dynamic software & engineering

Lectures

Benchmark of Nature - inspired Optimization Algorithms in fields of single and multiobjective scopes

David Schneider, Daniela Ochsenfahrt, Stephan Blum

presented at the Weimar Optimization and Stochastic Days 2010
Source: www.dynardo.de/en/library

Benchmark of Nature - inspired Optimization Algorithms in fields of single and multiobjective scopes

David Schneider[1]*, Daniela Ochsenfahrt[1], Stephan Blum[1]

[1] DYNARDO – Dynamic Software and Engineering GmbH, Weimar, Germany

Abstract

Within stochastic analyses, nature-inspired optimization algorithms (NOA) imitate natural processes like biological evolution or swarm intelligence. Based on the principle „survival of the fittest“, a population of artificial individuals searches the design space of possible solutions in order to find a better solution of the optimization problem. The usage of these algorithms is recommended for various applications. NOA are very robust against mathematically ill-conditioned problems. Particle swarm optimization algorithm and simple design improvement are added to the algorithms that are already available in optiSLang . Together they compose the new NOA flow. This paper also presents some improvements and enhancements that come with optiSLang 3.2. Thus, within optiSLang 3.2, difficult single or multiobjective optimization tasks can be solved even for large and difficult search spaces. Many practical and theoretical benchmarks have been accomplished to appraise the behaviour of the existing and appearing algorithms and settings. Some of them will be named and listed. To disburden users to choose the right algorithm and settings for optimization tasks we conclude with advises and an easy to use decision tree.

Keywords: optiSLang, Nature-inspired algorithms, PSO, SDI, evolutionary algorithms, singleobjective, multiobjective

1 Introduction

Nature Inspired Optimization Algorithms (NOA) imitate natural processes like biological evolution or swarm intelligence. Based on the principle 'survival of the fittest' a population of artificial individuals searches the design space of possible solutions in order to find a better approximation for the solution of the optimization problem. In optiSLang 3.2 we combined the existing nature-inspired optimization flows evolutionary algorithm (EA) and genetic algorithms (GA) with the new flows particle swarm optimization (PSO) and simple design improvement (SDI) to the new workflow NOA. Therewith we provide a clear arranged overview for the user. A homogene dialog design is chosen to represent the common information and differences between all nature-inspired optimization methods. In version 3.1 we introduced Particle Swarm

*Contact: David Schneider, DYNARDO – Dynamic Software and Engineering GmbH, Luthergasse 1d, D-99423 Weimar, Germany, E-Mail: david.schneider@dynardo.de

Optimization. We improved its convergence behaviour in version 3.2 by adapting the accelerations coefficients and bringing some more parameters to the GUI front end. In version 3.2 we also introduced a new algorithm Simple Design Improvement (SDI). Further enhancements are three adaptive mutation methods for PSO and EA and hybrid crossover operators for EA. This paper will describe the new implementation and major modifications. To get a better understanding of the different algorithms and to find good and robust settings we performed a lot of practical and theoretical tests. We did this for single- and multiobjective tasks. This paper presents an excerpt of the results. To help finding the right settings we implement approved default settings that are based on our gained knowledge and experience. Additionally we give our users an advice which algorithm to use in a specific problem via an easy understandable decision tree. The user will be guided to select a proper algorithm for his special problem.

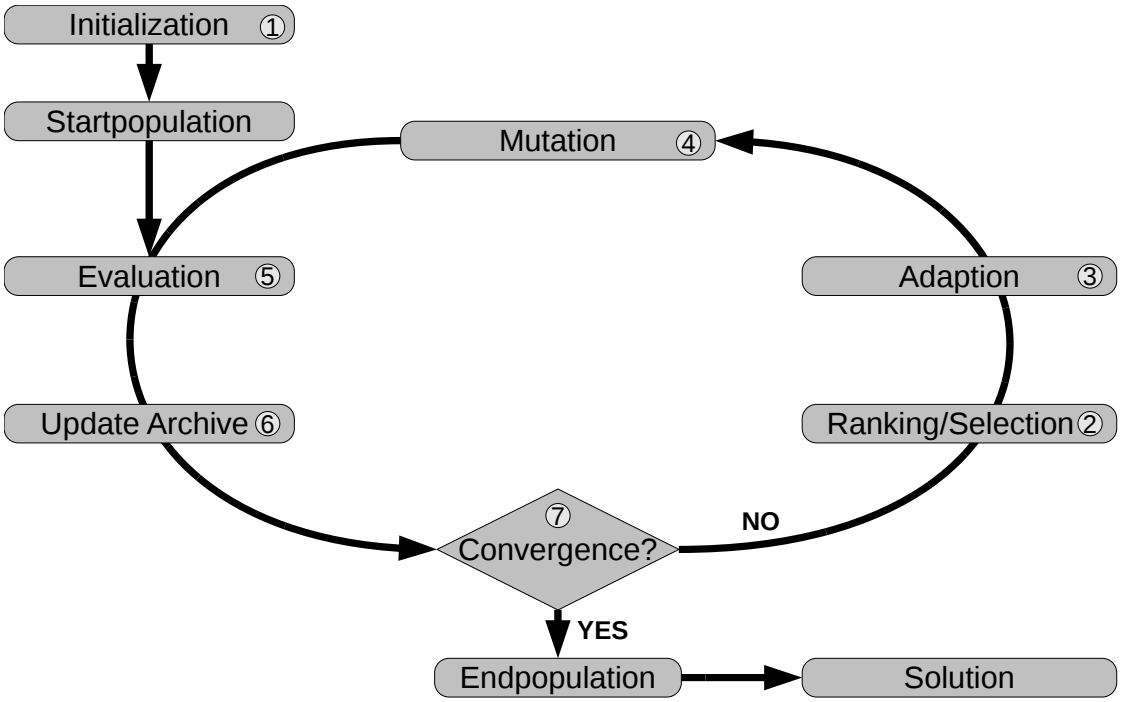

Figure 1: General flowchart of nature inspired optimization algorithms

All kinds of nature inspired algorithms are following a particular flow. It starts with a random initialization (see fig. 1 step (1)) of a population containing μ individuals representing possible solutions to the given optimization problem. After initialization the actual iteration loop starts, where every loop represents a generation g. To create a new generation the fittest designs will be selected (step (2)). The adaption refered to step (3) can be done in different ways, e.g. crossover or swarm movement. Mutation (step (4)) can be applied afterwards. All individuals are evaluated (step (5)) by assigning them a fitness value based on the objective value and possible constraint violations. Thus a high fitness score means a good accommodation to the problem and corresponds with a minimum objective value. After evaluating each design the archive, which keeps good solutions, will be updated for the next selection step (step (6)). The generation counter is increased and the iteration continues until a stopping criterion is fulfilled (step (7)).

2 Simple Design Improvement

Simple Design Improvement (SDI) is a nature inspired optimization method that improves a proposed design without extensive knowledge about interactions in design space. SDI works very robust while searching for better settings than the proposed one. SDI is not developed to find optimal solutions. It is designed to improve the proposal.

General SDI Flow A start population of size μ will be generated by a uniform distributed latin hypercube sampling Iman and Conover (1982); McKay et al. (1979) around a user defined start design $d_1 = (x_1, ..., x_n)^T$. Sampling bounds width $\gamma \in [0, 1]$ is chosen by the user. Thereby γ regards the percentage of the global search space (eq. 1f). Since SDI is a nature inspired optimization method producing the population from a latin hypercube sampling is interpreted as adaption step in NOA. The best design of the population will be evaluated. It is selected as center for the next sampling (Fig. 2). Depending on the optimization problem the whole population might move into a better region and achieve an improvement in each step.

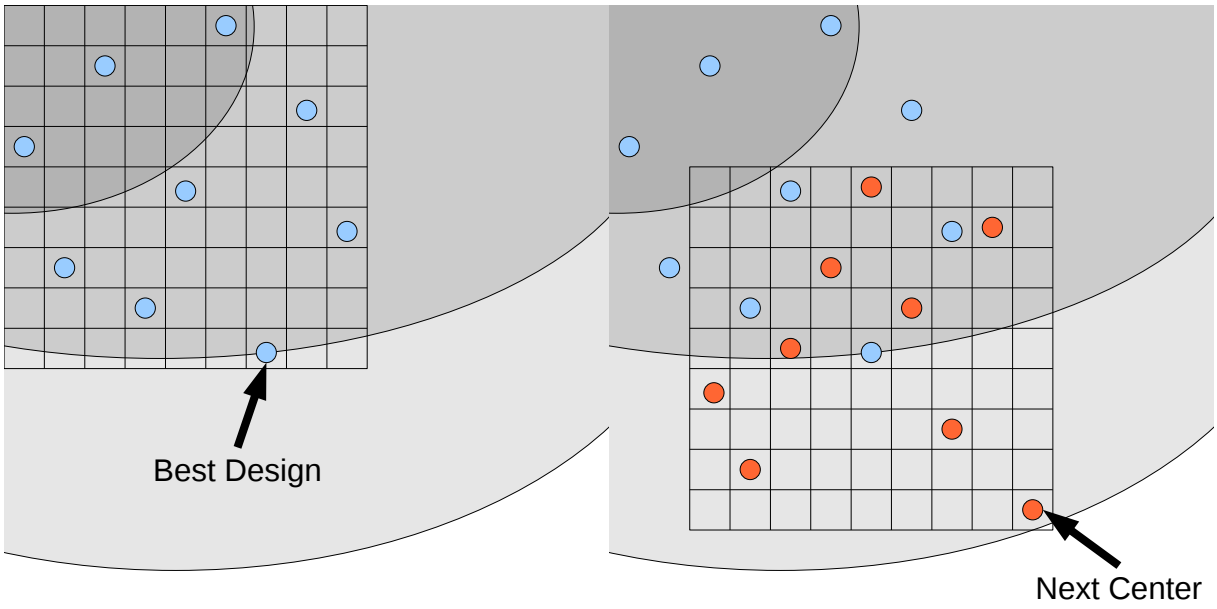

Figure 2: Movement of SDI - iterations

Selection In each iteration step SDI searches the best design of the current population. The one with best objective and least violations of constraints is chosen as center of the following sampling procedure.

