



Draft Manuscript for Review

**CREATING TEST FUNCTIONS FOR THE EVALUATION OF
OPTIMIZATION TECHNIQUES, APPLICATION FOR MAIN
GIRDER DESIGN OF AN OVERHEAD CRANE**

Journal:	<i>Structural and Multidisciplinary Optimization</i>
Manuscript ID:	SMO-14-0240
Manuscript Type:	Research Paper
Date Submitted by the Author:	30-Jul-2014
Complete List of Authors:	Marcsak, Gabor; University of Miskolc, Institute of Logistics Barcsak, Csaba; Evosoft, Jarmai, Karoly; University of Miskolc, Institute of Logistics
Keywords:	evolutionary optimization, benchmarking, test functions, main girder , overhead crane

SCHOLARONE™
Manuscripts

view

Dear Prof. G.I.N. Rozvany,

We would like to have our paper entitled **CREATING TEST FUNCTIONS FOR THE EVALUATION OF OPTIMIZATION TECHNIQUES, APPLICATION FOR MAIN GIRDER DESIGN OF AN OVERHEAD CRANE** by **Gábor Zoltán Marcsák, Csaba Barcsák, Károly Jármái*** (*corresponding author Tel: +36-46-565111, Fax:+36-46-563399, e-mail: altjar@uni-miskolc.hu) University of Miskolc, Miskolc, Hungary, altjar@uni-miskolc.hu, marcsakgabor@gmail.com, csaba.barcsak@gmail.com) published in your journal “Structural and Multidisciplinary Optimization”. It contains very recent results on this field, several evolutionary optimization techniques, the procedure, how to build test functions for algorithm testing and an example of the crane girder optimization.

We are looking forward to your positive answer, we remain

Yours sincerely

Miskolc, July 30. 2014.

Károly Jármái
Gábor Zoltán Marcsák
Csaba Barcsák

CREATING TEST FUNCTIONS FOR THE EVALUATION OF OPTIMIZATION TECHNIQUES, APPLICATION FOR MAIN GIRDER DESIGN OF AN OVERHEAD CRANE

Gábor Zoltán Marcsák¹, Csaba Barcsák², Károly Jármai³,

¹ MSc. student, *University of Miskolc*, ² software engineer, *Evosoft Ltd.*, ³ Professor,
University of Miskolc, H-3515 Miskolc, Egyetemváros, Hungary

Abstract

Heuristic algorithms have been applied successfully for solving difficult optimization problems in engineering. We have implemented a large number of evolutionary optimization techniques. Their efficiency depends greatly on the problem to be solved. Using synthetic test functions for the benchmarking of algorithms has many advantages, for example the complexity of the problem is adjustable, and the position of global optima is known. However real-world design problems can often provide enhanced insight on the optimization techniques' performance. A new synthetic test function generator has been developed to compare different optimization techniques. The generated test functions help to evaluate and compare their performance for further development. As for real-world design problem, we determined the optimal dimensions of an overhead crane's main girder, which is a non-linear, constrained optimization problem.

Keywords: evolutionary optimization, benchmarking, test functions, main girder, overhead crane

1. INTRODUCTION

It is hard to select which one is the most efficient optimization algorithm. Several novel evolutionary techniques have appeared recently. When a new heuristic algorithm is described, its performance should be compared objectively to other algorithms. There are a great number of test functions available to make benchmark

tests. The process of optimization is finding the best solution to a given problem, when the amount of available resources is often restricted. Despite the rapid development of computer science, most optimization problems cannot be solved by evaluating all feasible solutions. For example, the class of NP-hard problems, such as the Traveling salesman problem (also known as TSP), might have an enormously large search space, which requires exponential computation time to be fully explored. Heuristic algorithms can find approximate solutions, even when the search space is excessively huge. In this paper, fifteen optimization techniques have been benchmarked, to compare their efficiency in finding the global minima of different continuous mathematical test functions. Mathematical function optimization is very important, because most real world optimization problems can be modelled in this general framework. Numerous mathematical test functions can be found in the literature. Additionally, we created a software solution, which utilizes a novel method to construct customized test functions.

2. BENCHMARK PROBLEMS

As mentioned before, a lot of mathematical test functions can be found in the literature, for example in the paper of Mologa and Smutnicki (2014). The complexity of test functions is determined by the number of variables and the number and distribution of local extremes. Continuous test functions have been studied with two variables, since those problems can be plotted as 3D surfaces. Table **Table 1** summarizes the ten test functions we used in alphabetical order. The Ackley's function has a nearly flat outer region, and a large valley at its centre. This widely used multimodal test function can easily trap heuristic algorithms at one of its local optima. De Jong's function is a very simple, convex, unimodal benchmark problem. Drop-Wave function is very complex, with expanding ripples, like when an object is dropped into liquid surface. Easom's function is unimodal like De Jong's, however more complicated, because the global optima is relatively small, compared to the search space. Griewangk's function looks similar to De Jong's either, but it has a rugged surface with many regularly distributed local optima.

Matyas's function is a plate shaped problem, it doesn't have any local extremes, only the global one, which is relatively easy to find. However, convergence to the global optima is difficult, so that is a great benchmark problem to measure the accuracy and convergence rate of search algorithms. Rastrigin's function, also known as egg holder is a widely used, highly multimodal problem with regularly distributed local extremes. Rosenbrock's valley is unimodal, the global minimum can be found in a narrow, parabolic valley. Schaffer's second function is an extremely noisy optimization problem, with a lot of local optima very close to each other. Last but not least, Three-hump camelback looks like Rosenbrock's valley, however it has two local extremes.

Table 1 Numerical data of benchmark problems

Name	Definition	Search range and global optimum
Ackley's function (F1)	$f(x) = -20 \times \exp(-0.2 \times \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i^2)}) - \exp(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)) + 20 + \exp(1)$	$-32.768 \leq x_i \leq 32.768$ $x_i = 0, i = 1, \dots, n$ $f(x)=0$
De Jong's function (F2)	$f(x) = \sum_{i=1}^n x_i^2$	$-5.12 \leq x_i \leq 5.12$ $x_i = 0, i = 1, \dots, n$ $f(x)=0$
Drop-Wave function (F3)	$f(x) = -\frac{1 + \cos(12 \times \sqrt{x_1^2 + x_2^2})}{\frac{1}{2}(x_1^2 + x_2^2) + 2} + 1$	$-5.12 \leq x_i \leq 5.12$ $x_i = 0, i = 1, 2$ $f(x)=0$
Easom's function (F4)	$f(x) = -\cos(x_1) \times \cos(x_2) \times \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2) + 1$	$-10 \leq x_i \leq 10$ $x_i = \pi, i = 1, 2$ $f(x)=0$
Griewangk's function (F5)	$f(x) = \frac{1}{4000} \times \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$ $x_i = 0, i = 1, \dots, n$ $f(x)=0$
Matyas's function (F6)	$f(x) = 0.26 \times (x_1^2 + x_2^2) - 0.48 \times x_1 \times x_2$	$-10 \leq x_i \leq 10$ $x_i = 0, i = 1, 2$ $f(x)=0$

