating restrictions to avoid not promising recombinations, i.e. they propose to ban mating of distant solutions.

Notice 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. Note also that the distance measures used in the two techniques are, in general, dependent on scaling of objectives. Thus, in practice, the methods based on Pareto ranking are not independent on monotonic transformations of objective functions.

Note also that Pareto ranking is not well suited for hybridization with local search. Change of the rank of a given solution may require significant changes of the objective values; so, many local moves will not influence the rank. In the case of solutions having rank 1 no local improvement is possible. Furthermore, evaluation of local moves depends on other solutions in the population. In the case of problems, for which the evaluation of local moves is very fast, it may significantly increase running time of local optimization.

4.2 Ishibuchi's and Murata's multi-objective genetic local search

Ishibuchi and Murata (1998) were the first authors to propose a multi-objective genetic local search algorithm. The main idea of the method is to randomly generate a weight vector for each iteration. The weight vector is used in a linear utility function. Each iteration of the method consists of a single recombination and a single local search applied to the offspring. The method uses a standard genetic population with no mating restrictions. The solutions for recombination are selected according to the roulette wheel scheme, with fitness depending on the current utility function. Generation replacement is used. Furthermore, a portion of elite potentially efficient solutions is added to the new generation. Ishibuchi and Murata applied their method to a multi-objective flowshop problem.

4.3 MOSA-like multi-objective genetic local search

Ulungu et al. (1999) proposed a method based on simulated annealing called MOSA. The method uses a number of predefined weight vectors defining a set of weighted linear utility functions. Each of the functions is optimized sequentially or in parallel by an independent simulated annealing process. The outcome of the algorithm are not only the best solutions obtained for each of the optimized functions, but all the potentially efficient solutions generated during the optimization. The idea of the method is very general and can be easily used with any other metaheuristic applied to optimization of particular utility functions, e.g. GLS. Such an algorithm will be called MOSA-like MOGLS.

5 Basic single objective genetic local search algorithm

Similarly to other multi-objective metaheuristics our MOGLS is based on a single objective algorithm. The details of this algorithm are given in figure 1. The algorithm assumes complete elitism, i.e. the current population is always composed of a sample of best known solutions.

Parameters: K – size of the current population, stopping criterion
Initialization:
Current population P:=∅
repeat K 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 in the decision space from all the solutions in P **then**
        Add $\mathbf{x}_3$' to P and delete from P the worst solution
until the stopping criterion is met

Figure 1. Algorithm of the basic single objective genetic local search

For the TSP instances of the size similar to those used in our experiment the population relatively quickly converges to a number of close local optima, such that no other better local optima can be found in result of recombination and local search. In the experiments described

in section 7.5 the optimization was stopped if in K successive iterations current population was not changed. This value was selected experimentally. It was observed that population that was not changed in K iterations gives little chance for further improvements. The size of the current population K is the main parameter controlling the calculation time. In general, the larger K the larger CPU time and the better quality of results (see section 7.5).

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

6 The algorithm of multi-objective genetic local search

6.1 Main algorithm

The goal of multi-objective metaheuristics is to generate a set of approximately efficient solutions being a good approximation of the whole set of efficient solutions. Of course, the best possible approximation is set N itself. As it was mentioned in section 2, all weighted linear and all weighted Tchebycheff utility functions achieve optima at efficient solutions. Thus, finding all the efficient solutions is equivalent to finding the optima of all weighted Tchebycheff and all weighted linear utility functions. Hence, we reformulate the goal of multi-objective metaheuristics as simultaneous optimization of all weighted Tchebycheff or all weighted linear utility functions. The term "optimization" in the previous sentence is understood as a tendency of the algorithm to improve values of all the utility functions.

Our MOGLS implements the idea of simultaneous optimization of all weighted Tchebycheff or all weighted linear utility functions by random choice of the utility function optimized in each iteration. In other words, in each iteration, MOGLS tries to improve the value of a randomly selected utility function. A single iteration of MOGLS consists of a single recombination of a pair of solutions. The offspring is then used as a starting point for local search.

The general idea of the proposed algorithm is similar to that used by Ishibuchi and Murata (1998). The main difference is in the way the solutions are selected for recombination. Consider the algorithm of single objective GLS presented in section 5 that is a basis for the proposed MOGLS. In the single objective GLS two parents are drawn at random from the population of K solutions being the best known solutions on the single objective function. Analogously, in the proposed multi-objective genetic local search the parents are selected from the temporary population composed of K solutions being the best known solutions on the temporary utility function used in the current iteration. Of course, in each iteration, the temporary population is, in general, different.

In order to draw at random the utility function in our MOGLS algorithm, a normalized weight vector is drawn at random by the algorithm presented in figure 2. The algorithm uniformly samples the set of normalized weight vectors.

$$\lambda_1 = 1 - \sqrt[J-1]{rand()}$$

...

$$\lambda_j = \left(1 - \sum_{l=1}^{j-1} \lambda_l\right)\left(1 - \sqrt[J-1-j]{rand()}\right)$$

...

$$\lambda_J = 1 - \sum_{l=1}^{J-1} \lambda_l$$

Figure 2. Algorithm for generation of random normalized weight vectors. Function *rand*() returns a random value from the range <0,1> with uniform probability

The details of the MOGLS algorithm are given in figure 3. Figure 4 graphically illustrates the work of the method in a single iteration.

Parameters: K – 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 K different solutions being the best on utility function u forming temporary population TP
    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 $\mathbf{x}_3$' is better than the worst solution in TP and different in the decision space from all the solutions in TP **then**
        Add $\mathbf{x}_3$' to the current set of solutions CS
    Update set PE with $\mathbf{x}_3$'
until the stopping criterion is met

Figure 3. Algorithm of the multi-objective genetic local search

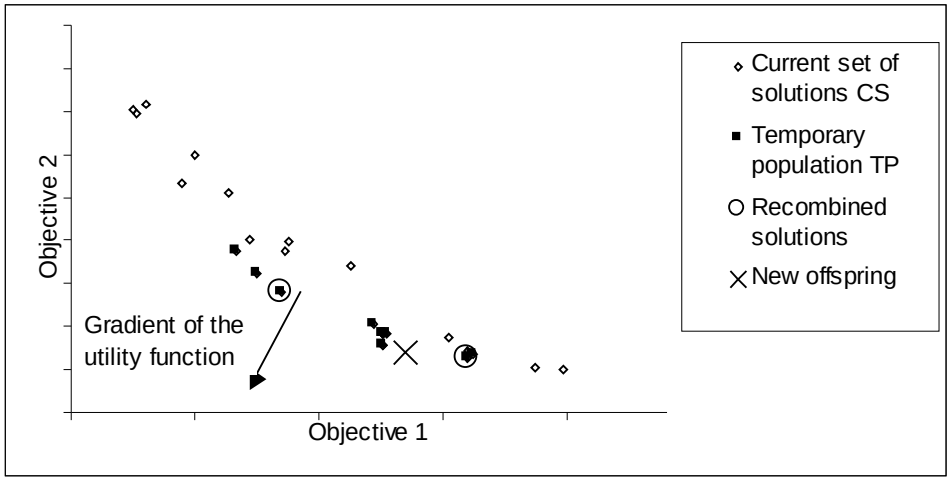


Figure 4. Graphical illustration of a single iteration of the multi-objective genetic local search

Our original idea was to store all the generated solutions in the current set of solutions *CS* from which the temporary populations are selected. Storing and handling all the solutions would, however, be very time and memory consuming. Thus, set *CS* is organized as a queue of size $K \times S$, where S is the number of initial solutions. In each iteration, the newly generated solution is added to the beginning of the queue if it is better than the worst solution in the temporary population and different in the decision space from all solutions in the temporary population. If the size of the queue is bigger than $K \times S$ then the last solution from the queue is removed. The size $K \times S$ was established experimentally. We have observed that such a size of *CS* results in no significant deterioration of the results with respect to the version of our algorithm that stores all generated solutions.