Sampling The adaption step of SDI is a uniform distributed latin hypercube sampling around the current best design. The lower sampling bound $x_i^{l(t)}$ will be

$$x_i^{l(t)} = x_i^b - 0.5\gamma \cdot r_i \tag{1}$$

and the upper sampling bound $x_i^{u(t)}$ analogically

$$x_i^{u(t)} = x_i^b + 0.5\gamma \cdot r_i \tag{2}$$

with

$r_i = x_i^u - x_i^l$ global parameter range
x_i^b design parameter i of best of last generation

Evaluation and Replacement All individuals will be compared concerning objective and constraints considering a parameter-free constraint handling method.

- Split the population into feasible and infeasible solutions.
- Determine the fitness of feasible solutions based on their objective values
- Assign ranks to all infeasible solutions based on their constaint violations using the dominance criterion (see section 4).
- Determine the fitness of each infeasible solution by adding the fitness of the worst feasible solution to the rank value.

The archive will keep the winner of a generation. Hence, the archive size is 1.

Convergence SDI stops if the chosen convergence criteria will be reached or if the maximum number of generations g_{max} is reached. optiSLang provides the following 2 convergence criteria.

- *Type 1:* Gained improvement

$$\|o_j\| \leq D_1 \|o_1\| \tag{3}$$

o_j objective vector of the best design belonging to the j-th generation
D_1 scaling value of initial objective

E.g. Choosing $D_1 = 0.8$ means the algorithm stops if an improvement of 20% was reached since beginning.

- *Type 2:* Deterioration of performance

$$\|o_j\| \geq (1 + D_{j-1}) \|o_{j-1}\| \tag{4}$$

D_{j-1} scaling value of the previous best objective of the previous generation.

E.g. Choosing $D_{j-1} = 0.5$ would mean the algorithm stops if a deterioration of 50% to the last generation happened.

3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a nature-inspired optimization method developed by Eberhart and Kennedy (1995) that imitates the social behaviour of a swarm searching for feed. Information about good positions will be transmitted to other individuals of the swarm so that they move into direction of previously good found positions. In nature an individual can be represented by a bird, bee or fish. An overview of modifications of this algorithm can be found e.g. in Coello and Reyes-Sierra (2006).

General PSO Flow As every nature inspired optimizer PSO starts with initializing a population of size μ where each individual represents a possible solution of the optimization problem. Swarm intelligence is influenced by two main components, the personal and global component. This means that each individual remembers its personal best solution and will also be influenced by the best solution of the swarm. In the first iteration each individual will store its first position as local best. The group leader (global best) will be the best out of all current solutions.

Ranking and Selection The essential point of the PSO is to find good positions to move the swarm to. Therefore all individuals will be compared considering objective and constraints to evaluate the best among them. Each individual remembers its personal best out of all timesteps. Global best will be the best out of all found positions of the whole swarm.

Movement Each individual will change its position into direction of its personal best found position and the global best found position. There are three important parameters that influence the speed and spread of the swarm. The inertia weight w_t is a scaling factor for the previous velocity. The personal acceleration coefficient $c_{p,t}$ is a scaling factor for the second term of eq. 5, which is also called cognitive component. The third term, called social component, is scaled by the global acceleration coefficient $c_{g,t}$.

$$\begin{aligned} v^i_{t+1} = \quad & w_t \cdot v^i_t \\ & + c_{p,t} \cdot R[0,1] \cdot (x^i_{p,t} - x^i_t) \\ & + c_{g,t} \cdot R[0,1] \cdot (x_{g,t} - x^i_t) \\ x^i_{t+1} = x^i_t + v^i_{t+1} & \end{aligned} \tag{5}$$

i	index of design
t	generation number
$x^i_{p,t}$	personal best position of design i
$x_{g,t}$	global best position
x^i_t	current position of design i
w_t	inertia weight
$c_{p,t}$	personal acceleration coefficient / cognitive component
$c_{g,t}$	global acceleration coefficient / social component
v^i_t	velocity of design i in generation t
$R[0,1]$	uniform distributed random variable between 0 and 1

The choice of good coefficients is very important for the convergence behaviour of PSO. Because in most applications it does not make sense to apply meta optimization optiSLang offers two predefined search strategies with default values for each coefficient. Local search strategy should be chosen if the user has some information about the optimization space and has maybe even preoptimized the problem. The swarm movement is slower and less intensive throughout all generations with the following values $w_t = 0.4$, $c_{p,t} = 1.0$ and $c_{g,t} = 1.0$. Global search strategy is recommended for mathematically ill or large problems. The swarm movement is very intensive in the beginning of the search and will be damped throughout the optimization process. Default values for the coefficients will be varied linear with $w_t = 0.9...0.2$, $c_{p,t} = 0.9...0.1$ and $c_{g,t} = 0.1...0.9$. Movements into direction of the global best will be stronger in the last terms of optimization and movements into direction of each personal best will be stronger in the first terms. That way a exploration in first terms and exploitation in last terms can be reached.

Mutation To allow a more sophisticated search, optiSLang provides the same mutation methods that can be used in evolutionary algorithms (section 5.2). For executing a classical PSO search one has to choose "None" - mutation. I.e. movement is the only working operator.

Evaluation and Archive Update The archive will be updated with the best solution, called global best. Hence, the archive size is 1. To find the global best solution the algorithm compares all individuals concerning objective and constraints. The feasable solution with minimal objective will be the global best (*dominance method*). If there are only infeasable solutions in the first population global best will be the individual with least constraint violations (*strength pareto method*).

4 Multiobjective Particle Swarm Optimization Algorithm

The Particle Swarm Optimization algorithm can also be applied to multiobjective optimization problems. Because there is not only one optimal point some changes in its handling occur. The stagnation criterion observes the changing ratio of the archive. In each iteration the archive designs will be compared to the archive of the last iteration and the relative frequency of changes will be calculated. Due to the fact that multiobjective archive keeps more than one design this ratio should be selected less strict. The maximum rate of changes in archive can be modified by the user. A recommended value of stagnation ratio would be between 10% and 20%. The main difference to singleobjective PSO is the treatment for archive selection which is based on the comparison of individuals. As mentioned in section 3 for single objective PSO optiSLang selects the fittest design as global leader. For the next timestep the swarm will move in this direction. In multiobjective optimization the fitness evaluation is different from the singleobjective approach.

optiSLang provides a strict dominance-based method for multiobjective fitness evaluation in PSO . The Pareto-dominance criterion is formulated for two decision vectors $\mathbf{a}$ and $\mathbf{b}$ as follows

- Solution $\mathbf{a} \in X$ dominates solution $\mathbf{b} \in X$ if it is feasable and better or equal in all objectives and better in at least one objective.

- Solution $\mathbf{a} \in X$ is indifferent to solution $\mathbf{b} \in X$ if none of both solutions dominates the respective other.

The dominance relation can be expressed as:

$$\begin{aligned} \mathbf{a} \succ \mathbf{b} \quad \Leftrightarrow \quad & \forall\ i \in \{1,2,\ldots,M\}: \quad f_i(\mathbf{a}) \leq f_i(\mathbf{b}) \\ & \wedge \forall\ j \in \{1,2,\ldots,J\}: \quad g_j(\mathbf{a}) \geq 0 \\ & \wedge \forall\ k \in \{1,2,\ldots,K\}: \quad h_k(\mathbf{a}) = 0 \\ & \wedge \exists\ l \in \{1,2,\ldots,M\}: \quad f_l(\mathbf{a}) < f_l(\mathbf{b}) \end{aligned} \tag{6}$$

The archive will keep all nondominated solutions for selecting the global best design. It is recommended to choose a high archive size to get a good representation of the pareto front. If there are no valid designs in the first generation e.g. because of many constraints, the strict dominance method will be changed automatically by optiSLang to the dominance based fitness method, strength pareto. The strength pareto method is a dominance-based ranking which takes into account by how many individuals a solution is dominated and how many individuals a solution dominates [Zitzler et al. (2001)].
For efficient usage the archive size must not exceed the population size which is changed automatically by optiSLang . A constant number of designs will be filled into the archive in each generation step. If there are enough pareto designs the archive will only store nondominated designs else it will be filled with suboptimal designs. To make sure that no information about good solutions will get lost we store all pareto-optimal solutions in the archive even if the maximum archive size is reached for the strength pareto criteria.