Rastrigin's function (F7)	$f(x) = 10 \times n + \sum_{i=1}^n [x_i^2 - 10 \times \cos(2\pi x_i)]$	$-5.12 \leq x_i \leq 5.12$ $x_i = 0, i = 1, \dots, n$ $f(x)=0$
Rosenbrock's valley (F8)	$f(x) = \sum_{i=1}^{n-1} [100 \times (x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$	$-2.048 \leq x_i \leq 2.048$ $x_i = 1, i = 1, \dots, n$ $f(x)=0$
Schaffer's N. 2. function (F9)	$f(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001 \times (x_1^2 + x_2^2)]^2}$	$-10 \leq x_i \leq 10$ $x_i = 0, i = 1, 2$ $f(x)=0$
Three-hump camelback (F10)	$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$	$-2.048 \leq x_i \leq 2.048$ $x_i = 0, i = 1, 2$ $f(x)=0$

3. COMPOSITION OF TEST FUNCTIONS

In Liang et al. (2005) the authors proposed a novel theoretical method to create more complex test functions from given basis functions. A practical framework has been described by Barcsák and Jármai (2013), which extends the above theory. A software solution has been developed, which utilizes the practical framework to easily create customized arbitrarily difficult test problems.

3.1 THEORETICAL METHOD

In order to generate arbitrarily difficult complex test functions, the algorithm requires several input parameters:

- $[X_{\min}, X_{\max}]^D$: search range of the complex function
- D : number of dimensions
- $f_i(\theta)$: list of basic functions
- $[x_{\min}, x_{\max}]^D$: search range of the basic functions
- o_i : the position of the global optima for the i-th basic function
- bias (ψ): a vector to define the global optima. It allows the user to shift the optimum values of the basic functions.

To define the position of global and local optimums, we have to shift the value of the basic function's global optimum points. In order to achieve this, the basic functions have to be evaluated outside the defined search range. Therefore, the given global optimum positions of the basic functions have to be independent of the search range. The complex test functions can be determined using the following formulas:

$$F(\theta) = \sum \left[w_i \left(\frac{f_i(\vartheta_i)}{f_i(o_i)} + bias_i \right) \right] \quad (1)$$

$$\vartheta_i = \frac{\theta - (X_{min} - x_{min_i} \kappa_i + \psi_i)}{\kappa_i} \quad (2)$$

Where w_i is the weighting function, which ensures to keep the predefined optimum positions and values. The closer the solver gets to a basic functions global optimum position (o_i), the bigger weighting coefficient it gets. At the same time, the other basic functions get smaller weighting coefficients. Three different types of weighting functions have been used.

3.1.1 EUCLIDEAN DISTANCE-BASED WEIGHTING

The first weighting function is based on the Euclidean distance between the complex function's given point (ϑ) and the optimum points (o_i) of the basic functions.

$$w_i = \text{Euclidean_distance}(\vartheta_i, o_i) \quad (3)$$

The normalized distances are as follows:

$$w_i = \frac{w_i}{\sum w} \quad (4)$$

$$w_i = 1 - w_i \quad (5)$$

If $w_i \neq \text{maximumDistance}(w)$, and n is the number of basic functions:

$$w_i = \frac{1 - \text{maximumDistance}(w)}{(n - 1)} \quad (6)$$

3.1.2 GAUSSIAN WEIGHTING

Smoother edges can be achieved by using the Gaussian functions to determine weighting coefficients.

$$w_i = e^{\frac{(\vartheta_i - o_i)^2}{6}} \quad (7)$$

3.1.3 GABOR-LIKE WEIGHTING

More difficult optimization problems can be created if the weighting function generates noise. However we have to make sure not to shift the original optimum values. The weighting function should return values between 0 and 1, its global maxima should be at the complex function's global optimum.

$$w_i = \left| \prod_k \cos(\vartheta_i(k)\tau_1) e^{-\frac{\sum_k \vartheta_i(k)^2}{\tau_2}} \right| \quad (8)$$

Where $\vartheta_i(k)$ is the k -th element of the ϑ_i vector, τ_1 is the noisiness parameter, and τ_2 is the convergence range. Increasing the τ_1 parameter results in more noise. If $w_i \neq \text{maximumWeight}(w)$, and n is the number of basic functions:

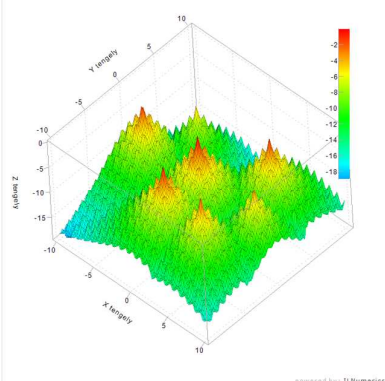
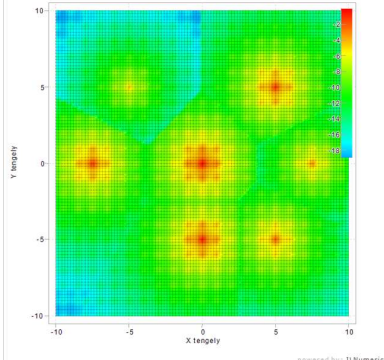
$$w_i = \frac{1 - \text{maximumWeight}(w)}{(n - 1)} \quad (9)$$