The way the temporary populations are built may be interpreted as a form of mating restrictions. The algorithm recombines only those solutions that are good on the same utility function.

In the case of our method we did not notice this kind of convergence that has been observed in the single objective GLS (see section 5). Even after a large number of iterations recombination and local search allow to obtain new efficient solutions. This happens because random selection of utility functions introduces additional diversification mechanism. Thus, the stopping criterion is defined by the maximum number if iterations.

Updating the set of potentially efficient solutions *PE* with solution **x** consists of:

- adding **x** to *PE* if no solution in *PE* dominates **x**,
- removing from *PE* all the solutions dominated by **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 updating of *PE* (Finkel and Bentley, 1974; Habenicht, 1982).

6.2 Setting the number of initial solutions

The number of initial solutions S is an additional parameter of the method. Its influence on the performance of the algorithm is yet to be tested. Below we propose an approach that allows to

stop generating the initial solutions when the average quality of K best solutions in CS over all utility functions is the same as the average quality of local optima of these functions. In other words, the method assures that on average the quality of K best solutions on a utility function will be the same as the quality of the staring population generated by the root single objective algorithm presented in section 5 applied to optimization of this utility function.

Let $\mathbf{x} \in CS$ be an initial solution obtained by the local optimization of utility function $u_{\mathbf{x}}$. Note that $\mathbf{x}$ does not need to be the best solution on $u_{\mathbf{x}}$ in the current set of solutions CS. Let $B(K, CS, \mathbf{x}, u_{\mathbf{x}}) \subseteq CS$ be the set of K best solutions of function $u_{\mathbf{x}}$ different from $\mathbf{x}$. Let $\bar{u}_{\mathbf{x}}(B(K, CS, \mathbf{x}, u_{\mathbf{x}}))$ be the average value of $u_{\mathbf{x}}$ in $B(K, CS, \mathbf{x}, u_{\mathbf{x}})$, i.e.:

$$\bar{u}_{\mathbf{x}}(B(K, CS, \mathbf{x}, u_{\mathbf{x}})) = \frac{\sum_{\mathbf{y} \in B(K, CS, \mathbf{x}, u_{\mathbf{x}})} u_{\mathbf{x}}(\mathbf{y})}{K}.$$

We propose to stop the generation of initial solutions when the following condition is met:

$$\frac{1}{|CS|} \sum_{\mathbf{x} \in S} \left(\bar{u}_{\mathbf{x}}(B(K, CS, \mathbf{x}, u_{\mathbf{x}})) - u_{\mathbf{x}}(\mathbf{x}) \right) \geq 0.$$

Of course, the above condition could only be tested if $|CS| \geq K + 1$.

Table 1 presents exemplary sizes of the initial sets of solutions obtained with the above approach. The results are averages over 50 results - 5 runs for 10 instances of each size (see section 7.1). In addition standard deviations are presented. Note that the variance of the values is relatively low.

Table 1. Numbers of starting solutions for multi-objective TSP instances. Standard deviations are given in brackets.

	Bi-objective TSP instances with 50 cities	Three-objective TSP instances with 50 cities	Bi-objective TSP instances with 100 cities	Three-objective TSP instances with 100 cities
K	S	S	S	S
4	33.6 (3.13)	115.52 (8.07)	43.52 (2.87)	198.96 (11.84)
8	57.28 (3.74)	203.84 (14.20)	75.04 (5.08)	349.76 (18.57)
16	108.48 (6.70)	381.12 (20.87)	142.4 (6.66)	662.4 (31.17)
32	202.88 (15.31)	733.44 (35.90)	268.8 (15.83)	1290.88 (41.70)

7 Computational experiment on multi-objective symmetric travelling salesperson problem

7.1 Multi-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 a cost (distance) of travel between each pair of cities. In symmetric TSP the cost does not depend on the direction of travel between two cities. The goal is to find the lowest cost hamiltonian cycle.

In J-objective TSP, J different cost factors are defined between each pair of cities. In practical applications the cost factors may for example correspond to cost, length, travel time or tourist attractiveness. In our case, J-objective symmetric TSP instances are constructed from J different single objective TSP instances having the same number of cities. Thus, j-th cost factor, j=1,...,J, between a pair of cities comes from j-th single objective instance. Individual

optima of particular objectives are equal to optima of corresponding single objective instances. In our case, the single objective instances 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 multi-objective TSP instances based on the TSPLIB library (Reinelt, 1991). For example, problem instance kroAB100 denotes a bi-objective instance with cost factors corresponding to the first objective taken from kroA100, and cost factors corresponding to the second objective taken from kroB100. kroABC100 denotes a three objective instance with cost factors taken from kroA100, kroB100 and kroC100 instances. In this way 10 different bi-objective instances and 10 three-objective instance were created. We used also instances with 50 leading cities taken from kroA100-kroE100 instances.

7.2 Quality evaluation

Most of the quality measures used to evaluate results of multi-objective metaheuristics assume the knowledge of the exact set of nondominated points (Van Veldhuizen, 1999, ch. 6.3.4). In the case of our TSP instances, however, the sets of all nondominated points are not known.

In order to measure the quality of solutions generated by the tested algorithms we follow the approach proposed by Hansen and Jaszkiewicz (1998). The quality of a set of approximately efficient solutions A is evaluated by the expected value of weighted Tchebycheff utility function over the set of normalized weight vectors:

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

where $\Psi = \left\{\Lambda \in \Re^J \mid \sum_{j=1}^{J} \lambda_j = 1 \text{ and } \lambda_j \geq 0, j = 1,..,J\right\}$ is the set of normalized weight vectors, $p(\Lambda)$ is a probability intensity function, $u_\infty^*(A,\Lambda) = \max_{\mathbf{z}\in A}\left\{u_\infty\left(\mathbf{z},\mathbf{z}^*,\Lambda\right)\right\}$ is the best utility achieved by function $u_\infty\left(\mathbf{z},\mathbf{z}^*,\Lambda\right)$ on approximation A.