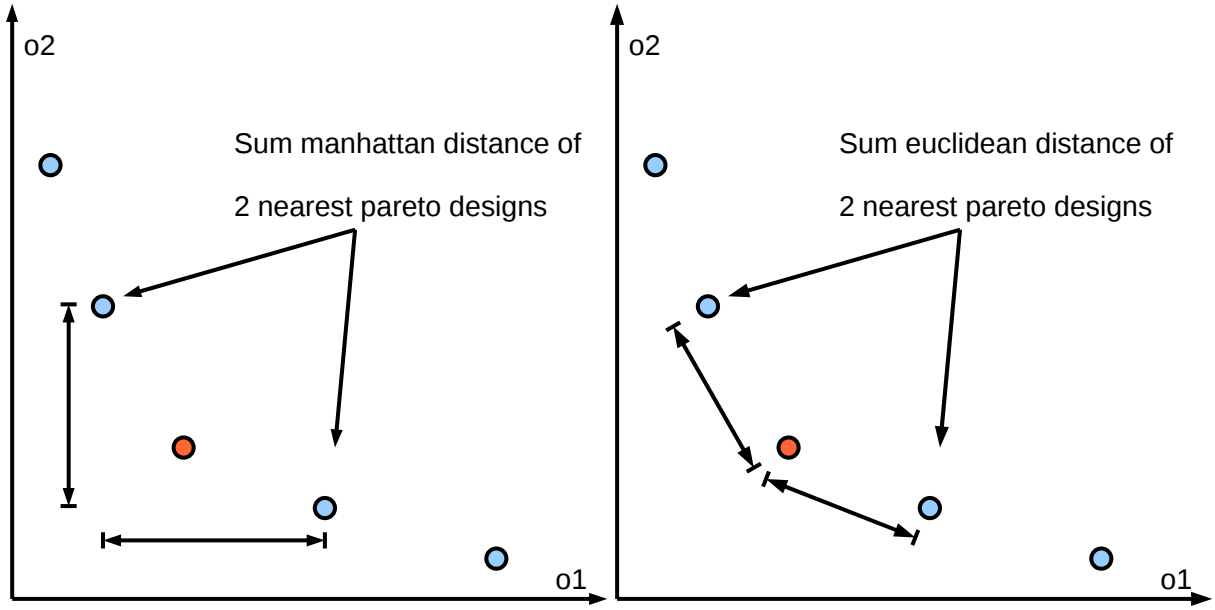

Figure 3: Refinement criteria: crowding distance and cumulated distance

The target of multicriteria optimization is to display the front uniformly. Less observed regions of the pareto front are prefered to be stored in the archive. optiSLang provides two density ranking methods that will be considered for the strict dominance based mehod (Fig. 3).

Crowding Distance Sum of the manhattan distance of the 2 nearest pareto designs to the observed one Raquel and P.C. Naval (2005).

Cumulated Distance Sum of the euclidean distance of the 2 nearest pareto designs to the observed one.

To reach an evenly spread pareto front, individuals that are very close to their neighbors will be changed by new nondominated individuals then. For global best selection all archive designs will be sorted in ascending order by the density method. With **local search** strategy the top archive individual will be chosen for global best (move to most crowded region), while with **global search** strategy the global best will be chosen randomly based on a uniform distribution out of 10% of the top archive.

5 More enhancements

In addition to the realization of new methods some changes and enhancements to previous versions have to be mentioned here. First of all we improved the performance of the flow by unifying the implementation in C++. Thus we can access and provide a C++ library of the nature-inspired optimization algorithms. We brought some more parameters to the GUI front end to allow choosing suitable settings. We also extended the number of possible combinations of different operators. Through homogene dialog design and a clear arranged overview the common information and differences between all nature-inspired optimization methods will be represented. Beside these enhancements we implemented hybrid crossover operators and adaptive mutation. These will be explained in detail in the following text.

5.1 Hybrid Crossover/Recombination

The crossover operator is a method of recombination where two parent individuals produce two offsprings by sharing information between chromosomes. The intention is to obtain individuals with better characteristics (exploitation) and to maintain the diversity of the population (exploration). Crossover is regarded as the main search operator in GAs. The original formulation of GAs uses a binary representation of the decision variables. The optiSLang implementation uses a real-valued coding where each chromosome is a vector of floating point numbers, representing the decision variables of an individual. Crossover operators for real-coded GAs can be classified into the following groups:

Discrete Crossover Operators: This group is based on binary crossover operators and includes singlepoint , multipoint and uniform crossover. These operators do not change the numerical values of the genes but exchange them between the parent chromosoms according a specific scheme. Geometrically they generate a corner of the hypercube defined by the two parent chromosoms.

Aggregation Based Crossover Operators: These operators use an aggregation function to combine the genes of the parents numerically to generate the genes of the offspring. The arithmetic crossover is a representative of this group.

Neighbourhood Based Crossover Operators: The genes of the new individuals are determined from intervals defined in neighbourhoods of the parent genes throughout probability distributions. An example of this group is the simulated binary crossover.

Hybrid Crossover Operators Crossover operators are recombining two parent chromosoms according to specific schemes. For different optimization problems a different crossover operator might be the most suitable. The idea behind hybrid crossover is to apply diverse crossover operators on the parent chromosoms in order to take advantage of their distinct offspring generation mechanisms. optiSLang provides the selection of two crossover operators to keep the number of control parameter as low as possible. Both operators produce two offspring chromosoms from two parents, but only one offspring of each pair is randomly chosen. Using a hybrid crossover can improve the effectiveness of this genetic operator.

5.2 Adaptive Mutation

Mutation introduces random variation to the genes of the offspring chromosom. In general there exist two parameters that can be modified by the user, standard deviation and mutation rate. Each gene g_i is selected for mutation with a specified probability or mutation rate respectively. The real-valued mutation is based on a normal distribution function for each gene with the value of the gene as its mean value. The standard deviation σ of this distribution influences the size of the mutation steps.

$$g_i^{t+1} = g_i^t + N(0, \sigma_i^t) \;\; ; \quad i \in (1, n) \tag{7}$$

Mutation is the main search operator for ES (evolutionary strategies), but can also be applied after recombination (see fig. 1). The two parameters mutation rate and standard deviation can stay constant or get modified during the run of the algorithm. In the first case the user needs to specify appropriate values for the parameters, what requires much experience and always depends on the specific optimization problem. For the mutation rate a value of $1/n$ is recommended, where n is the number of variables. The standard deviation is defined in relation to the variable range. This parameter can strongly influence the performance of the optimization. Depending on the problem and the optimization strategy, standard deviations between 0.01 and 0.1 are appropriate values for this strategy parameter.

Self Adaptive Mutation Baeck (1996): „Technically, this so-called self-adaption principle combines the representation of a solution and its associated strategy parameters within each individual, and the strategy parameters are subject to mutation and recombination just as the object variables.“In this method the standard deviation of a normal distribution for mutation will be adapted during the search. All design parameter (object variable) that are selected for mutation will be mutated according to a normal distributed random number as in Equation 7. Other than the above mentioned approach the standard deviation σ_i will be mutated according to a logarithmical distribution in each generation.

$$\sigma_i^{\backprime t} = \sigma_i^t \cdot \exp s^{\backprime} \cdot \exp s_i \tag{8}$$

where $s^{\backprime} \sim N(0, \frac{1}{2n})$ and $s_i \sim N(0, \frac{1}{2\sqrt{n}})$. Strategy parameter σ_{new} is obtained from arithmetical recombination ($\lambda = 0.5$) of the parents mutation standard deviations $\hat{\sigma}_k, \hat{\sigma}_l$.

$$\sigma_{new} = (1 - \lambda)\hat{\sigma}_k + \lambda\hat{\sigma}_l \tag{9}$$

Constraint Adaptive Mutation In this method the standard deviation of a normal distribution for mutation depends on the distance between each design and all violated designs. The intention of this approach is to consider the violation of constraints in a special way. Designs that are close to an infeasable region will be mutated less than designs far away from infeasable regions. Therefore the distances between each parameter j of design i of the population to all violated designs k will be calculated. These violated designs are stored in an extra archive and consider also previous generations.

$$d_j^{ik} = |x_j^k - x_j^i| \tag{10}$$

The distances will be sorted in ascending order for each parameter j and the μ shortest distances will be quadratically accumulated. The standard deviation of the normal distribution function for mutation of design i for parameter j will be calulated by

$$\sigma_j^i = \sqrt{\alpha \cdot \sum_{k=1}^{\mu} d_j^{ik}} \tag{11}$$

Within optiSLang the value of α is 0.1. Hence, this method is only active if there are violated designs. Otherwise a mutation based on a normal distribution with linear descreasing standard deviation will be applied.

Modulated Adaptive Mutation The method analyzes successful mutations and adaptively changes the distribution hypothesis for mutation steps. The mutation of the gene g_i is defined as

$$g_i^{t+1} = g_i^t + X_i \;\; ; \quad i \in (1, n) \tag{12}$$

where X_i is a random variable with zero mean and standard deviation σ_i^t. At the initial stage a normal distribution is assumed for each random variable X_i. At starting point the corresponding *probability density function* (PDF) is given by

$$\varphi^t(x_i) = \frac{1}{\sigma_i^t \sqrt{2\pi}} \exp\left[-\frac{x^2}{2(\sigma_i^t)^2}\right] \;\; ; \quad t = 0 \tag{13}$$

The adaption takes place on component level and is based on the evaluation of successful mutations. For a successful mutated offspring individual, whose fitness is better than the fitness of its parent, the mutated genes and the realised mutation steps δ_i are identified, seeRiedel et al. (2005). The probability density function of the random variable corresponding to a mutated gene, which has led to an improved fitness, is modulated by a symmetric modulation function. The method provides a self-adaptive strategy for changing the probability density of the mutation steps. The probability of mutation steps which led to an improvement is increased by the modulation function. Because the strategy evaluates single mutations of individuals, it can only be applied to evolution strategies, where no crossover operator can blur the effects of successful mutations.

6 Benchmarks

To appraise the quality of the enhancements and new algorithms we applied a lot of practical applications and many of the testfunctions that could be found in literature. Several combinations

of the available nature-inspired optimization algorithms in optiSLang were tested with different single and multiobjective testfunctions. Different parametersettings that can be chosen have been tested. Amongst easy understandable parameters like population size or maximal number of generations, we modified parameters like the number of crossover points n_{xover}, crossover rate/probability p_{xover}, distribution parameter n_c, mutation rate r_{mutate} and maximal archive size $n_{archive}$ and so on. The different settings will be described in the belonging subsections. The aim was also to validate the existing or find some more approved default settings.
Before we start presenting some of our results one have to know that the found optimum of a nature-inspired algorithms relies on the stochastic behavior that comes with the definition of the method. To avoid judging on luck we ran each setting 100 times for each problem. Than we compared the statistics over all optimization runs. Therefore we take mean, standard deviation and median in singleobjective case. In the case of multiobjective optimization we judge on the average number of designs that are dominated. These designs are the shifted pareto fronts of initial reference runs. Due to some limits in time and resources we did not apply this procedure to practical applications. For time consuming problems we used the settings which showed small standard deviations in academic examples. These more or less robust settings had been considered to give comparable results with less (or only one) optimization run. The average time of a single optimization run can also be seen in all result tables. But for the reason of some computational overhead in optiSLang 3.x we do not judge on that in this paper. These times should only be regarded if the C++-library of the tested workflow is used and solverruns are sufficiently fast, respectively.