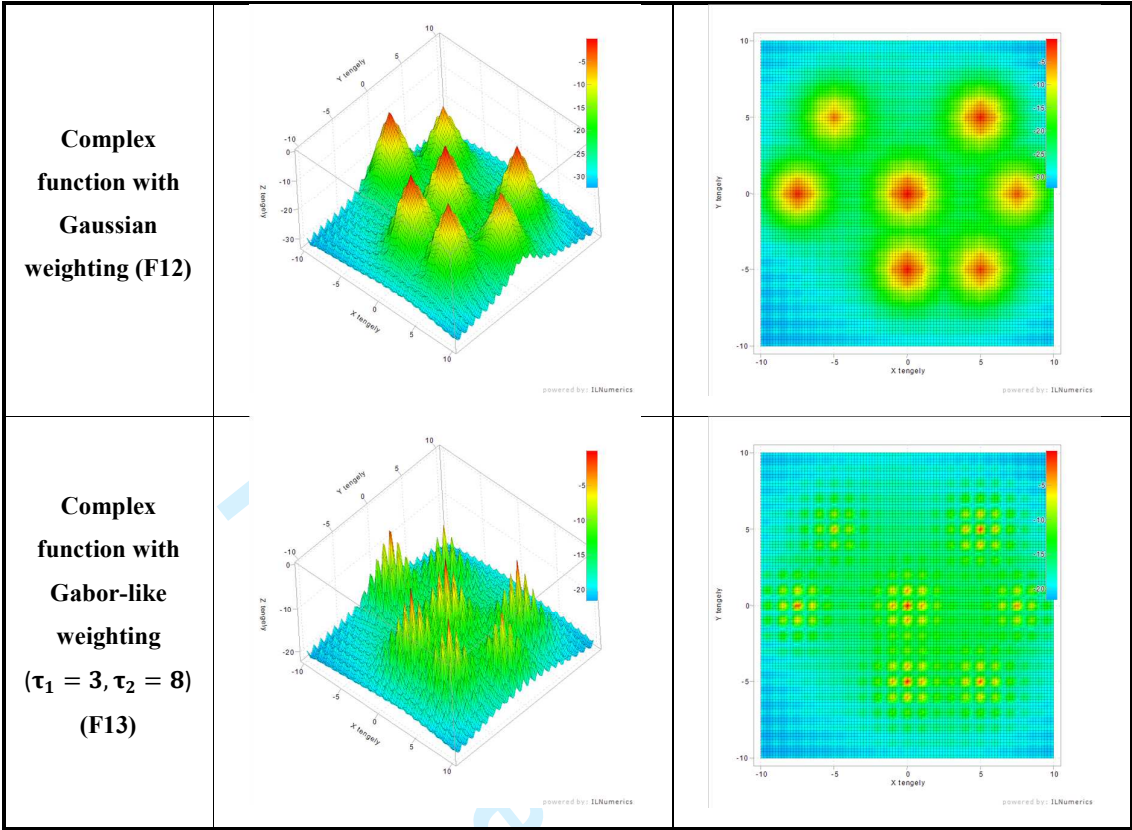
3.2 PRACTICAL EXAMPLE

A complex test function has been generated to demonstrate the method and expand the number of benchmark problems used in this paper. The input parameters are the following:

- $[X_{min}, X_{max}]^D$: the search range of the complex function is $[-10, 10]^D$
- D : the number of dimensions is 2
- $f_i(\theta)$: 7 basic functions have been used, all of them are Ackey's functions (F1).
- $[x_{min}, x_{max}]^D$: since we shift the optimum points of the basic functions, we don't have to define the basic functions' search range.
- $o(x)$: the x coordinates for the basic functions: $[7.5; 5; 5; 0; -7.5; -5; 0]$
- $o(y)$: the y coordinates for the basic functions: $[0; 5; -5; -5; 0; 5; 0]$
- bias (ψ): a vector to shift the basic functions: $[3; 1; 2; 0.5; 1; 4.5; 0]$

Table 2 Surface and contour plot of complex benchmark problems

Name	Surface plot	Contour plot
Complex function with Euclidean distance-based weighting (F11)		



In Table 2 you can see the generated Complex functions, which all consists of 7 Ackley's basic functions. The different characteristics of the three weighting functions are also observable. The basic functions global optimum values have been shifted, therefore the complex function's global optimum value is $f(x) = 0$, can be found at $x = 0, y = 0$ coordinates.

3.3 NOVEL SOFTWARE SOLUTION TO EASILY GENERATE COMPLEX TEST FUNCTIONS

A software solution has been developed which utilizes the above framework to easily create customized arbitrarily difficult test problems. All the coding was done in C#, the program uses the .NET Framework. The best feature of the software is the automatic source code generation. As the user builds the complex function in a visual editor, at the same time the program generates the function's C# source code. Therefore, the newly created optimization problem can be used straight away.

4. MAIN CHARACTERISTICS OF HEURISTIC ALGORITHMS

As mentioned before, the biggest advantage of heuristic algorithms is that they can find approximate solutions, even when the search space is excessively huge. However, finding the global optimum cannot be guaranteed, since they don't evaluate all feasible solutions. With the utilization of heuristic algorithms, we obtain computing speed, but we may have to pay with accuracy. A good heuristic algorithm has to maintain balance between local search and global search. On one hand, it has to explore the entire search space properly, on the other hand search near the current best positions efficiently. In other words, quickly find regions with quality solutions, and don't waste too much time in low quality areas. Most of the time, heuristic algorithms have stochastic behaviour. Ideally, the final solutions, through slightly different, will converge to the optimal solution of the given problem. However, the way heuristic algorithms get to the solution is always a bit different because of the stochastic factor. Nowadays a lot of nature inspired heuristic algorithms emerge based on the behaviour of biological and physical systems. We benchmarked fifteen optimization techniques, like evolutionary (Cultural Algorithm, Differential Evolution, Memetic Algorithm), physical (Harmony Search, Simulated Annealing), and swarm intelligence (Bacterial Foraging, Bat Algorithm, Bees Algorithm, Cuckoo Search, Firefly, Particle Swarm) optimization methods. We made the algorithms' source code available on the Internet. The algorithms search for the global minima of the Rastrigin's function in the source code (Marcsák and Jármai 2014).

4.1 BENCHMARKED HEURISTIC ALGORITHMS AND THEIR PSEUDOCODES

Bacterial Foraging Optimization Algorithm (BFOA) was first described by Liu and Passino (2002). It's a relatively new swarm intelligence search algorithm. These techniques use the collective intelligence of numerous homogenous individuals. In principle, an individual entity may not be able to solve a problem on its own.

However, if a large number of individuals form a group, the group's collective intelligence may be enough to solve the task. Bacterial Foraging is based on the foraging and reproduction behaviour of E. Coli bacteria colonies. The chemotaxis movement of the group tends toward precious nutrients (global optima), while avoiding dangerous places (local optima) in the environment (Fig.1).

```
Input: Problemsize, Cellsnum, Ned, Nre, Nc, Ns, Stepsize, Ped
Output: Cellbest
1. Population ← InitializePopulation(Cellsnum, 1 Problemsize);
2. for l = 0 to Ned do
3.     for k = 0 to Nre do
4.         for j = 0 to Nc do
5.             ChemotaxisAndSwim(Population, Problemsize, Cellsnum, Ns, Stepsize);
6.             foreach Cell ∈ Population do
7.                 if Cost(Cell) ≤ Cost(Cellbest) then
8.                     Cellbest ← Cell;
9.                 end
10.            end
11.        end
12.        SortByCellHealth(Population);
13.        Selected ← SelectByCellHealth(Population, Cellsnum/2);
14.        Population ← Selected;
15.        Population ← Selected;
16.    end
17.    foreach Cell ∈ Population do
18.        if Rand() ≤ Ped then
19.            Cell ← CreateCellAtRandomLocation();
20.        end
21.    end
22. end
23. return Cellbest;
```

Fig. 1 Pseudocode of Bacterial Foraging Algorithm (BFOA) by Brownlee (2011)

Bat Algorithm (BATA) is a swarm intelligence heuristic method published by Yang (2010), based on the echolocation behaviour of bats. Echolocation allows bats to find and identify their prey (global optima), and avoid obstacles (local optima), even in complete darkness. Echolocation works like a sonar, as bats emit a sound pulse and listen for the echo that bounces back from the surrounding environment. During the search, frequency, loudness and emission rate of the individual's sound changes regarding the position of the prey, which informs other bats. The loudness of the emitted sound decreases as the bat approaches the prey, while the rate of pulse emission increases. Bat Algorithm combines the advantages of existing swarm intelligence methods, such as Particle Swarm Optimization and Firefly Algorithm (Fig.2).

```

Input: PopulationSize, ProblemSize, Loudness, PulseRate, Frequency
Output: best
1. Initialize the bat population  $x_i$  ( $i = 1, 2, \dots, \text{populationSize}$ ) and  $v_i$ 
2. Define pulse Frequency  $f_i$  at  $x_i$ 
3. Initialize PulseRates  $r_i$  and the Loudness  $A_i$ 
4. while ( $t < \text{Max number of iterations}$ )
5.   Generate new solutions by adjusting frequency,
6.   and updating velocities and locations/solutions
7.   if ( $\text{rand} > r_i$ )
8.     Select a solution among the best solutions
9.     Generate a local solution around the selected best solution
10.   end if
11.   Generate a new solution by flying randomly
12.   if ( $\text{rand} < A_i \ \& \ f(x_i) < f(x^*)$ )
13.     Accept the new solutions
14.     Increase  $r_i$  and reduce  $A_i$ 
15.   end if
16. Rank the bats and find the current best  $x^*$ 
17. end while
18. return best