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

7.3 Adaptation of genetic local search to travelling salesperson problem

The recombination operator used in this experiment is 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 arcs common to both parents

Step 2. Complete the hamiltonian cycle with randomly selected arcs.

Local optimization is performed in two phases. In the first phase local search does not take into account arcs that were common to both parents. In the second phase all arcs are considered.

The local search uses a standard 2-arcs exchange neighborhood (see figure 5). 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. The greedy local search tests the neighborhood moves in random order and performs first improving move found. The steepest local search tests all neighborhood moves and performs the best improving move. Both versions stop when no improving move is found in the whole neighborhood.

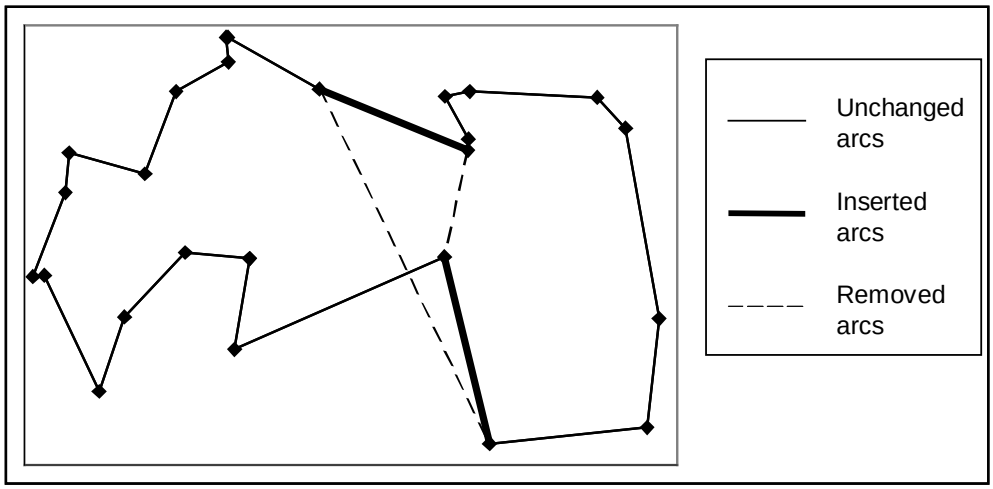


Figure 5. Illustration of the 2-arcs exchange move in 2D Euclidean TSP

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

7.4 Selection 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 a global optimum of a weighted Tchebycheff utility function, while weighted linear utility functions achieve global optima on a subset of efficient solutions only, i.e. supported efficient solutions (see section 2). Because of it, we use Tchebycheff functions in our quality measure (see section 7.2). Our implementation of MOGLS for TSP uses, however, weighted linear function. Several reasons for that are discussed below.

First of all, weighted Tchebycheff functions are more difficult to optimize than weighted linear functions for multi-objective TSP. Hansen (2000) noticed that in the case of multi-objective TSP both local search and tabu search give better final values of weighted Tchebycheff functions, when the search within neighborhood of the current solution is guided by the weighted linear functions than when it is guided by the weighted Tchebycheff functions. Our observations confirm this phenomenon. For example, figure 6 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 instance with 100 cities. We present also a reference set, i.e. the best set of approximately efficient solutions we have obtained. The figure illustrates that local optima of weighted linear functions are much closer to the reference set.

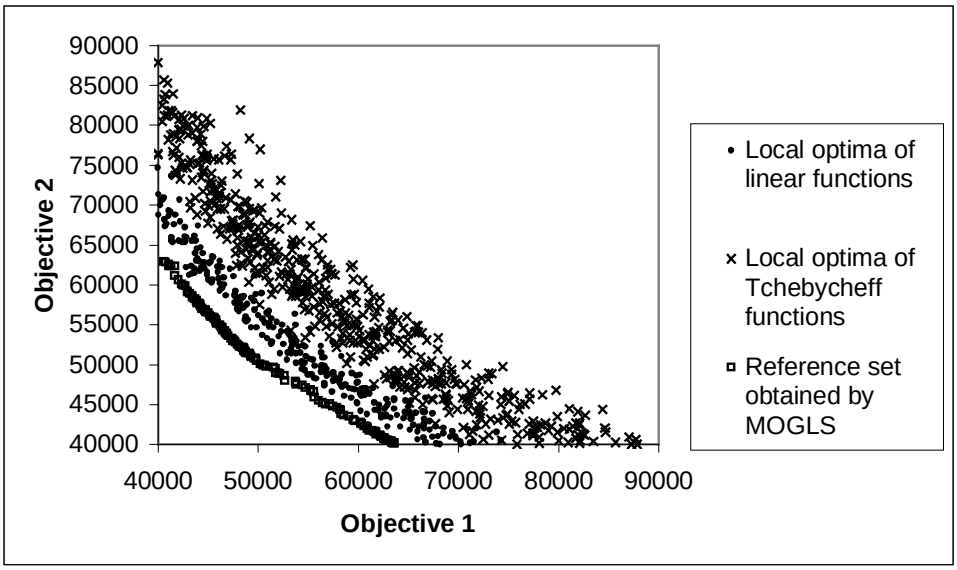


Figure 6. Local optima of weighted linear and weighted Tchebycheff functions with random weights

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

As it was mentioned above global optima of weighted Tchebycheff functions contain all efficient solutions. Since we work with a heuristic algorithm and since 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 composed of 1, 2,...,50 best solutions were selected from the set of 1000 solutions. For each sample the number of potentially efficient solutions contained in the sample was counted. The results are presented in figure 7. On average sample of n best solutions 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 8 and 9. Good solutions of weighted linear functions are dispersed over the set of efficient solutions while good solutions of weighted Tchebycheff functions are more concentrated and placed inside the feasible set.

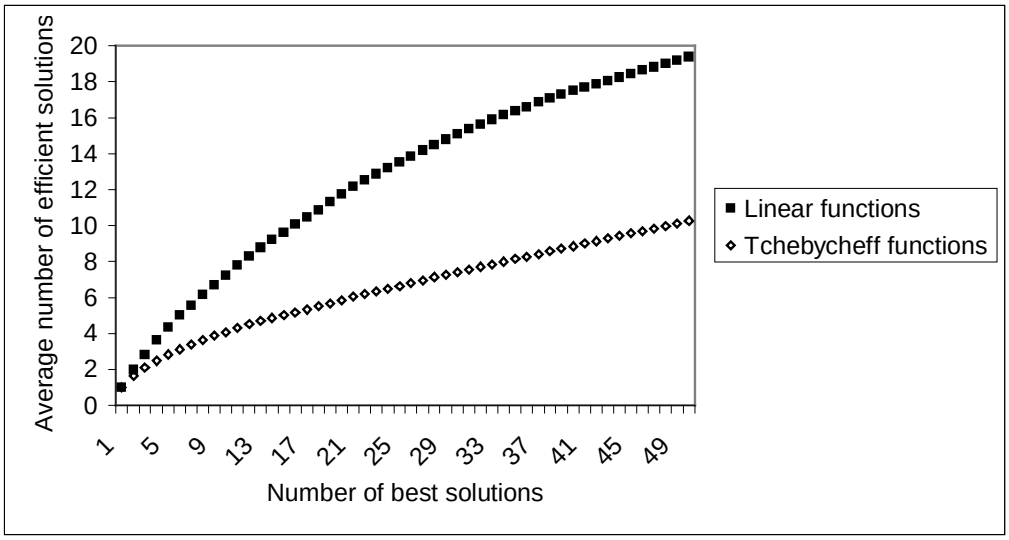


Figure 7. Numbers of potentially efficient solutions among *n* best solutions of linear and weighted Tchebycheff functions