6.1 Singleobjective Problems

We tested a lot of practical applications and theoretical testfunctions that can be found in literature (Hock and Schittkowski (1980); Moré et al. (1981); Molga and Smutnicki (2005); Storn and Price (1995) etc.). The tested problems have different characteristics. Thus the behaviour of the algorithms and settings is tested for large input spaces, numerous constraints and problems with a lot of local optima or discrete responses and inputs.
Because each algorithm is following a stochastical pattern of behavior single test runs would not be very significant. For that reason we tested each algorithm setting 100 times and evaluated statistical data of best objective as mean $\bar{x}_{bestobj}$, median $\tilde{x}_{bestobj}$, standard deviation $s_{bestobj}$, minimum x_{min} and maximum x_{max}. All algorithms had been tested for several population size and number of maximum generations. An excerpt of the numerical results can be seen in section A.3.

Rosenbrock: This smooth testfunction in a 10-dimensional space has an optimum at $x_i = 1, \quad i = 1, \cdots, 10$. The standard deviation of setting S7 is smaller than that of others. I.e. the global PSO converges robust (in every run) near to the optimum. Settings S8 and S10 are comparably good for most runs, but higher x_{max} and $s_{bestobj}$ show that the algorithms are less robust.

Griewank: This testfunction has several local minima in $[-600, 600]^5$, but a global minima at $x_i = 0, \quad i = 1, \cdots, 5$. Here settings S7, S8, S10, S11, S15 and S16 show good results with a small standard deviation. Even the SDI shows good results with less designs. By trend PSO and SDI converge earlier which means they need less calculations than EAs.

Name	Method	Adaption	Ranking	Selection	Mutation	Special parameter
S1	EA	Singlepoint	Exponential	Stochastic	None	
S2	EA	Multipoint	Linear	Stochastic	None	$n_{xover} = \frac{n}{2}$
S3	EA	Uniform	Linear	Stochastic	None	$p_{xover} = 0.5$
S4	EA	Copy	Linear	Stochastic	None	
S5	EA	SBX	Linear	Stochastic	None	$p_{xover} = 0.5$ $n_c = 5$
S6	EA	Arithmetic	Linear	Stochastic	None	p_{xover} = 0.5
S7	PSO(global)	-	-	-	normal distributed	$r_{mutate} = 0.2$ $w_t = 0.9 \ldots 0.2$ $c_{p,t} = 0.9 \ldots 0.1$ $c_{g,t} = 0.1 \ldots 0.9$
S8	PSO(local)	-	-	-	normal distributed	$r_{mutate} = 0.2$ $w_t = 0.4$ $c_{p,t} = 1.0$ $c_{g,t} = 1.0$
S9	SDI	-	-	-	-	$\gamma = 0.1$
S10	EA	Multipoint	Linear	Stochastic	normal distributed	$r_{mutate} = 0.2$ $n_{xover} = \frac{n}{2}$
S11	EA	Multipoint	Linear	Stochastic	self adaptive	$n_{xover} = \frac{n}{2}$
S12	EA	Multipoint	Linear	Stochastic	modulated adaptive	$n_{xover} = \frac{n}{2}$
S13	PSO(global)	-	-	-	self adaptive	$r_{mutate} = 0.3$ $n_{archive} = 100$
S14	PSO(global)	-	-	-	self adaptive	$r_{mutate} = 0.2$ $n_{archive} = 500$
S15	EA	Multipoint	Linear	Stochastic	self adaptive	$n_{xover} = \frac{n}{2}$ hybrid += SBX
S16	EA	Multipoint	Linear	Stochastic	self adaptive	$n_{xover} = \frac{n}{2}$ hybrid += Uniform

Table 1: Overview of tested nature-inspired algorithms with different settings for single-objective testproblems

Rastrigin: This function has similar characteristics as Griewank, but is defined in a smaller space $[-5.12, 5.12]^3$. S10, S11, S15 and S16 give best results with only a small standard deviation.

Six-hump-camel-back: This function does not have local optima, but 2 global optima. All PSO settings and EA with mutation are very good. Setting S8 will reach the one optima in every run because of $s_{bestobj} = 0$.

Ackley: We tested this function with $x_1, x_2 \in [-30, 30]$. There are many local minima but in direction of the global minima the difference of function values of neighbored minima increase. Best results showed setting S8 with only a small standard deviation, but also S7, S10 and S10 reached good results. Results of PSO are acceptable but in some runs the algorithm converged towards a neighbored local optima.

10 bar truss: This problem was tested a few times in optiSLang 3.2 and optiSLang 3.1.4. In that runs the global PSO was worse than EA settings. The reason is the influence of the inequality constraints to the quality of PSO. Local improvement of the preobtained solution gives good results with all algorithms.

Conclusions: Results for EA without mutation (S1-S6) are worse compared to other settings for each testfunction. PSO with normal mutation (S7/S8), EA with Multipoint crossover, self adaptive mutation and hybrid crossover SBX (S15) and EA with Multipoint crossover and normal mutation (S10) showed good results for all testfunctions. It should be mentioned that setting S10 is by neglecting some minor changes default setting in version 3.1.x. Setting S15 is now chosen as default setting for EA. Setting S7 is the new default setting for global PSO, S8 for local PSO.

6.2 Multiobjective Problems

A lot of testfunctions that can be found in literature (Deb et al. (2002); Okabe et al. (2004); van D. A. and B. (2000); Osyczka and Kundu (1995) etc.) have been tested. These problems have different characteristics. Thus the behaviour of the algorithms and settings is tested for large input spaces, numerous constraints, problems with local pareto optimal solutions, disconnected pareto fronts and so on.
In a single run with 50000 calculations ($\mu = 100$, $\lambda = 100$ and $g_{max} = 500$) and setting PSO (M3) we created a reference file with pareto designs for testproblems Kita, Kursawe, Deb, Tanaka and with setting EA (M9) for testproblem Osyczka respectively. Reference pareto fronts can be seen in section A.2. We have chosen two test criteria for each testfunction. Therefore we moved the reference front by 1% and 10% into the non-dominant direction and check if the pareto front of each setting dominates these. We count the number of dominated points of the shifted reference front (#DP1 / #DP10). Some of our results are shown in A.4 and listed below.

Kita: All PSO settings gave good results. Even with 400 designs ($\mu = 10$, $g_{max} = 40$) the first test criteria was fullfilled with about 66%. Compared to that EA with multipoint crossover and several mutation methods only reached about 30%. With $\mu = 50$ and $g_{max} = 200$ PSO is still the best algorithm for this testproblem with only 1500 calculated designs.

Name	Method	Adaption	Ranking	Selection	Mutation	Special parameter
M1	EA	Multipoint	Pareto	Tourn.	None	$n_{xover} = \frac{n}{2}$
M2	EA	Uniform	Pareto	Tourn.	None	$p_{xover} = 0.5$
M3	PSO (global)	-	-	-	normal distributed	D, $n_{archive} = 1000$ $r_{mutate} = 0.3$ $w_t = 0.9 \ldots 0.2$ $c_{p,t} = 0.9 \ldots 0.1$ $c_{g,t} = 0.1 \ldots 0.9$
M3*	see M3					SP, $n_{archive} = 10$
M4	PSO (local)	-	-	-	normal distributed	D, $n_{archive} = 1000$ $r_{mutate} = 0.3$ $w_t = 0.4$ $c_{p,t} = 1.0$ $c_{g,t} = 1.0$
M4*	see M4					SP, $n_{archive} = 10$
M5	SDI	-	-	-	-	$\gamma = 0.1$
M6	EA	Multipoint	Pareto	Tourn.	normal distributed	$r_{mutate} = 0.2$ $n_{xover} = \frac{n}{2}$
M7	EA	Multipoint	Pareto	Tourn.	self adaptive	$n_{xover} = \frac{n}{2}$
M8	PSO (global)	-	-	-	self adaptive	D, $n_{archive} = 100$
M8*	see M8					SP, $n_{archive} = 10$
M9	EA	Multipoint	Pareto	Tourn.	self adaptive	$n_{xover} = \frac{n}{2}$ hybrid += SBX
M10	EA	Multipoint	Pareto	Tourn.	self adaptive	$n_{xover} = \frac{n}{2}$ hybrid += Uniform

Table 2: Overview of tested nature-inspired algorithms with different settings for multi-objective testproblems

Kursawe: The phenomenon of disconnected pareto fronts appears quiet often in real world problems. With the Kursawe testproblem we can benchmark the different algorithms and settings on this characteristic Kursawe (1991). Differences of results between PSO and hybrid EA with self adaptive mutation can hardly be found. PSO gives better first criteria results and EA better second criteria results for $\mu = 10$ and $g_{max} = 40$. With higher population size EA gives better first criteria results too. The pareto front of this testproblem is divided into 3 sections. A bigger population size is suggested. So a local convergence can be prevented.