```

Fig. 2 Pseudocode of Bat Algorithm (BATA)

Bees Algorithm (BA) was published by Pham et al. (2005). Primarily it was developed to search for the global optima of continuous mathematical functions. It belongs to the field of swarm intelligence procedures. The Bees Algorithm, as the name suggests, was inspired by the foraging behaviour of honey bees. Bee colonies send scout bees to explore the environment, and find areas rich of nectar (global or local optima). The scout bees then return to the hive and inform the worker bees about the position and quality of food sources. The number of worker bees sent to each food source depends on these characteristics. The scout bees continually search for good sites, while worker bees accurately explore the already found spots. Generally speaking, the scout bees are responsible for global search, while worker bees provide good local search. The algorithm is similar to Ant Colony Optimization and Particle Swarm optimization, however the rank system of bees makes it unique. The Bees Algorithm is capable to solve both continuous and combinatorial optimization problems (Fig.3).

```
Input: Problemsize, Beesnum, Sitesnum, EliteSitesnum, PatchSizeinit, EliteBeesnum, OtherBeesnum
Output: Beebest
1. Population ← InitializePopulation(Beesnum, 1 Problemsize);
2. while ¬StopCondition() do
3.   EvaluatePopulation(Population);
4.   Beebest ← GetBestSolution(Population);
5.   NextGeneration ← ∅;
6.   Patchsize ← ( PatchSizeinit × PatchDecreasefactor);
7.   Sitesbest ← SelectBestSites(Population, Sitesnum);
8.   foreach Sitei ∈ Sitesbest do
9.     RecruitedBeesnum ← ∅;
10.    if i < EliteSitesnum then
11.      RecruitedBeesnum ← EliteBeesnum;
12.    else
13.      RecruitedBeesnum ← OtherBeesnum;
14.    end
15.    Neighborhood ← ∅;
16.    for j to RecruitedBeesnum do
17.      Neighborhood ← CreateNeighborhoodBee(Sitei, Patchsize);
18.    end
19.    NextGeneration ← GetBestSolution(Neighborhood);
20.  end
21.  RemainingBeesnum ← (Beesnum- Sitesnum);
22.  for j to RemainingBeesnum do
23.    NextGeneration ← CreateRandomBee();
24.  end
25.  Population ← NextGeneration;
26. end
27. return Beebest;
```

Fig. 3 Pseudocode of Bees Algorithm (BA) by Brownlee (2011)

Cross-Entropy Method (CEM) is a probabilistic optimization algorithm developed by Rubinstein (1997). The name of the technique comes from the Kullback-Leibler cross-entropy divergence, as a measure of closeness between two probability distributions. Cross-Entropy Method is an adaptive importance estimation technique for rare-event probabilities in discrete event simulation systems. Optimization problems can be described as rare-event systems, because the probability of locating an optimal solution by using random search is a rare-event probability. The method changes the sampling distribution of random search, so that the rare-event (finding the optima) is more likely to occur (Fig.4).

```

Input: Problemsize, Samplesnum, UpdateSamplesnum, Learnrate
Output: Sbest
1. Means ← InitializeMeans(); Variances ← InitializeVariances(); Sbest ← ∅;
2. while Max(Variances) ≤ Variancecmin do Samples ← ∅;
3.   for i = 0 to Samplesnum do
4.     Samples ← GenerateSample(Means, Variances);
5.   end
6.   EvaluateSamples(Samples); SortSamplesByQuality(Samples);
7.   if Cost(Samples0) ≤ Cost(Sbest) then Sbest ← Samples0;
8.   end
9.   Samplesselected ← SelectBestSamples(Samples, UpdateSamplesnum);
10.  for i = 0 to Problemsize do
11.    Meansi ← Meansi + Learnrate × Mean(Samplesselected, i);
12.    Variancesi ← Variancesi + Learnrate ×
13.      Variance(Samplesselected, i);
14.  end
15. end
16. return Sbest;

```

Fig. 4 Pseudocode of Cross-Entropy Method (CEM) by Brownlee (2011)

Cuckoo Search Algorithm (CS) was developed by Yang and Deb (2009). The algorithm was inspired by the obligate brood parasitism of certain cuckoo species, which means they lay their eggs in the nests of other host species. Sometimes if the host birds discover the eggs are not their own, they throw the alien eggs away, or simply abandon the current nest and build a new elsewhere. The algorithm randomly distributes a fixed number of nests through the search space. The cuckoos lay down their eggs, each cuckoo lays one egg at a time in a randomly chosen nest. If the egg is discovered by the host bird, it can either throw the egg away, or abandon the current nest and build a completely new elsewhere. Each egg in a nest represents a solution, and a cuckoo egg represents a new solution. The aim of the algorithm is to find new, potentially better solutions (global optima) to replace the worse (local optima). The best nests with high quality of eggs will be members of the next generation (Fig.5).

```

Input: Problemsize, NestSize, DiscoveryRate
Output: best
1. begin
2.   Generate initial population of NestSize nests xi (i = 1, 2, ..., NestSize)
3.   while (t < MaxGeneration) or (stop criterion)
4.     Get a cuckoo randomly by L'evy flights
5.     evaluate its quality/fitness Fi
6.     Choose a nest among n (say, j) randomly
7.     if (Fi > Fj), replace j by the new solution; end
8.     Worse nests < DiscoveryRate are abandoned and new ones are built;
9.     Keep the best solutions (or nests with quality solutions);
10.    Rank the solutions and find the current best
11.  end while
12. end
13. return best

```

Fig. 5 Pseudocode of Cuckoo Search Algorithm (CS)

Cultural Algorithm (CA) was described by Reynolds (1994). This evolutionary algorithm simulates the cultural evolution of human society. Culture includes habits, beliefs and morals of a member of the society. Culture can interact with the environment via positive or negative feedback cycles. As the evolutionary process goes on, individuals gain information about the search space, which is communicated to other individuals in the population. This creates a knowledge base that stores positive feedback about useful areas of the environment (global optima), as well as potentially hazardous areas (local optima). This cultural knowledge base is expanded and exploited through generations as situations change (Fig.6).

```

Input: Problemsize, Populationnum
Output: KnowledgeBase
1. Population ← InitializePopulation(Problemsize, Populationnum);
2. KnowledgeBase ← InitializeKnowledgebase(Problemsize, Populationnum);
3. while ~StopCondition() do
4.   Evaluate(Population);
5.   SituationalKnowledgecandidate ← AcceptSituationalKnowledge(Population);
6.   UpdateSituationalKnowledge(KnowledgeBase, SituationalKnowledgecandidate);
7.   Children ← ReproduceWithInfluence(Population, KnowledgeBase);
8.   Population ← Select(Children, Population);
9.   NormativeKnowledgecandidate ← AcceptNormativeKnowledge(Population);
10.  UpdateNormativeKnowledge(KnowledgeBase, NormativeKnowledgecandidate);
11. end
12. return KnowledgeBase;