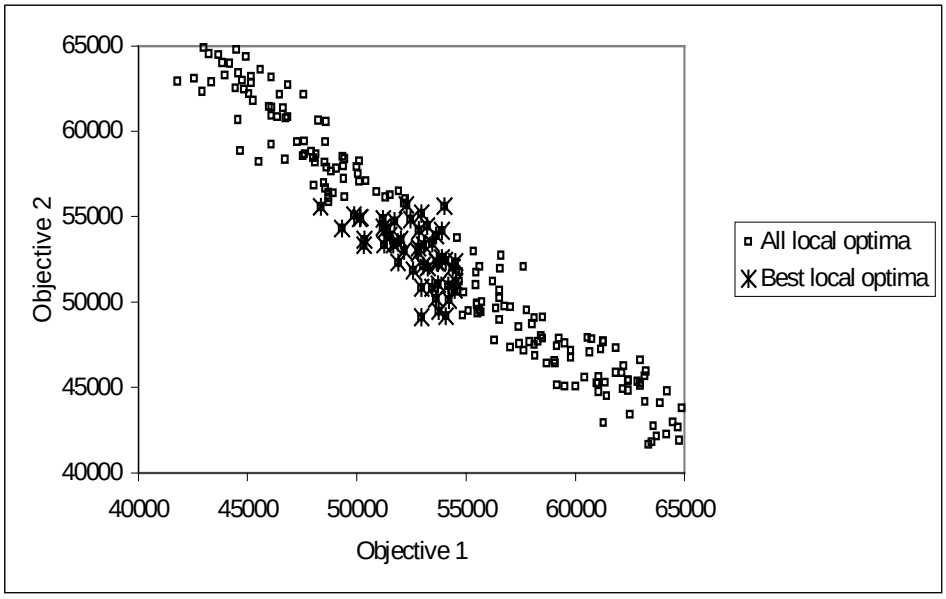


Figure 8. 50 best local optima of a weighted Tchebycheff function

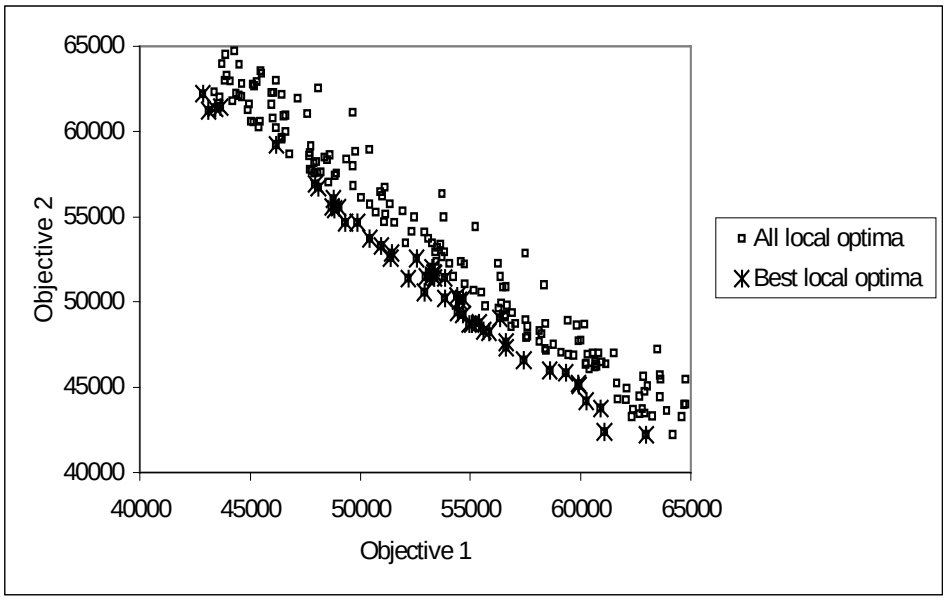


Figure 9. 50 best local optima of a weighted linear function

Summarizing, we have decided to use weighted linear functions in the implementation of MOGLS for multi-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 kinds of functions have the same properties in the case of other problems. The global shape of the nondominated set in the case of multi-objective TSP is relatively smooth (see e.g. figure 12). Weighted Tchebycheff functions may be better for problems with more irregular nondominated sets. Similar experiments on other MOCO problems should be performed.

7.5 Experiment with single objective metaheuristics

The goal of the experiment described in this section was to confirm the quality of single objective genetic local search heuristic in the case of TSP. The algorithm was compared to genetic algorithm (GA), multiple start local search (MLS) and simulated annealing (SA).

GA used population of size 50 and roulette wheel selection with linear scaling. It used the same recombination operator as GLS. In the case of GLS no mutation operator is used. In the case of GA we have observed that the lack of mutation operator results in a very fast convergence of the population. Thus, a mutation operator with probability 0.1 was used. The mutation operator exchanges two randomly selected arcs (see section 7.3).

In MLS starting solutions were constructed randomly and greedy algorithm was used. In the case of SA, the starting temperature was set equal to 100 and the 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 a temperature plateau was constant in each run of the procedure.

The three methods were compared on five instances coming from TSPLIB library (Reinelt, 1991) – kroA100, kroB100,..., kroE100. The results presented in table 2 and in figure 10 are averages of 25 runs of the algorithms (5 runs for each problem). Comparison of heuristic algorithms should take into account at least two main criteria – computational effort and quality of results. Since for each of the 5 instances global optimum is known, the quality is measured by the relative excess over the optimum value, i.e. $(f - f_{opt})/ f_{opt}$, where f is the

obtained value of the objective and f_{opt} is the optimum value. The computational effort is measured by both number of function evaluations (number of tested local moves) and CPU time. The experiments were performed on 350 MHz Pentium PC. Implementations of all the algorithms shared majority of the same code.

GA is clearly the worst algorithm if CPU time is used as the measure of effectiveness. The difference is that high that we decided not to include it in figure 10. Partially it is related to the fact that it performs almost 300 times fewer function evaluations per second than multiple start local search and genetic local search. Allowing GA to perform the same number of functions evaluation as the longest runs of the other algorithm would be extremely time consuming. Note, however, that the longest runs of GA required numbers of function evaluations comparable to the shortest runs of the other algorithms and the quality of solutions generated by GA was still much worse.