Deb: The special characteristic of this testproblem is that there are local and global pareto-optimal solutions. The local pareto-optimal solutions occur at $y = 0.6$ and the global ones at $y = 0.2$. Some algorithms have problems with finding the global pareto front, see Deb (1999). EA with mutation (M6, M7) and hybrid crossover (M9, M10) are the only settings that find the global pareto front in its whole range in most optimization runs. PSO has problems in finding a good representation of the global front. It finds some global solutions but not enough to cover the whole range. The main problem of PSO in this case is the alternation of two good solutions. Once global best is close to $y = 0.6$ and next time it is close to $y = 0.2$. So the swarm will move in an area between these two points and can not converge to a solution. Additionally the influence of global best will increase with further generations, but the chance of moving towards the local solution is higher if this region is searched better in the beginning because the archive will keep more local pareto-optimal solutions then. So PSO can get trapped into the local pareto front lately.

Tanaka: PSO gives better results than other algorithm settings, but even with $\mu = 50$ and $g_{max} = 200$ the first test criteria is fullfilled only 30%. The second criteria generally fulfills the requirements. The optimizer converges to early. Choosing a smaller stagnation rate can help to overcome this problem.

Osyczka: Since this testproblem has 6 parameters we have better results with a higher population size ($\mu = 100$ instead of $\mu = 50$). Because of the number of constraints it might be hard then to find valid designs, we used strength pareto (settings M3*, M4* and M8*). Best results had been found by EA with hybrid crossover and self adaptive mutation.

Conclusions: Looking at the result we can determine that there is not a special setting that is always better than the others. The quality of all settings differ a lot for each testproblem. There is a trend which we can ratify from some practical applications that PSO works better in case of continuous input space and only a few and less violated constraints. As far as the number of discrete inputvariables rises or constraints are violated often EA works better. Most practical applications showed good results when using self adaptive mutation (see also 3). Evolutionary algorithms generally work better with hybrid crossover. As the tests show a combination of multipoint and simulated binary crossover gives good and stable results. Higher population sizes tend to give a better representation of the pareto front. These settings will be default for the new pareto workflow.

7 Recommended Applications

NOAs produce good results for many problems but they often converge slower compared to other optimization methods. Depending on the choosen settings for some problems they tend to converge towards local optima rather than the global optimum of the problem. But rapidly locating good solutions even for difficult search spaces is an advantage.

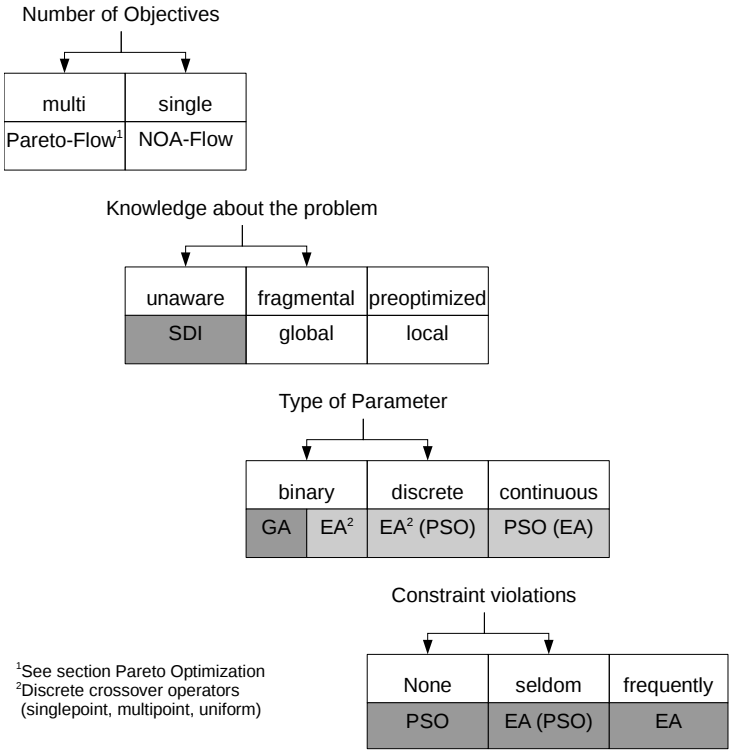

Figure 4: Recommended Applications

The use of NOA algorithms are recommended:

- in all cases where gradient based optimization or response surface approximation fails.
- in case of a high number of variables or constraints.
- in case of discrete or binary variables
- in case of discrete responses
- if the user is unaware about the problem

The offered nature-inspired algorithms differ partially a lot. Their usage and suggested settings depend on the number of objectives as well as any preoptained information about the behaviour of the investigated system, the type of variables and the character of constraint violations. An overview of the advised selection can be seen in figure 4. A decision tree is shown. Therein a light shaded rectangle gives a possible solution, a dark shaded rectangle means a decision.

Example 1 No information about the behaviour of the system, that is to be optimized, exists or a complete new solution should be found. Recommended application: SDI

Example 2 Based on previous observations, a preoptimized design exists. All variables of the underlying scope are continuous. Recommended application: PSO local

Example 3 Example 2 but with a further information: The problem contains frequently violated constraints. Recommended application: EA local

In some cases the use of NOA can be more complex than the usage of other strategies. One possibility to reduce this effort is the search for optima on response surfaces. Because evaluations based on metamodels are very cheap, many generations should be used here.

7.1 Multiobjective optimization

Multiobjective optimization is applied if the task contains at least two conflicting objectives. Generally the number of objectives describing the optimization problem is unlimited. Formulating optimization problems with more than three conflicting objectives should become a challenging task for the engineer and also the analysis of the resulting hypersurface is nontrivial.

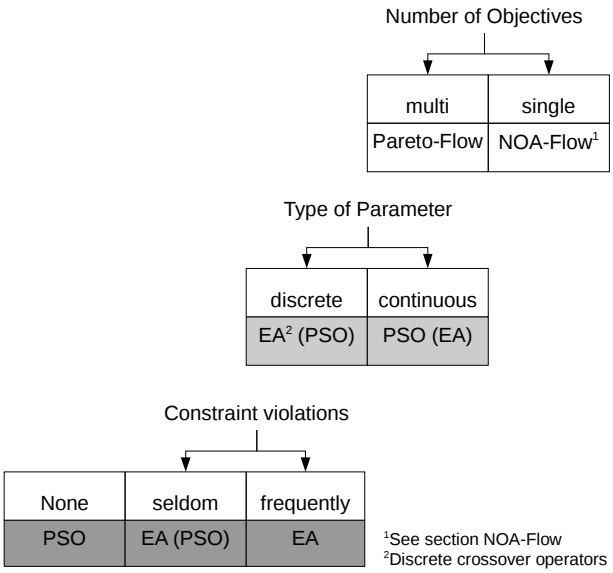

Figure 5: Recommended Applications

The application of multi-objective-optimization is recommended if there is a comprehensive knowledge about the problem, which not only includes the objective functions but also important design variables and constraints. For those reasons this method should not be used when analyzing a problem for the first time. If objectives are positive correlated, or non conflicting respectively, the optimum of the problem would be represented by a point instead of a trade-off front. It should be decided whether the application of a task with a reduced number of objectives (singleobjective) is more suitable.

The offered algorithms differ partially a lot. Their usage and suggested settings depend on multiple parameters. Therewith it is not easy to find a suitable algorithm for non - specialists. To help the user to choose good settings even for very difficult tasks, a decision tree is given in figure 5. The interpretation of this decision tree equals that given in the single objective section (sec. 7, fig. 4).

8 Conclusion

Within stochastic analyses, nature-inspired optimization algorithms (NOA) imitate natural processes like biological evolution or swarm intelligence. Based on the principle „survival of the fittest“, a population of artificial individuals searches the design space of possible solutions in order to find a better solution of the optimization problem. The usage of these algorithms is recommended for various applications. NOA are very robust against mathematically ill-conditioned problems. Thus, with help of the NOA wizard within optiSLang , difficult single or multiobjective optimization tasks can be solved even for large and difficult search spaces.
Particle swarm optimization algorithm and simple design improvement had been added to the algorithms that were already available in previous versions of optiSLang . Together they compose the new NOA flow. This paper also presented some improvements and enhancements that come with optiSLang 3.2. Three adaptive mutation operators had been implemented as well as the possibility to combine multiple crossover operators. Many practical and theoretical benchmarks have been accomplished to appraise the behaviour of the existing and appearing algorithms and settings. Some of them had been named and listed. It could be seen, that the introduced enhancements improved the behaviour of the implemented algorithms a lot.
The offered natural inspired algorithms differ partially a lot. Their usage and suggested settings depend on multiple parameters. Therewith it is not easy to find a suitable algorithm for non - specialists. To help the user to choose good settings even for very difficult tasks, optiSLang provides various approved pre-settings and an easy understandable decision tree. This allows even non-optimization specialists to successfully use (multi-disciplinary) optimization.

9 References

References

T. Baeck. Evolution strategies: An alternative evolutionary algorithm. *Lecture Notes in Computer Science*, 1063/1996:1–20, 1996.

C. A. Coello and M. Reyes-Sierra. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2:287 – 308, 2006.

K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary computation*, 7(3):205–230, 1999. ISSN 1063-6560.

K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 825–830. IEEE, 2002. ISBN 0780372824.

R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, pages 39–43, 1995.

W. Hock and K. Schittkowski. Test examples for nonlinear programming codes. *Journal of Optimization Theory and Applications*, 30(1):127–129, 1980. ISSN 0022-3239.

R. L. Iman and W. J. Conover. *A Distribution Free Approach to Inducing Rank Correlation Among Input Variables*. Communications in Statistics, vol. b11 edition, 1982.

F. Kursawe. A variant of evolution strategies for vector optimization. *Parallel Problem Solving from Nature*, pages 193–197, 1991.

M.D. McKay, R.J. Beckman, and W.J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21 (2):239–245, 1979.

M. Molga and C. Smutnicki. Test functions for optimization needs. 2005. URL http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf.