```

Fig. 6 Pseudocode of Cultural Algorithm (CA) by Brownlee (2011)

Differential Evolution (DE) algorithm was developed by Storn and Price (1995), and belongs to the field of evolutionary algorithms. Differential Evolution is mainly

based on Darwin's Theory of Evolution, because its main principle is natural selection. The algorithm maintains a population of candidate solutions with recombination, evaluation and selection as generations unfold. The recombination creates a new candidate solution based on the weighted difference between two randomly selected population members added to a third population member (Fig.7).

```

Input: Populationsize, Problemsize, Weightingfactor, Crossoverrate
Output: Sbest
1. Population ← InitializePopulation(Populationsize, Problemsize);
2. EvaluatePopulation(Population);
3. Sbest ← GetBestSolution(Population);
4. while ¬ StopCondition() do
5.     NewPopulation ← ∅;
6.     foreach Pi ∈ Population do
7.         Si ← NewSample(Pi, Population, Problemsize, Wf, Cr);
8.         if Cost(Si) ≤ Cost(Pi) then
9.             NewPopulation ← Si;
10.        else
11.            NewPopulation ← Pi;
12.        end
13.    end
14.    Population ← NewPopulation;
15.    EvaluatePopulation(Population);
16.    Sbest ← GetBestSolution(Population);
17. end
18. return Sbest;

```

Fig. 7 . Pseudocode of Differential Evolution (DE) by Brownlee (2011)

Firefly Algorithm (FF) is a nature-inspired heuristic optimization algorithm for multimodal optimization developed by Yang (2009). The algorithm was inspired by the flashing behaviour of fireflies, in order to communicate with other fireflies. The flashing light is produced by a biological process called bioluminescence, and can attract mating partners as well as potential preys. It can also serve as a protective warning mechanism. The rate of flashing and the light intensity are very important characteristics of the communication. The flashing light can be associated with the objective function to be optimized. As a firefly gets closer to a good solution the more light it will emit. The less bright fireflies will move toward the brighter ones. Attractiveness is proportional to the brightness and it decreases as the fireflies distance increases. If there is no brighter one than a particular firefly it will move randomly. Firefly Algorithm is very similar to Particle Swarm Optimization,

however, the fireflies have adjustable visibility and more versatile on attractiveness variations (Fig.8).

```
Input: Problemsize, PopulationSize, γ
Output: best
1. Generate initial population of fireflies xi(i=1,2,...,PopulationSize)
2. Light intensity Ii at xi is determined by f(xi)
3. Define light absorption coefficient γ
4. while (t<MaxGeneration)
5.   for i=1:n all n fireflies
6.     for j=1:i all n fireflies
7.       if (Ij>Ii) Move firefly i towards j in d-dimension;
8.     end if
9.     Attractiveness varies with distance r via exp[-γr]
10.    Evaluate new solutions and update light intensity
11.  end for j
12. end for i
13. Rank the fireflies and find the current best
14. end while
15. return best
```

Fig. 8 Pseudocode of Firefly Algorithm (FF)

Harmony Search (HS) was published by Geem et al. (2001). It was inspired by the improvisation of Jazz musicians. When they start a musical performance, they adapt their music to the band, creating musical harmony. If a false sound occurs, each individual of the band makes modifications to improve their performance. The musicians seek harmony through small variations and improvisation, and the harmony is taken as a complete candidate solution. The audiences’ aesthetic appreciation of the harmony, represent the cost function. The algorithm is similar to Cultural Algorithm, because the candidate solution’s components are stochastically created either directly from the memory of high-quality solutions, adjusted from the memory of high-quality solutions, or assigned randomly (Fig.9).

```

Input: Pitchnum, Pitchbounds, Memorysize, Consolidationrate, PitchAdjustrate, Improvisationmax
Output: Harmonybest
1. Harmonies ← InitializeHarmonyMemory(Pitchnum, 1 Pitchbounds, Memorysize);
2. EvaluateHarmonies(Harmonies);
3. for i to Improvisationmax do
4.   Harmony ← ∅;
5.   foreach Pitchi ∈ Pitchnum do
6.     if Rand() ≤ Consolidationrate then
7.       RandomHarmonyipitch ←
8.       SelectRandomHarmonyPitch(Harmonies, Pitchi);
9.       if Rand() ≤ PitchAdjustrate then
10.        Harmonyipitch ← AdjustPitch(RandomHarmonyipitch);
11.       else
12.        Harmonyipitch ← RandomHarmonyipitch ;
13.       end
14.     else
15.       Harmonyipitch ← RandomPitch(Pitchbounds);
16.     end
17.   end
18.   EvaluateHarmonies(Harmony);
19.   if Cost(Harmony) ≤ Cost(Worst(Harmonies)) then
20.     Worst(Harmonies) ← Harmony;
21.   end
22. end
23. return Harmonybest;

```

Fig. 9 Pseudocode of Harmony Search (HS) by Brownlee (2011)

Memetic Algorithm (MA) was developed by Moscato (1989). The algorithm simulates the creation and inheritance of cultural information among individuals. Meme is the basic unit of cultural information (an idea, discovery, etc.), which name derives from the biological term gene. Universal Darwinism is the generalization of genes beyond biological-based systems to any system, where discrete units of information can be distributed, and subjected to evolutionary changes. The objective of the algorithm is to make a population based global search, while the individuals explore the promising areas with a local search. Balance is crucial between global and local search mechanisms to ensure the algorithm doesn't get stuck at a local optimum, while it spares computational resources. Meme is information about the search, shared between individuals through generations, which influence the evolutionary processes. Memetic Algorithm is a duality of genetic and cultural evolutionary methods (Fig.10).

```
Input: ProblemSize, Popsiz, MemePopsiz
Output: Sbest
1. Population ← InitializePopulation(1 ProblemSize, Popsiz);
2. while ¬StopCondition() do
3.   foreach Si ∈ Population do
4.     Sicost ← Cost(Si);
5.   end
6.   Sbest ← GetBestSolution(Population);
7.   Population ← StochasticGlobalSearch(Population);
8.   MemeticPopulation ← SelectMemeticPopulation(Population, MemePopsiz);
9.   foreach Si ∈ MemeticPopulation do
10.    Si ← LocalSearch(Si);
11.  end
12. end
13. return Sbest;
```

Fig. 10 Pseudocode of Memetic Algorithm (MA) by Brownlee (2011)