GLS by far outperforms the other algorithms. In less than 2 seconds it gives better results than achieved by the other algorithms in more than 45 seconds. It is also the only algorithm that gives high probability of finding the global optimum. Note that the good performance of GLS is an example of a synergy of two mechanisms, i.e. recombination and local search, that used alone perform worse than simulated annealing.

Table 2. Comparison of three single objective algorithms

Multiple start local search				
Number of local searches	CPU time [s]	Thousands of function evaluations	Average excess	Percent of runs with global optimum found
10	0.52	523	0.049287	0
20	1.12	1060	0.040346	0
40	2.09	2096	0.0335	0
80	4.24	4180	0.027339	0
160	8.41	8390	0.021984	0
320	16.93	16849	0.019942	0
640	33.74	33623	0.014479	0
1280	67.40	67169	0.01543	0
Genetic algorithm				
Number of generations	CPU time [s]	Thousands of function evaluations	Average excess	Percent of runs with global optimum found
50	0.71	2.5	0.575154	0
100	1.41	5	0.310289	0
200	2.71	10	0.237564	0
400	5.29	20	0.228006	0
800	10.38	40	0.182186	0
1600	20.66	80	0.178661	0
3200	41.14	160	0.142605	0
12800	165.72	640	0.098903	0
Simulated annealing				
Number of moves on temperature plateau	CPU time [s]	Thousands of function evaluations	Average excess	Percent of runs with global optimum found
10000	0.74	440	0.042622	0
20000	1.50	880	0.032959	0
40000	2.93	1760	0.024034	0
80000	5.91	3520	0.017107	0
160000	11.78	7040	0.01466	0
320000	23.48	14080	0.0146	0
640000	47.07	28160	0.010489	0
Genetic local search				
Population size	CPU time [s]	Thousands of function evaluations	Average excess	Percent of runs with global optimum found
4	0.60	626	0.019031	0
12	1.74	1874	0.003975	12
20	2.95	3187	0.003883	20
28	3.97	4281	0.001781	48
44	5.84	6257	0.001456	48
60	7.67	8208	0.000803	64

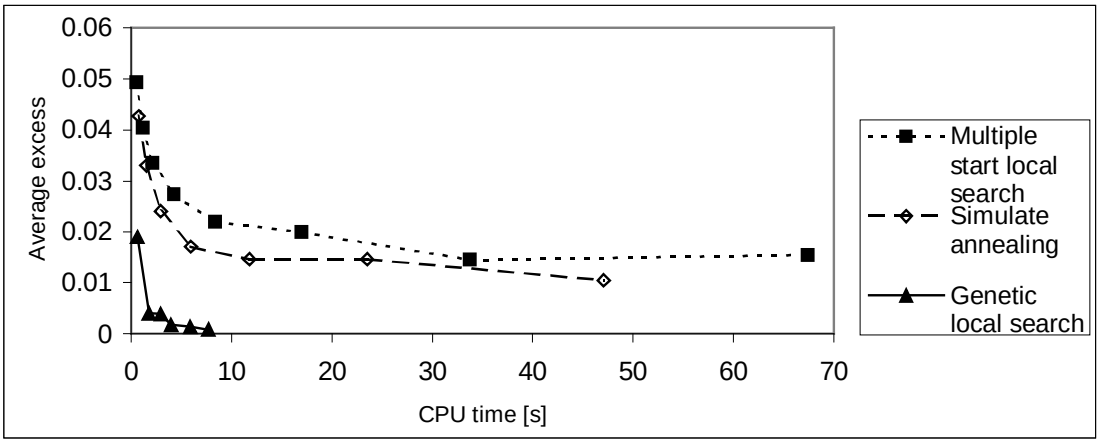


Figure 10. The results of single objective algorithms

7.6 Experiments with multi-objective metaheuristics

The proposed MOGLS algorithm was compared with the Ishibuchi's and Murata's MOGLS (IM MOGLS), with MOSA-like MOGLS and with a Pareto ranking based GA. We used our own implementations of the algorithms that shared most of the code with implementation of our MOGLS. All MOGLS algorithms used the same recombination and local search operators.

In the case of our algorithm the size of the temporary population was equal to 16. In MOSA-like MOGLS the population size was set to the same value. The number of initial solutions of our MOGLS was set according to the values reported in table 1. The CPU time used by the algorithm was controlled by changing the number of recombinations.

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 our MOGLS. Thus both algorithms were starting in the same way by generating the same number of random local optima. Furthermore, in both algorithms the random utility functions were generated in the same way. Moreover, the number of recombinations was the same in the case of the two algorithms. The elite size in IM MOGLS was set equal to 10% of the population size. This size was chosen experimentally using the best choice principle. We have noticed however, that this parameter has relatively small influence on the performance of the method. Ishibuchi and Murata (1998) in the case of flowshop scheduling, reduce the CPU time used by local optimization by restricting the number of neighborhood solutions evaluated in each iteration of local search. Thus, in general, local search does not achieve local optima. We did not use this technique. Note that the state of the art versions of GLS for single objective TSP (see e.g. Merz and Freisleben, 1997) do not use this kind of restrictions in local search. Furthermore, the same kind local search was used in all MOGLS algorithms.

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 CPU time was controlled by changing the number of utility functions. Optimization of each of the utility functions was continued till the stopping criterion described in section 5 was met.

We used Pareto ranking based GA (Pareto GA) proposed by Fonseca and Flemming (1993) witht fitness sharing and no mating restrictions. The recombination operator was the same as in the case of MOGLS algorithms (see section 7.3). Fonseca and Flemming (1993) do not give guidelines for setting the population size. Following (Van Veldhuizen, 1999, ch. 6.3.3.6)

we used population of size 100. The mutation operator was the same as in the case of single objective GA (see section 7.5) and used with probability 0.1.

Results of the experiment are presented in tables 3 to 5 and in figure 11. For each problem size 10 different instances described in section 7.1 were used. On each instance a single run of each method was performed.

Pareto GA is clearly outperformed by all MOGLS algorithms. This could be expected taking into account results of single objective GA reported in the previous section. Alike in the case of single objective GA we decided not to include its results in figure 11.

It can be observed that Ishibuchi's and Murata's MOGLS is by far least effective of the MOGLS 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 the shortest runs of our MOGLS even in 14 times longer time. Note also, that on average Ishibuchi's and Murata's MOGLS performs more functions evaluations per recombination than the other algorithms, i.e. on average local search in this algorithm is significantly longer. This happens because, in general, solutions recombined in IM MOGLS are worse on the current utility function and less similar than in the other algorithms. In result, local search needs more functions evaluations to reach local optimum.