J.J. Moré, B.S. Garbow, and K.E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41, 1981. ISSN 0098-3500.

T. Okabe, Y. Jin, M. Olhofer, and B. Sendhoff. On test functions for evolutionary multi-objective optimization. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 792–802. Springer, 2004.

A. Osyczka and S. Kundu. A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm. *Structural and Multidisciplinary Optimization*, 10 (2):94–99, 1995.

C.R. Raquel and Jr. P.C. Naval. An effective use of crowding distance in multiobjective particle swarm optimization. In *Genetic and Evolutionary Computation Conference*. ACM Press, Washington DC, USA, 2005.

J. Riedel, S. Blum, R. Puisa, and M. Wintermantel. Adaptive mutation strategies for evolutionary algorithms: A comparative benchmark study. *Weimarer Optimierungs- und Stochastik Tage 2.0*, 2005.

R. Storn and K. Price. Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces. *INTERNATIONAL COMPUTER SCIENCE INSTITUTE-PUBLICATIONS-TR*, 1995. ISSN 1075-4946.

Veldhuizen van D. A. and Lamont G. B. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147, 2000.

E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

A Appendix

A.1 Singleobjective Testfunctions

Testfunction Rosenbrock

$$\begin{aligned} f(\vec{x}) &\rightarrow \min \\ f(\vec{x}) &= \sum_{i=1}^{n}((100(x_{i+1} - x_i^2)^2) + (1 - x_i)^2) \\ \vec{x} &\in [-2.4, 2.4]^n \end{aligned} \tag{14}$$

optimal point: $x_i = 1, \quad i = 1, \ldots, n$
optimal value: $f(x) = 0$

Testfunction Griewank

$$\begin{aligned} f(\vec{x}) &\rightarrow \min \\ f(\vec{x}) &= \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1 \\ \vec{x} &\in [-600, 600]^n \end{aligned} \tag{15}$$

optimal point: $x_i = 0, \quad i = 1, \ldots, n$
optimal value: $f(x) = 0$

Testfunction Rastrigin

$$\begin{aligned} f(\vec{x}) &\rightarrow \min \\ f(\vec{x}) &= 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i)) \\ \vec{x} &\in [-5.12, 5.12]^n \end{aligned} \tag{16}$$

optimal point: $x_i = 0, \quad i = 1, \ldots, n$
optimal value: $f(x) = 0$

Testfunction Sixhump

$$
\begin{aligned}
f(x,y) &\rightarrow \min \\
f(x,y) &= (4 - 2.1x^2 + (x^4)/3)x^2 + xy + (-4 + 4y^2)y^2 \\
x, y &\in [-2,2]^n
\end{aligned}
\tag{17}
$$

optimal points: $(x1, y1) = (-0.0898; 0.7126)$,
$(x2, y2) = (0.0898; -0.7126)$
optimal value: $f(x) = -1.0316$

Testfunction Ackley

$$
\begin{aligned}
f(\vec{x}) &\rightarrow \min \\
f(\vec{x}) &= -ae^{-b\sqrt{1/n \sum_{i=1}^{n} x_i^2}} - e^{1/n \sum_{i=1}^{n} \cos(cx_i)} + a + e \\
& a = 20, \quad b = 0.2, \quad c = 2\pi \\
\vec{x} &\in \mathbb{R}^n \\
x_i &\in [-30, 30] \rightarrow \text{ many local optima} \\
x_i &\in [-2, 2] \rightarrow \text{area of global optimum}
\end{aligned}
\tag{18}
$$

optimal point: $x_i = 0, \quad i = 1, \ldots, n$
optimal value: $f(x) = 0$

A.2 Multiobjective Testfunctions

Testfunction Kita

$$
\begin{aligned}
f_1(x,y), f_2(x,y) &\rightarrow \min \\
f_1(x,y) &= x^2 - y \\
f_2(x,y) &= -0.5x - y - 1 \\
x, y &\in [0,7] \\
0 &\leq -x/6 - y + 6.5 \\
0 &\leq -x/2 - y + 7.5 \\
0 &\leq -5x - y + 30
\end{aligned}
\tag{19}
$$

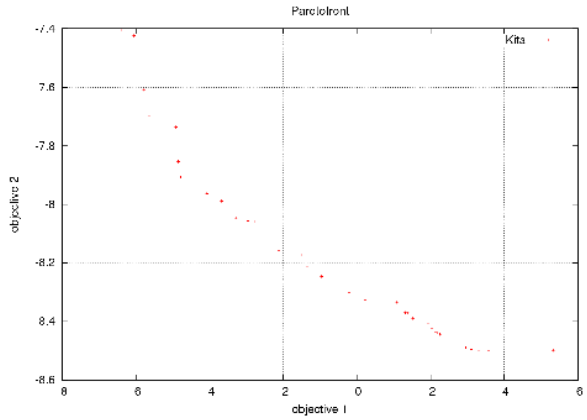

Figure 6: Reference pareto front of testfunction Kita

Testfunction Kursawe

$$
\begin{aligned}
f_1(\vec{x}), f_2(\vec{x}) &\rightarrow \min \\
f_1(\vec{x}) &= \sum_{i=1}^{n} -10e^{-0.2\sqrt{x_i^2+x_{i+1}^2}} \\
f_2(\vec{x}) &= \sum_{i=1}^{n} |x_i|^0.8 + 5\sin(x_i)^3 \\
\vec{x} &\in [-5,5]^n \quad \text{(n=3)}
\end{aligned}
\tag{20}
$$

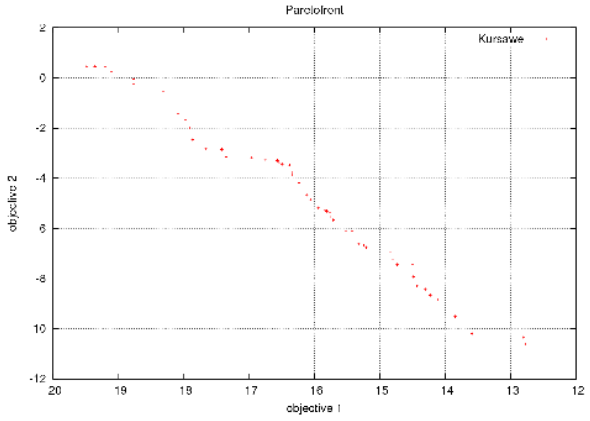

Figure 7: Reference pareto front of testfunction Kursawe

Testfunction Deb

$$\begin{aligned} f_1(x), f_2(x,y) &\rightarrow \min \\ f_1(x) &= x \\ f_2(x,y) &= (2 - e^{-(\frac{y-0.2}{0.004})^2} - 0.8e^{-(\frac{y-0.6}{0.04})^2})/x \\ x, y &\in [0.1, 1] \end{aligned} \tag{21}$$

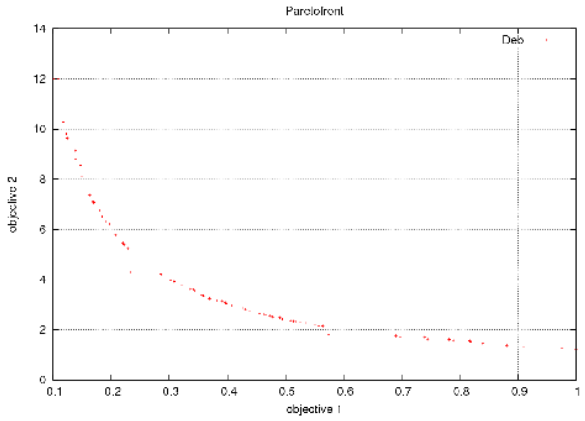

Figure 8: Reference pareto front of testfunction Deb

Testfunction Tanaka

$$
\begin{aligned}
f_1(x), f_2(y) &\rightarrow \min \\
f_1(x) &= x \\
f_2(y) &= y \\
x, y &\in (0, \pi] \\
0 &\leq x^2 + y^2 - 1 - 0.1 \cos(16 \arctan(\tfrac{x}{y})) \\
0 &\leq \tfrac{1}{2} - (x - \tfrac{1}{2})^2 - (y - \tfrac{1}{2})^2
\end{aligned}
\tag{22}
$$

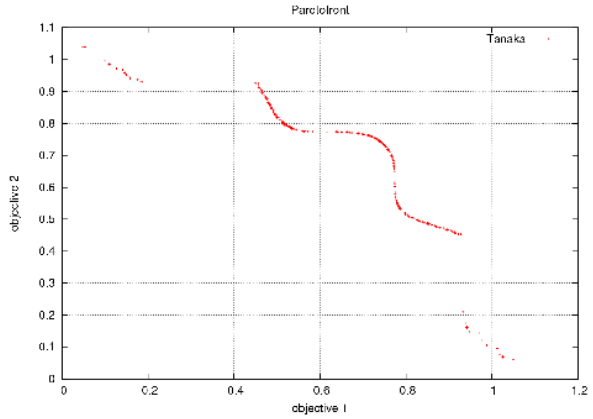

Figure 9: Reference pareto front of testfunction Tanaka

Testfunction Osyczka

$$
\begin{aligned}
f_1(\vec{x}), f_2(\vec{x}) &\rightarrow \min \\
f_1(\vec{x}) &= -(25(x_1-2)^2 + (x_2-2)^2 + (x_3-1)^2 + (x_4-4)^2 + (x_5-1)^2) \\
f_2(\vec{x}) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 \\
\vec{x} &\in \mathbb{R}^6 \\
0 &\leq x_1, x_2, x_6 \leq 10 \\
1 &\leq x_3, x_5 \leq 5 \\
0 &\leq x_4 \leq 6 \\
0 &\leq x_1 + x_2 - 2 \\
0 &\leq 6 - x_1 - x_2 \\
0 &\leq 2 - x_2 + x_1 \\
0 &\leq 2 - x_1 + 3x_2 \\
0 &\leq 4 - (x_3 - 3)^2 - x_4 \\
0 &\leq (x_5 - 3)^2 + x_6 - 4
\end{aligned}
\tag{23}
$$