Nelder-Mead (NM) algorithm was named after its creators, as Nelder and Mead (1965) proposed this heuristic algorithm. In the literature (McCaffrey 2014) the method is often referred as Amoeba Method. Basically Nelder-Mead is a simplex search algorithm, commonly used for nonlinear optimization problems. The algorithm creates random candidate solutions, and each solution has an associated fitness function value. The candidates are ordered by their fitness, and at each generation, the algorithm attempts to replace the worst solution with a better one. The better solution is chosen from among three candidates, the reflected point, the expanded point and the contracted point. All of these points lie along a line from the worst point through the centroid. The centroid is in the middle of all points except the worst point. If neither the points are better than the current worst solution, the amoeba moves all points, except for the best point, halfway towards the best point (Fig.11).

```

Input: ProblemSize, AmoebaSize
Output: Sbest
1. generate amoebaSize random solutions
2. while not done loop
3.   compute centroid, reflected
4.   if reflected is better than best solution then
5.     compute expanded
6.     replace worst solution with better of reflected, expanded
7.   else if reflected is worse than all but worst then
8.     if reflected is better than worst solution then
9.       replace worst solution with reflected
10.    end if
11.    compute contracted
12.    if contracted is worse than worst
13.      shrink the amoeba
14.    else
15.      replace worst solution with contracted
16.    end if
17.  else
18.    replace worst solution with reflected
19.  end if
20. end loop
21. return best solution found

```

Fig. 11 Pseudocode of Nelder-Mead (NM)

Particle Swarm Optimization (PSO) swarm intelligence heuristic algorithm was developed by Eberhart and Kennedy (1995). Particle Swarm's operation was inspired by the foraging movement of bird and fish swarms. The particles of the swarm move around the search space towards historically good areas. A particle's new position is influenced by the best known position of their own, also by the best known position of the whole swarm. The mathematical formulae of the movement determine the new velocity and position of particles, guiding the swarm towards the global optima. The process is repeated in each generation, while stochastic factors also affect the particles' movement (Fig.12).

```

Input: ProblemSize, Populationsize, w, c1, c2
Output: Pg best
1. Population  $\leftarrow \emptyset$ , Pg best  $\leftarrow \emptyset$ ;
2. for i = 1 to Populationsize do
3.   Pvelocity  $\leftarrow$  RandomVelocity();
4.   Pposition  $\leftarrow$  RandomPosition(ProblemSize);
5.   Pcost  $\leftarrow$  Cost(Pposition);
6.   Pp best  $\leftarrow$  Pposition;
7.   if Pcost  $\leq$  Pg best then Pg best  $\leftarrow$  Pp best;
8.   end
9. end
10. while  $\neg$ StopCondition() do
11.   foreach P  $\in$  Population do
12.     Pvelocity  $\leftarrow$  UpdateVelocity(Pvelocity, Pg best, Pp best, w, c1, c2);
13.     Pposition  $\leftarrow$  UpdatePosition(Pposition, Pvelocity);
14.     Pcost  $\leftarrow$  Cost(Pposition);
15.     if Pcost  $\leq$  Pp best then Pp best  $\leftarrow$  Pposition;
16.     if Pcost  $\leq$  Pg best then Pg best  $\leftarrow$  Pp best;
17.   end
18. end
19. end
20. end
21. return Pg best;

```

Fig. 12 Pseudocode of Particle Swarm Optimization (PSO) by Brownlee (2011)

Multi-Swarm Optimization (MSO) is a variant of Particle Swarm Optimization (PSO). Instead of using one swarm, MSO uses a user-defined number of swarms to locate the global optima (Zhao et al. 2008). The algorithm is especially good for multi-modal optimization problems, where numerous local optima exist. Multi-Swarm Optimization is a new approach to improve the balance between global and local search.

Random Search (RS) algorithm, as the name suggests, is a simple random search algorithm. It takes any position in the search space with equal probability (Brooks 1958). The new solutions are always independent from the previous ones. Random search provides a candidate solution construction and evaluation routine (Fig.13).

```

Input: NumIterations, ProblemSize, Populationsize, SearchSpace
Output: Best
1. Best  $\leftarrow \emptyset$ ;
2. foreach iteri  $\in$  NumIterations do
3.   candidatei  $\leftarrow$  RandomSolution(ProblemSize, SearchSpace);
4.   if Cost(candidatei) < Cost(Best) then Best  $\leftarrow$  candidatei;
5.   end
6. end
7. return Best;

```

Fig. 13 Pseudocode of Random Search (RS) by Brownlee (2011)

Simulated Annealing (SA) method was described by Kirkpatrick (1983). The operation of the algorithm is based on a physical phenomenon. In metallurgy, certain materials gain beneficial properties when heated and then cooled under controlled conditions. The materials' crystal structure is transformed during the process, because the particles take more favourable positions. The heuristic algorithm emulates this process to search for better solutions to a given problem. Each solution of the algorithm represents a particle, the acceptance of new position is controlled by the Metropolis-Hastings Monte Carlo algorithm. As the system is cooled, the acceptance criteria of samples is narrowed to focus on improving movements. Theoretically, if the cooling process is long enough, the system always converge to the global optima. The continuous version of the algorithm was developed by Corana et al. (1987) (Fig.14).

```

Input: ProblemSize, iterationsmax, tempmax
Output: Sbest
1. Scurrent 1 ← CreateInitialSolution(ProblemSize);
2. Sbest ← Scurrent;
3. for i = 1 to iterationsmax do
4.   Si ← CreateNeighborSolution(Scurrent);
5.   tempcurr ← CalculateTemperature(i, tempmax);
6.   if Cost(Si) ≤ Cost(Scurrent) then Scurrent ← Si;
7.   if Cost(Si) ≤ Cost(Sbest) then Sbest ← Si;
8.   end
9.   else if Exp( (Cost(Scurrent) - Cost(Si)) / tempcurr ) > Rand() then Scurrent ← Si;
10.  end
11. end
12. return Sbest;

```

Fig. 14 Pseudocode of Simulated Annealing (SA) by Brownlee (2011)

Table 3 The parameter set and values at different algorithms

Algorithm	Name of the parameter	Value of the parameter
Bacterial Foraging Algorithm (BFOA)	Cellsnum	50
	Ned	4
	Nre	5
	Nc	25
	Ns	10
	Stepsize	0.05
	Ped	0.25
Bat Algorithm (BATA)	PopulationSize	40
	Loudness	0.5

	PulseRate	0.5
	Frequency	2
Bees Algorithm (BA)	Beesnum	50
	Sitesnum	3
	EliteSitesnum	1
	PatchSizeinit	3
	OtherBeesnum	2
	EliteBeesnum	7
Cross-Entropy Method (CEM)	Samplesnum	50
	UpdateSamplesnum	5
	Learnrate	0.7
Cuckoo Search Algorithm (CS)	NestSize	40
	DiscoveryRate	0.25
Cultural Algorithm (CA)	Populationnum	50
Differential Evolution (DE)	Populationsize	50
	Weightingfactor	0.8
	Crossoverrate	0.9
Firefly Algorithm (FF)	PopulationSize	40
	Gamma	1
Harmony Search (HS)	Memorysize	50
	Consolidationrate	0.95
	PitchAdjustrate	0.7
Memetic Algorithm (MA)	Populationsize	50
	MemePopsize	16
Multi-Swarm Optimization (MSO)	NumberOfSwarms	3
	c3	0.3645
	death	0.005
	immigrate	0.005
Nelder-Mead Algorithm (NM)	Amoebasize	20
Random Search (RS)	Populationsize	50
Particle Swarm Optimization (PSO)	Populationsize	50
	W	0.729
	C1	1.49445
	C2	1.49445
Simulated Annealing (SA)	Tempmax	Searchspace (max)-
		Searchspace (min)

5. NUMERICAL EXPERIMENTS

Benchmarking heuristic optimization algorithms is a quite difficult and complex process. Due to the algorithms' stochastic factors, we had to use statistical methods to present satisfactory results. The heuristic algorithms did 100 Monte Carlo searches per test function. We specified the iteration limit for each search to be 500.