The difference between MOSA-like MOGLS and our algorithm is lower. Figure 11 illustrates, however, that our algorithm outperforms MOSA-like MOGLS, i.e. gives better quality in shorter time. Because the difference between the qualities of results of the two algorithms is low we study statistical significance of the differences. Since the average values of quality reported in tables 3 - 5 are averages of results obtained for different instances their variations cannot be used directly. Thus we analyze variations of differences between results of the two algorithms for the same instances. More precisely we test the statistical hypothesis that the differences of quality of results of our MOGLS and MOSA-like MOGLS corresponding to the parameters settings described in the same rows in tables 3 - 5 are greater than 0. For example we compare results of MOSA-like MOGLS with 10 utility functions with results of our MOGLS with 1080 recombinations on bi-objective instances with 50 cities. The results of the analysis are reported in table 6. The differences are significant at level 0.01 except of the longest runs on bi-objective instances. We expect that on the smaller instances results of both the algorithms converge very close to the set of efficient solutions and thus no significant differences between the algorithms are possible.

The quality measure used in the experiment allows comparison of different algorithms but does not give information about absolute quality of the sets of approximately efficient solutions. Unfortunately, the exact nondominated sets for the problems tested are not known. However, Borges and Hansen (1998) generated a large set of supported efficient solutions for kroABC100 instance. From this set one can extract supported solutions of kroAB100, kroAC100 and kroBC100 instances. Unfortunately, there is no warranty that the sets will contain all supported solutions. We use, however, the supported solutions to graphically illustrate the quality of obtained approximations. Figures 13 and 14 present supported solutions of kroAB100 instance and the approximation generated by our MOGLS with population of size 16 after 7100 recombinations. The approximation contains solutions very close or identical to the supported ones.

The TSP instances used in the experiment and as well as the results of particular methods are available in the Internet at http://www-idss.cs.put.poznan.pl/~jaszkiewicz/motsp/. This page contains also the software used for evaluation of the quality of results.

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

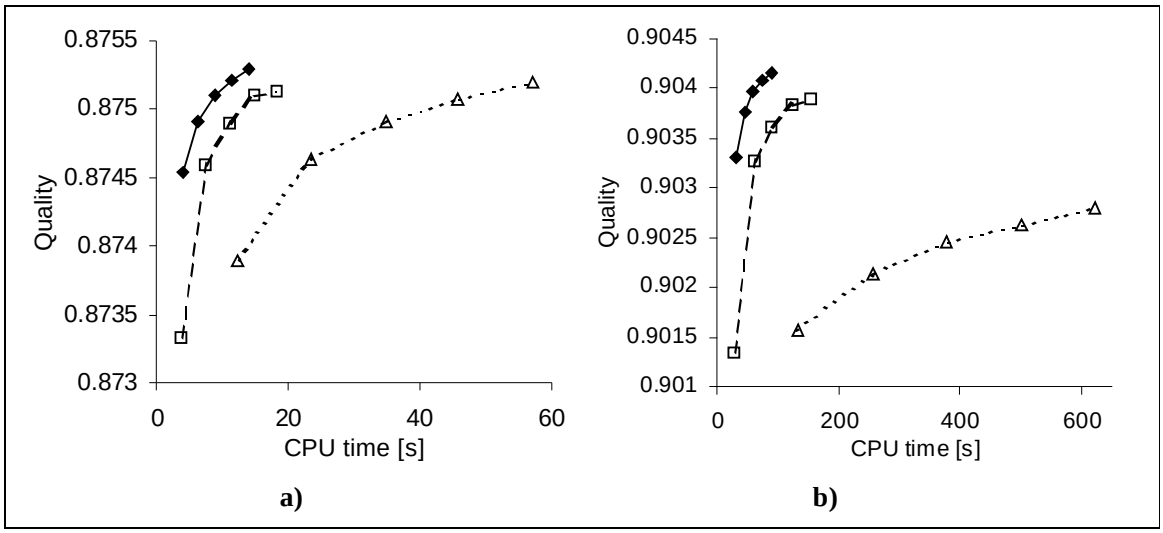
Pareto GA					**IM MOGLS**			
Number of generations	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality
1000	100000	13.7	100	0.77301	1080	12.3	13938	0.873895
2000	200000	27.7	200	0.77550	2160	23.5	26667	0.874639
3000	300000	41.2	300	0.77575	3240	34.8	39503	0.874909
4000	400000	54.9	400	0.77696	4320	45.8	52103	0.875076
5000	500000	68.7	500	0.77715	5400	57	64871	0.875194
MOSA-like MOGLS					**MOGLS**			
Number of utility functions	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality
10	1071	3.6	4014	0.873336	1080	3.9	3815	0.874539
20	2194	7.4	7992	0.874595	2160	6.4	6010	0.874903
30	3154	11	11795	0.874894	3240	8.9	8144	0.875108
40	4402	14.9	16154	0.875098	4320	11.5	10266	0.875217
50	5349	18.4	19842	0.875134	5400	14	12367	0.875297

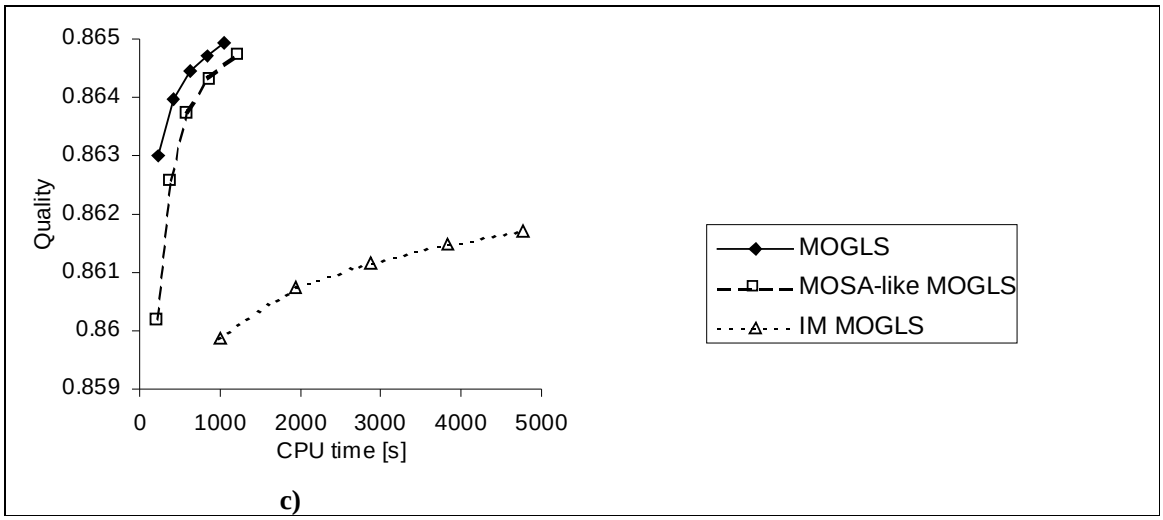
Table 4. Comparison of the algorithms on bi-objective instances with 100 cities