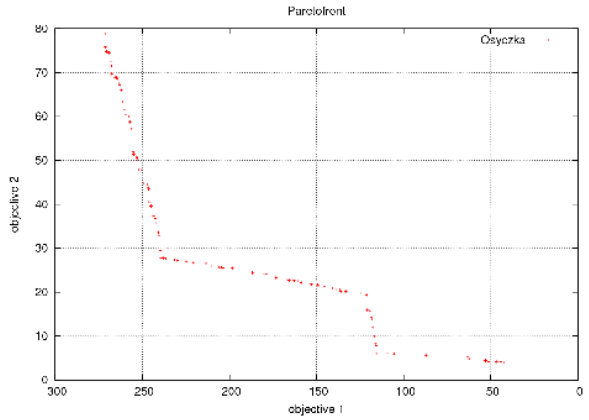

Figure 10: Reference pareto front of testfunction Osyczka

A.3 Benchmark results of singleobjective testfunctions

We compare mean $\bar{x}_{bestobj}$, median $\tilde{x}_{bestobj}$, standard deviation $s_{bestobj}$, minimum x_{min} and maximum x_{max} of best objective and the mean of calculated designs per setting for every testfunction. When comparing the results in these tables it is obvious that mutation is a very important setting in evolutionary algorithms. Mutation prevents the algorithms to stagnate too early and find suboptimal solutions as it can be seen in settings S1-S6.

Results of testfunction Rosenbrock with 10 parameter

Setting	$\bar{x}_{bestobj}$	$\tilde{x}_{bestobj}$	$s_{bestobj}$	x_{min}	x_{max}	#Designs
S1	688.138	571.513	401.789	143.310	1933.938	364.6
S2	258.107	212.920	165.508	57.039	1063.481	423.2
S3	236.111	205.362	135.836	31.110	966.716	427.6
S4	2148.240	2096.703	940.248	428.560	5310.298	200.0
S5	202.160	169.253	122.058	40.289	600.705	975.2
S6	103.426	88.725	62.410	27.080	361.327	438.4
S7	10.021	9.897	1.509	5.168	15.751	981.6
S8	14.846	10.445	15.163	6.514	74.125	919.0
S9	16.771	14.795	11.263	9.008	97.647	733.2
S10	16.200	10.275	16.912	1.788	78.693	992.4
S11	28.373	16.876	24.365	8.299	110.203	966.2
S12	101.831	95.284	35.728	22.582	217.784	652.0
S13	49.558	36.925	36.983	12.418	207.358	760.4
S14	26.094	17.869	20.566	10.551	109.726	847.4
S15	25.776	16.097	19.964	9.328	83.789	975.8
S16	28.005	16.043	24.310	8.778	109.314	968.8

Results of testfunction Griewank with 5 parameter

Setting	$\bar{x}_{bestobj}$	$\tilde{x}_{bestobj}$	$s_{bestobj}$	x_{min}	x_{max}	#Designs
S1	11.779	10.870	6.899	1.686	31.195	342.4
S2	10.721	9.529	6.488	1.959	42.725	348.4
S3	9.352	8.286	5.684	1.148	28.159	352.6
S4	51.614	52.207	19.858	7.061	100.854	200.0
S5	7.031	5.130	5.929	0.289	30.613	944.2
S6	3.916	2.713	3.542	0.537	20.469	382.2
S7	0.741	0.714	0.247	0.251	1.746	804.2
S8	0.723	0.728	0.197	0.216	1.222	634.6
S9	0.976	0.989	0.177	0.590	1.375	536.0
S10	0.484	0.454	0.228	0.108	1.169	896.6
S11	0.783	0.767	0.225	0.266	1.224	884.4
S12	3.636	3.505	1.162	1.087	7.448	473.4
S13	1.687	1.432	0.886	0.436	5.793	710.6
S14	1.144	0.984	0.524	0.404	3.307	765.8
S15	0.534	0.497	0.222	0.056	1.023	957.4
S16	0.770	0.786	0.273	0.198	1.579	878.0

Results of testfunction Rastrigin with 3 parameter

Setting	$\bar{x}_{bestobj}$	$\tilde{x}_{bestobj}$	$s_{bestobj}$	x_{min}	x_{max}	#Designs
S1	9.721	8.631	5.297	1.464	25.551	303.2
S2	10.125	9.636	4.430	2.084	22.987	293.4
S3	8.951	8.130	4.807	2.400	26.951	301.8
S4	23.876	24.438	7.074	9.307	39.694	200.0
S5	4.225	3.225	3.261	0.044	19.098	683.8
S6	7.308	6.478	3.877	0.485	22.108	320.4
S7	3.203	2.711	2.553	0.018	14.683	595.6
S8	1.776	1.188	1.663	0.009	7.996	708.2
S9	11.934	10.754	7.093	0.881	36.262	398.2
S10	0.526	0.185	0.836	0.002	5.553	757.4
S11	0.951	0.566	0.999	0.019	6.241	693.8
S12	2.390	2.340	1.009	0.410	7.817	526.6
S13	5.267	5.194	2.696	0.294	14.865	462.8
S14	4.165	3.557	2.580	0.020	11.316	542.6
S15	0.573	0.221	0.847	0.002	4.977	827.0
S16	0.767	0.409	0.874	0.009	5.628	756.2

Results of testfunction Six-hump-camel-back

Setting	$\bar{x}_{bestobj}$	$\tilde{x}_{bestobj}$	$s_{bestobj}$	x_{min}	x_{max}	#Designs
S1	-0.867	-0.932	0.185	-1.029	-0.218	281.2
S2	-0.849	-0.921	0.214	-1.031	0.089	262.0
S3	-0.833	-0.920	0.234	-1.031	0.171	262.4
S4	-0.522	-0.555	0.357	-1.017	0.290	200.0
S5	-0.966	-1.016	0.114	-1.032	-0.476	581.8
S6	-0.955	-1.011	0.121	-1.032	-0.302	361.6
S7	-1.030	-1.031	0.004	-1.032	-0.999	652.2
S8	-1.032	-1.032	0.000	-1.032	-1.031	650.4
S9	-0.957	-1.031	0.235	-1.032	-0.208	393.2
S10	-1.031	-1.031	0.001	-1.032	-1.023	617.6
S11	-1.027	-1.030	0.009	-1.032	-0.967	530.4
S12	-1.009	-1.018	0.024	-1.032	-0.925	352.2
S13	-1.023	-1.028	0.013	-1.032	-0.949	488.4
S14	-1.029	-1.031	0.005	-1.032	-1.006	614.0
S15	-1.030	-1.031	0.003	-1.032	-1.010	701.6
S16	-1.028	-1.031	0.005	-1.032	-1.000	537.8

Results of testfunction Ackley with 2 parameter

Setting	$\bar{x}_{bestobj}$	$\tilde{x}_{bestobj}$	$s_{bestobj}$	x_{min}	x_{max}	#Designs
S1	8.291	8.072	3.615	0.275	16.397	267.0
S2	8.665	8.595	3.547	1.558	18.640	265.6
S3	8.828	9.055	3.366	0.804	16.112	267.0
S4	12.589	12.883	3.836	3.093	19.882	200.0
S5	6.100	5.437	3.834	0.020	15.606	643.6
S6	2.022	1.838	2.119	0.000	10.247	415.2
S7	0.418	0.115	0.664	0.005	2.917	740.8
S8	0.094	0.056	0.125	0.002	0.708	668.2
S9	1.016	0.903	0.638	0.074	2.633	455.2
S10	0.749	0.453	0.724	0.008	3.092	642.2
S11	1.566	1.259	1.090	0.035	3.946	558.2
S12	3.979	3.839	1.536	0.216	7.883	368.2
S13	1.794	1.747	1.105	0.020	4.324	547.6
S14	1.054	0.550	1.153	0.011	5.508	652.0
S15	0.862	0.409	0.998	0.004	3.813	713.6
S16	1.564	1.343	1.187	0.054	4.242	607.0

Results of 10 bar truss problem

The optimization problem is to minimize the mass of the structure subject to displacement and stress constraints which can be described as follows.

$$\begin{aligned} disp + 0.02 &\geq 0 \\ 1.7e8 - stress &\geq 0 \end{aligned} \tag{24}$$

Setting in optiSLang 3.1	$x_{bestobj}$	#Designs
EA global	764	400
Best of EA global as start design		
EA local	658	400

Within optiSLang 3.2 we tested the default settings. Global search strategies use a randomly generated start population of a predefined size. These results can be compared with the result for EA with global search in optiSLang 3.1. The local search settings use all the same start design. PSO local search also uses constant acceleration coefficients that force the swarm to move into direction of global best with a constant rate throughout the whole optimization. SDI was first tested with default $\gamma = 10\%$ but we couldn't improve the solution because the sampling space was chosen to big in that preoptimized area. Thus, we used a smaller sampling bound of $\gamma = 1\%$ to help the algorithm search in a smaller space. That way we could improve the preoptimized solution.

Setting in optiSLang 3.2	$x_{bestobj}$	#Designs
PSO global	661	400
EA global (self adaptive)	635	400
EA global (self adaptive, hybrid crossover)	598	400
EA global (normal)	741	400
Best of EA global as start design		
EA local	585	400
PSO local	584	400
SDI ($\gamma = 10\%$)	598	400
SDI ($\gamma = 1\%$)	586	400

A.4 Benchmark results of multiobjective testfunctions

We compare the percentage of designs that dominate a pareto front with a shifted reference front (shift: 1% and 10%). In the following tables #TP is the number of designs on reference pareto front, #PD is the number of found pareto designs and #Designs is the mean of the total number of designs. #PD includes also identical designs for which reason in some settings the number can be very high but both test criteria are not fullfilled.