The input parameters for the algorithms are determined based on the proposals found in the literature indicated.

5.1 STATISTICAL RESULTS

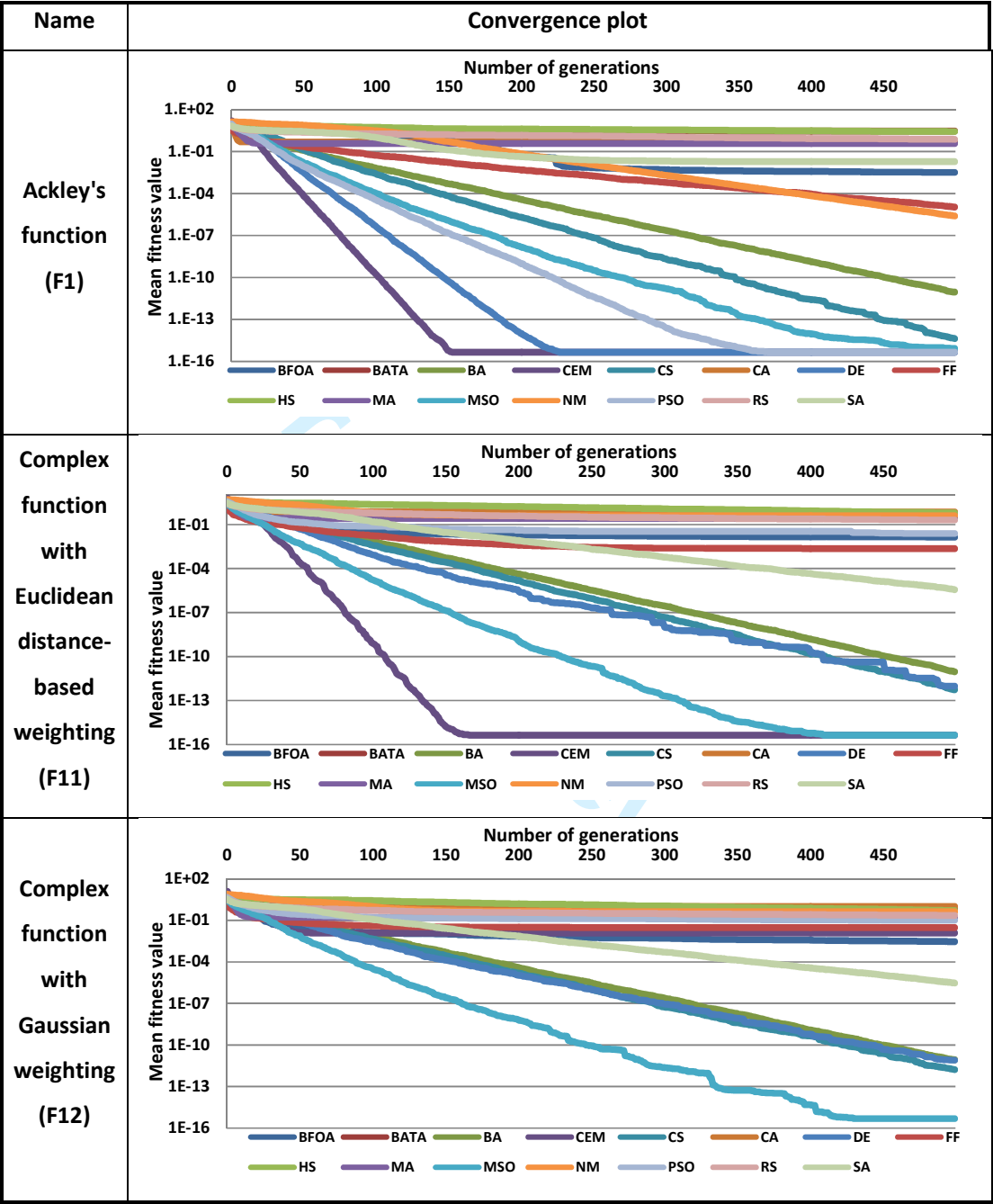
The statistical table summarizes the performance of algorithms broken down by test functions. Table 4 gives an overview regarding algorithms' efficiency and reliability. The values are normalized, so that the optima in each row are 0, and the maximum is 100. These are not the absolute minima found by each algorithm, but the average minima for 100 Monte Carlo simulations.

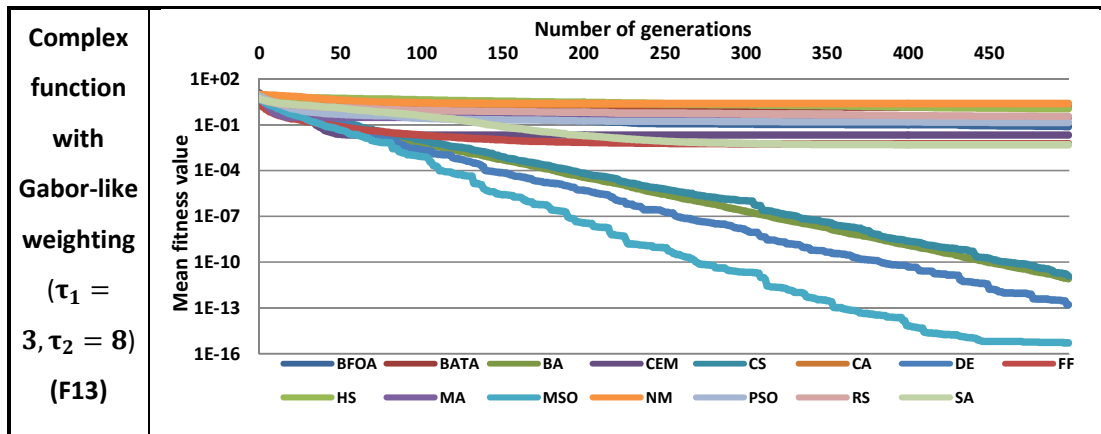
Table 4 Mean normalized optimization results for thirteen benchmark functions.