Pareto GA					**IM MOGLS**			
Number of generations	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality
1000	100000	32.9	100	0.75524	1420	133.1	159739	0.901577
2000	200000	65.8	200	0.75639	2840	256	309435	0.902141
3000	300000	99.2	300	0.75837	4260	377.4	457891	0.902456
4000	400000	132.8	400	0.75861	5680	499.9	607413	0.902632
5000	500000	167.1	500	0.75900	7100	621.7	756456	0.902802
MOSA-like MOGLS					**MOGLS**			
Number of utility functions	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality
10	1630	29.4	30350	0.901347	1420	30.1	31478	0.903309
20	3341	60.4	61497	0.903267	2840	44.9	46414	0.903768
30	5051	90.6	94029	0.903619	4260	59.6	60406	0.903972
40	7167	124.3	129379	0.903828	5680	74.2	73863	0.904082
50	8766	154.4	160442	0.903889	7100	88.3	86862	0.904167

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

Pareto GA					**IM MOGLS**			
Number of generations	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality
5000	500000	220.6	500	0.70643	6620	996.6	1029637	0.859863
10000	1000000	487.7	1000	0.71364	13240	1938.8	2015579	0.860726
15000	1500000	767.2	1500	0.71638	19860	2883	2999410	0.86117
20000	2000000	1035.3	2000	0.71684	26480	3830.1	3981994	0.861476
25000	2500000	1300.7	2500	0.71792	33100	4777.3	4966231	0.861703
MOSA-like MOGLS					**MOGLS**			
Number of utility functions	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality	Number of recombinations	CPU time [s]	Thousands of functions evaluations	Average quality
55	10435	211.4	194225	0.860187	6620	223.2	179230	0.862988
91	17279	369.7	323748	0.862565	13240	411	268712	0.863973
136	26102	589.5	488423	0.863735	19860	619.8	347328	0.864439
190	36520	868.5	682463	0.864336	26480	830.8	421326	0.86472
253	48698	1224.1	911775	0.864751	33100	1044.4	492552	0.864923





c)

Figure 11. The results on a) bi-objective instances with 50 cities b) bi-objective instances with 100 cities c) three-objective instances with 100 cities

Table 6. Statistical significance of differences between MOSA-like MOGLS and MOGLS

Number of utility functions in MOSA-like MOGLS	Number of recombinations in MOGLS	Average difference of results over 10 instances	Standard deviation of the difference of results	Statistically significant at level 0.01
Bi-objective instances with 50 cities				
10	1188	0.001515	0.000341	yes
20	2268	0.000414	0.000188	yes
30	3348	0.000179	0.000144	yes
40	4428	8.35E-05	9.32E-05	no
50	5508	4.73E-05	8.12E-05	no
Bi-objective instances with 100 cities				
10	1420	0.00234	0.000349	yes
20	2840	0.000425	0.000227	yes
30	4260	0.000209	0.000155	yes
40	5680	0.000202	0.000143	yes
50	7100	0.000118	0.000152	no
Three-objective instances with 100 cities				
55	6620	0.004387	0.000293	yes
91	13240	0.001993	0.000197	yes
136	19860	0.000943	0.000144	yes
190	26480	0.000486	0.000116	yes
253	33100	0.000275	0.000116	yes

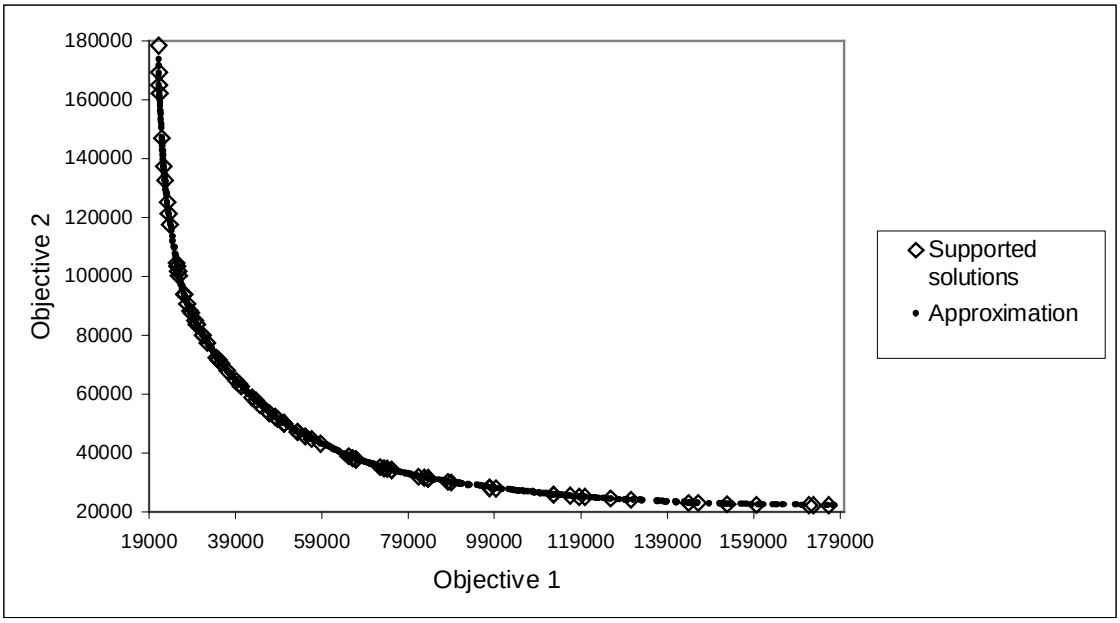


Figure 12. Supported solutions of kroAB100 instance and one of the approximations obtained with population of size 16 after 7100 recombinations

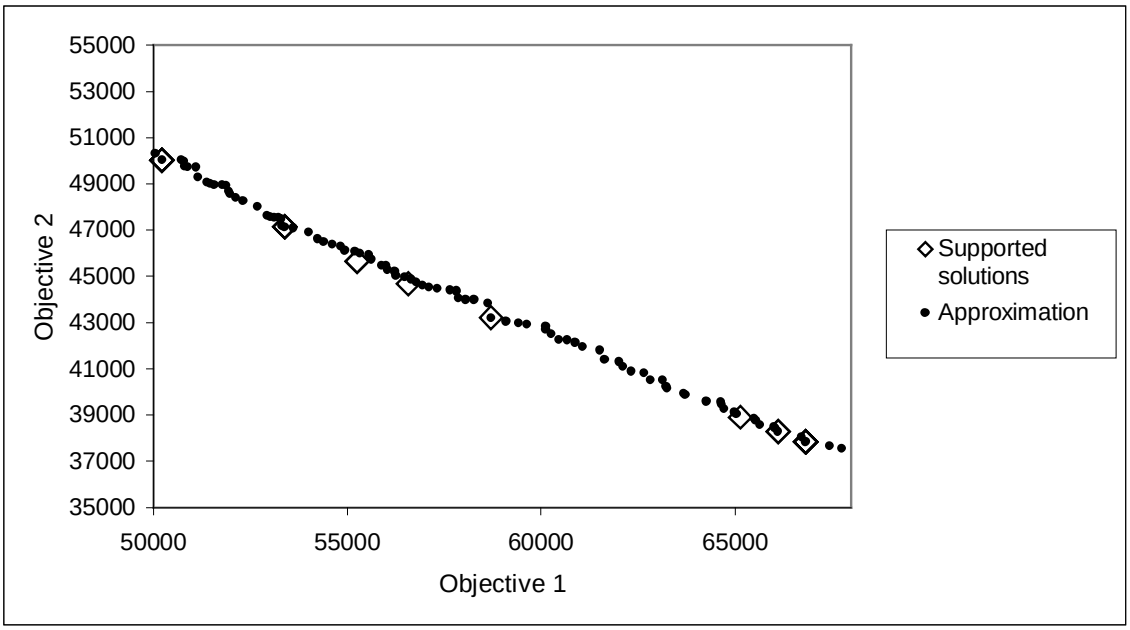


Figure 13. Supported solutions of kroAB100 instance and one of the approximations obtained with population of size 16 after 7100 recombinations – zoomed view