Results of testfunction Kita

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	10	10	40	0.08	36	103.59	0.02	0.33	206.0
M2	10	10	40	0.05	36	104.77	0.02	0.44	201.7
M3	10	10	40	0.03	36	22.15	0.60	0.99	360.9
M4	10	10	40	0.03	36	23.49	0.66	1.00	351.8
M5	10	10	40	0.03	36	15.38	0.42	0.98	311.8
M6	10	10	40	0.06	36	64.02	0.31	0.97	357.9
M7	10	10	40	0.10	36	63.06	0.22	0.96	341.9
M8	10	10	40	0.04	36	19.23	0.64	1.00	351.8
M9	10	10	40	0.07	36	59.08	0.31	0.99	364.6
M10	10	10	40	0.11	36	66.73	0.22	0.97	342.5

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	50	50	200	0.60	36	432.75	0.05	0.82	752.5
M2	50	50	200	0.49	36	428.81	0.08	0.88	749.5
M3	50	50	200	0.17	36	30.29	0.90	1.00	1452.5
M4	50	50	200	0.17	36	45.33	0.96	1.00	1372.0
M5	50	50	200	0.07	36	31.76	0.55	0.97	1213.0
M6	50	50	200	0.55	36	349.76	0.74	1.00	1775.0
M7	50	50	200	0.33	36	370.53	0.64	1.00	1724.5
M8	50	50	200	0.11	36	25.41	0.85	1.00	1494.0
M9	50	50	200	2.63	36	313.99	0.78	1.00	2253.5
M10	50	50	200	0.30	36	364.61	0.62	1.00	1651.0

Results of testfunction Kursawe

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	10	10	40	0.07	59	108.17	0.01	0.25	236.3
M2	10	10	40	0.09	59	107.36	0.01	0.33	229.1
M3	10	10	40	0.05	59	30.65	0.45	0.93	371.5
M4	10	10	40	0.03	59	32.02	0.45	0.88	345.1
M5	10	10	40	0.04	59	27.21	0.23	0.52	302.5
M6	10	10	40	0.09	59	62.89	0.38	0.96	383.0
M7	10	10	40	0.12	59	63.82	0.25	0.98	371.5
M8	10	10	40	0.06	59	22.82	0.29	0.95	358.8
M9	10	10	40	0.12	59	51.15	0.39	0.98	381.5
M10	10	10	40	0.13	59	60.80	0.34	0.98	377.9

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	50	50	200	0.39	59	524.94	0.06	0.82	932.0
M2	50	50	200	0.38	59	510.19	0.09	0.84	922.5
M3	50	50	200	0.07	59	39.92	0.66	1.00	1778.5
M4	50	50	200	0.10	59	67.73	0.83	1.00	1567.5
M5	50	50	200	0.18	59	54.08	0.53	0.69	1407.0
M6	50	50	200	0.66	59	315.93	0.92	1.00	1797.0
M7	50	50	200	0.95	59	331.50	0.87	1.00	2037.0
M8	50	50	200	0.41	59	25.41	0.36	1.00	1643.5
M9	50	50	200	0.36	59	257.65	0.93	1.00	1923.5
M10	50	50	200	2.17	59	325.99	0.87	1.00	2040.5

Results of testfunction Deb

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	10	10	40	0.14	81	121.78	0.01	0.12	214.2
M2	10	10	40	0.08	81	119.16	0.03	0.15	195.9
M3	10	10	40	0.05	81	39.67	0.14	0.50	251.4
M4	10	10	40	0.08	81	52.32	0.14	0.37	259.8
M5	10	10	40	0.01	81	51.41	0.10	0.24	186.9
M6	10	10	40	0.05	81	87.97	0.28	0.80	324.9
M7	10	10	40	0.09	81	88.45	0.23	0.83	334.2
M8	10	10	40	0.04	81	34.64	0.17	0.58	270.6
M9	10	10	40	0.14	81	87.90	0.37	0.89	346.0
M10	10	10	40	0.06	81	90.51	0.21	0.84	327.7

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	50	50	200	0.75	81	517.68	0.14	0.60	806.0
M2	50	50	200	0.59	81	513.35	0.13	0.59	820.0
M3	50	50	200	0.06	81	52.21	0.35	0.82	1015.0
M4	50	50	200	0.11	81	108.42	0.35	0.71	1034.5
M5	50	50	200	0.19	81	139.64	0.25	0.42	878.5
M6	50	50	200	0.55	81	494.16	0.83	0.99	1580.5
M7	50	50	200	0.39	81	468.35	0.82	0.99	1424.0
M8	50	50	200	0.13	81	48.77	0.34	0.85	1082.5
M9	50	50	200	0.48	81	405.16	0.93	1.00	1554.5
M10	50	50	200	0.69	81	462.21	0.82	0.99	1457.5

Results of testfunction Tanaka

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	10	10	40	0.07	231	98.24	0.00	0.05	232.5
M2	10	10	40	0.09	231	96.22	0.00	0.05	226.5
M3	10	10	40	0.02	231	16.02	0.13	0.75	338.4
M4	10	10	40	0.02	231	16.61	0.13	0.67	321.6
M5	10	10	40	0.02	231	12.32	0.06	0.47	287.9
M6	10	10	40	0.05	231	38.44	0.03	0.48	379.1
M7	10	10	40	0.06	231	44.05	0.02	0.45	365.1
M8	10	10	40	0.04	231	14.09	0.07	0.77	327.6
M9	10	10	40	0.06	231	29.79	0.04	0.53	386.6
M10	10	10	40	0.06	231	41.80	0.02	0.44	376.8

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	50	50	200	0.43	231	386.50	0.01	0.22	804.5
M2	50	50	200	0.61	231	399.13	0.01	0.19	804.0
M3	50	50	200	0.09	231	26.38	0.21	0.95	1522.0
M4	50	50	200	0.06	231	32.72	0.33	0.89	1371.0
M5	50	50	200	0.19	231	26.33	0.20	0.54	1347.5
M6	50	50	200	1.03	231	298.07	0.10	0.83	1973.0
M7	50	50	200	0.53	231	295.75	0.08	0.76	1777.5
M8	50	50	200	0.11	231	21.21	0.12	0.94	1427.0
M9	50	50	200	2.83	231	273.96	0.16	0.88	2237.5
M10	50	50	200	0.59	231	301.48	0.08	0.78	1849.0

Results of testfunction Osyczka

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	100	100	100	3.27	80	1239.69	0.00	0.01	2574.0
M2	100	100	100	3.28	80	1239.20	0.00	0.01	2574.0
M3*	100	100	100	1.02	80	56.33	0.17	0.68	9852.0
M4*	100	100	100	1.13	80	52.61	0.27	0.73	9864.0
M5	100	100	100	6.17	80	45.29	0.16	0.39	8480.0
M6	100	100	100	5.28	80	789.35	0.54	0.87	9979.0
M7	100	100	100	19.35	80	931.84	0.47	0.79	9837.0
M8*	100	100	100	4.85	80	34.36	0.08	0.66	9703.0
M9	100	100	100	9.00	80	447.10	0.62	0.92	10000.0
M10	100	100	100	24.09	80	945.50	0.50	0.82	9950.0

Setting	μ	λ	g_{max}	time (sek)	#TP	#PD	#DP1	#DP10	#Designs
M1	50	50	200	2.42	80	613.02	0.00	0.00	1325.0
M2	50	50	200	1.73	80	601.37	0.00	0.00	1360.0
M3*	50	50	200	0.23	80	57.98	0.24	0.62	8597.5
M4*	50	50	200	0.25	80	57.27	0.29	0.69	8580.5
M5	50	50	200	3.80	80	33.18	0.10	0.35	4927.5
M6	50	50	200	12.54	80	792.21	0.47	0.76	9626.5
M7	50	50	200	22.71	80	893.64	0.40	0.71	8154.5
M8*	50	50	200	5.20	80	27.62	0.08	0.49	7245.5
M9	50	50	200	22.51	80	517.60	0.53	0.84	9827.5
M10	50	50	200	85.24	80	923.80	0.41	0.72	8068.5

Benchmark of mutation methods

EA with multipoint crossover

- AM1 self adaptive mutation (M7)
- AM2 constraint mutation
- AM3 normal mutation (M6)
- AM4 modulated mutation

Test	Setting	μ	λ	g_{max}	time	#TP	#PD	#DP1	#DP10	#Designs
Kita	AM1	10	10	40	0.08	36	67.60	0.21	0.99	327.0
Kita	AM2	10	10	40	0.09	36	59.00	0.35	0.98	347.4
Kita	AM3	10	10	40	0.11	36	57.56	0.31	0.98	362.6
Kita	AM4	10	10	40	0.21	36	33.30	0.21	1.00	299.6
Tanaka	AM1	100	100	100	3.52	231	591.22	0.11	0.87	3202.0
Tanaka	AM2	100	100	100	22.10	231	603.98	0.24	0.93	4384.0
Tanaka	AM3	100	100	100	2.46	231	634.18	0.17	0.93	4264.0
Tanaka	AM4	100	100	100	9.46	231	10.80	0.14	0.93	3160.0
Osyczka	AM1	100	100	100	1.37	80	368.30	0.41	0.78	6178.0
Osyczka	AM2	100	100	100	20.47	80	324.64	0.50	0.80	7786.0
Osyczka	AM3	100	100	100	2.09	80	346.84	0.47	0.82	6998.0
Osyczka	AM4	100	100	100	33.72	80	366.80	0.07	0.37	3294.0

Table 3: Benchmark of different mutation methods for EA with multipoint crossover