8 Conclusions and directions for further research

A new multi-objective genetic local search (MOGLS) algorithm has been described. Results of the presented experiment indicate that the new algorithm can effectively generate sets of high quality approximately efficient solutions for relatively large instances of multi-objective combinatorial problems. The algorithm significantly outperformed other tested algorithms on TSP instances. The new MOGLS has, however, higher memory requirements than the other algorithms. Thus, the good performance of the algorithm is obtained at the price of memory usage. Taking into account capacities of present computers this fact does not limits significantly applications of our MOGLS.

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

Another result presented in this paper is the proposition of a MOGLS algorithm based the idea of MOSA method (Ulungu et al., 1999). The method performs worse than our algorithm but outperforms Ishibuchi's and Murata's MOGLS.

In this paper, we used a hybridization of recombination operators with local search. The idea of our MOGLS is, however, more general and allows the use of other local heuristics taking into account the value of the current utility function. In fact, we have already used this possibility in the case of multi-objective knapsack problem (Jaszkiewicz, 2000). In that case, we have used two simple greedy repair and insertion heuristics after each recombination.

The presented MOGLS algorithm generates approximately efficient solutions from all regions of the efficient 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 efficient set. In our algorithm it can be achieved by constraining the set of possible weight vectors.

In the presented experiment all objectives are of the same type, i.e. they have the same mathematical definition and differ only by parameter values. In practice, one should rather expect problems with objectives of different mathematical form e.g. sum, min-max, max-min, quadratic, etc. Some of them may be more difficult to optimize than the others. In result, some regions of the efficient set may require more computational time, i.e. more recombinations, to achieve good results. The question arises whether MOGLS could automatically discover differences of difficulty in different regions.

Our MOGLS 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 the current version of our algorithm we use uniformly distributed normalized weight vectors. There is, however, no reason to assume that this distribution is appropriate for all problems. In fact one reason for changing the distribution was already discussed above, i.e. different difficulty of a problem in some regions of the efficient set. Furthermore, the overall shape of the nondominated 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 the range equalization factors. We suggest to generate J first local optima by optimization of particular objectives in order to obtain the first estimation of the range equalization factors. Then, the values may be updated during the run of MOGLS on the basis of the objective ranges in the current set of potentially efficient solutions.

According to the presented experiment weighted linear functions give better results in the case of TSP than weighted Tchebycheff functions. In general, however, weighted Tchebycheff functions should be more robust if the shape of the nondominated set is more complicated.

Multi-objective metaheuristics are applied to both large scale combinatorial optimization problems and non-convex 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 similarity of good solutions exhibited by many combinatorial optimization problems. Other types of problems may require different approaches. Taking into account limitations imposed by the “No free lunch” theorem (Wolpert and Macready, 1997) we believe that it is better to clearly define the class of problems an algorithm is designed for, than to make unjustified statements about its generality.

Acknowledgement

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

This work has been supported by KBN grant No. 8T11F00619.

Bibliography

Ackley D. H. (1987), *A connectionist machine for genetic hillclimbing*. Kluwer Academic Press, Boston.

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.

Czyżak 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. (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.

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

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.

Galinier P., Hao J.-K. (1999), *Hybrid evolutionary algorithms for graph coloring*. Technical Report, Parc Scientifique Georges Besse, Nimes. (To appear in Journal of Combinatorial optimization).

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.

Goldberg D.E. (1988), *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Addison-Wesley.

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.P. (1998), *Metaheuristics for multiple objective combinatorial optimization*, Ph.D. Thesis, IMM-PHS-1998-45, Technical University of Denmark, Lyngby.

Hansen M. (2000), Use of substitute scalarizing functions to guide a local search based heuristic: the case of moTSP. *Journal of Heuristics*, **6**, 3, 419-430.

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., Goldberg D. E. (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 Flowshop Scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, **28**, 3, 392-403.

Jaszkiewicz A. (2000), *On the performance of multiple objective genetic local search on the 0/1 knapsack problem. A comparative experiment*. Research report, Institute of Computing Science, Poznań University of Technology, RA-002/2000, pp.15.

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

Murata T., Ishibuchi H. (1994), Performance evaluation of genetic algorithms for flowshop scheduling problems. *Proc. of the 1st IEEE Int. Conf. Evolutionary Computat.*, 812-817.

Radcliffe N.J., Surry P.D. (1994), Formal memetic algorithms, in: T. Fogarty (Ed.), *Evolutionary Computing: AISB Workshop*, Springer-Verlag, 1994.

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

Schaffer J.D. (1985), Multi-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 multi-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.

Taillard É. D. (1995), Comparison of iterative searches for the quadratic assignment problem, *Location science*, 3, 87-105.

Ulder N.L.J., Aarts E.H.L., Bandelt H.-J. von Laarhoven P.J.M., Pesch E. (1991), Genetic local search algorithms for the travelling salesman problem, in H.-P. Schwefel, R. Männer (Eds.) *Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, 1991, 109-116.

Ulungu E.L. and Teghem J. (1994), Multiobjective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis*, **3**, 83-101.

Ulungu E.L., Teghem J., Fortemps Ph., Tuyttens (1999), MOSA method: a tool for solving multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 8, 221-236.

Van Veldhuizen D.A. (1999), *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

Wolpert D.H., Macready W. G. (1997), No free lunch theorem for optimization. *IEEE Transactions on Evolutionary Computation*, **1** (1), 67-82.

Yu G. (1998), *Industrial Applications of Combinatorial Optimization*, Kluwer Academic Publisher, Boston, 366 pp.

Appendix.

Following the proposition of Hansen and Jaszkiewicz (1998) we use the following approach in order to systematically 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. The set of such weight vectors is denoted by Ψ_s and defined mathematically as:

$$\Psi_s = \{\Lambda = [\lambda_1,...,\lambda_J] \in \Psi \mid \lambda_j \in \{0, 1/k, 2/k,..., k-1/k, 1\}\}.$$

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 quality measure described in section 7.2. While evaluation bi-objective instances the parameter k was equal to 100, and in the case of three-objective instances to 40.

In order to estimate the quality measure (see section 7.2), the uniformly distributed normalized weight vectors were applied to objective values multiplied by range equalization factors (1). We used the ranges over set N. This required 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.