	BFOA	BATA	BA	CEM	CS	CA	DE	FF	HS	MA	MSO	NM	PSO	RS	SA
F1	0,11	100,00	0	0	0	16,45	0	0	91,79	12,49	0	0	0	25,28	0,63
F2	0,01	0	0	0	0	100,00	0	0	48,51	1,57	0	0	0	5,36	0
F3	2,82	100,00	0	14,67	0	47,96	0,17	0,02	76,81	13,93	0	63,26	0	27,15	50,81
F4	0	67,91	3,05	0	0	11,78	75,26	18,81	62,98	60,31	0	100,00	0	1,68	3,33
F5	100,00	31,93	0	0,43	0	4,38	0,01	0,03	40,28	1,90	0	3,55	0,11	8,03	28,69
F6	0	0	0	0	0	13,84	2,50	0	100,00	35,87	0	0	0	0,86	0,03
F7	0,24	49,39	0	2,28	0	82,37	0	0,17	43,75	22,23	0	100,00	0	19,63	0
F8	0,03	0	4,65	0	0,24	100,00	0,11	0	86,60	32,11	0	0	0	2,63	0
F9	100,00	65,45	0	0	0	21,40	0,01	0	48,21	7,25	0	76,73	0	3,92	23,27
F10	0,02	100,00	0	0	0	16,45	0	0	2,74	1,13	0	0	0	1,53	0
F11	1,86	76,03	0	0	0	100,00	0	0,31	83,33	36,74	0	54,40	3,40	27,87	0
F12	3,07	56,07	0	0,84	0	83,39	0	0,22	46,27	12,23	0	100,00	5,79	14,01	0,20
F13	0,29	59,48	0	1,18	0	100,00	0	2,99	49,47	16,54	0	42,93	9,99	22,29	0
Σ	2	3	11	8	12	0	7	6	0	0	13	5	9	0	6

We used convergence plots in Table 5 to measure the convergence rate of search algorithms. Convergence rate shows how quickly the heuristic algorithms can find the optimum. The data points are the best fitness found in each iteration, averaged for 100 Monte Carlo simulations.

Table 5 Convergence plots for Ackley's (F1) and Complex (F11, F12, F13) functions





5.2 BENCHMARK CONCLUSION

Fifteen search algorithms have been used for the thirteen benchmark problems. The test functions had quite diverse characteristics, like unimodal and multimodal problems with varying number and distribution of local extremes. Three complex benchmark problems were constructed by us with the software we created. After we examined the significant amount of statistical data, we get a picture of the algorithms' overall performance. If we take a look at Table 4, we can see the superiority of swarm intelligence methods. Multi-Swarm Optimization (MSO) always found the global optima, while Cuckoo Search (CS) and Bees Algorithm (BA) almost always found the global optima, even in case of very difficult and noisy functions. Additionally, Cross-Entropy Method and Simulated Annealing (SA) worked well, however sometimes they trapped at local optima. To improve the performance of slowly converging methods (HS, SA), we should increase the number of iterations. The rate of convergence could be observed very well on the convergence plots of Table 5. The best algorithms' rate of convergence is relatively fast, and this characteristic proved to be crucial for success. Furthermore, the convergence plots showed the weighting functions really affect the difficulty of complex functions. The averaged fitness function values revealed the Complex function with Euclidean distance-based weighting (F11) was harder to solve than the basis function (F1). However the use of Gaussian weighting function resulted in smoother edges, which decreased the number and distribution of local extremes,

making it easier to find the global optima for the search algorithms. Probably the hardest test function was Complex function with Gabor-like weighting, because the weighting function generated significant noise. In case of this test problem, the best algorithm was Multi-Swarm Optimization, however its solution was worse than the solution of Euclidean distance-based weighting. As final conclusion, we can say that the complex functions with different weighting proved to be great benchmark problems.

6. DETERMINING THE OPTIMAL DIMENSIONS OF AN OVERHEAD CRANE’S MAIN GIRDER

The overhead crane is one of the most frequently used lifting machinery in the modern industry. The primary objective of the crane is to handle and transfer heavy payloads from one position to another. Besides lifting heavy load, also it’s capable of short-term horizontal transportation, serving an area of floor space within its own travelling restrictions. Overhead cranes are typically used for manufacturing and logistic purposes, where efficiency and downtime are crucial factors. It can transport objects between workshops of factories, storehouses, and in the freight yard of railways and ports. The parallel runways of the crane are usually built on steel or concrete columns, or on the reinforced walls of the facility. Since the structure of the bridge is mounted high above ground level, it occupies relatively small amount of useful space. A traveling bridge spans the gap between the runways, which is able to roll on its driven wheels. The trolley travels along the rails of the bridge, with the lifting component of the crane, called hoist. The key element of the traveling bridge is the girder, as the main load-bearing part of the structure. Regarding the number of girders, single and double constructions are the most prevalent. Most of the time, the girders are fabricated from structural steel. Determining the main girder's dimensions is a crucial part of the crane’s planning process. In order to keep the production and operating costs as low as possible, we have to minimize the dead weight of the girder. However, the structure must remain sturdy enough, so that the bridge crane can operate reliably through its lifetime (Sun

et al. 2011). The exaggeration of safe coefficients may lead to waste of materials and energy consumption, and so forth (Zuberi et al. 2008). In this paper, we examined a double welded box type girder, made of structural steel plate. The optimization of the overhead crane's main girders is a non-linear, constrained optimization problem.

6.1 THE MATHEMATICAL MODEL OF THE OPTIMUM DESIGN

To solve the overhead crane's main girder design problem, the mathematical model of the optimization should be described. We have to define the decision variables, the constraints and the objective functions.

6.1.1 DECISION VARIABLES

The dimensions of the overhead crane's double welded box type girders can be described with four variables, respectively h , $\frac{t_w}{2}$, b , t_f (Farkas and Jármai 1997).

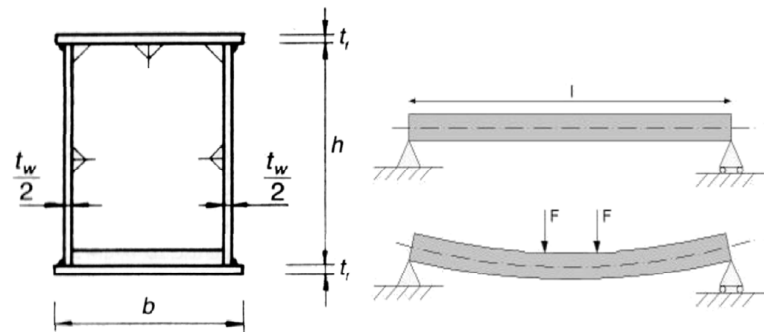


Fig. 15 The decision variables of the design problem

6.1.2 CONSTRAINTS

We should determine minimum and maximum limits (Constraints F1, F2, F3, F4) on the four decision variables:

$$300 \text{ mm} \leq h \leq 2000 \text{ mm}$$

$$4 \text{ mm} \leq \frac{t_w}{2} \leq 50 \text{ mm}$$

$$300 \text{ mm} \leq b \leq 1200 \text{ mm}$$

$$4 \text{ mm} \leq t_f \leq 50 \text{ mm}$$

The **fatigue** (Constraint F5) constraint is defined by Eurocode 3 (2009)

$$\Delta\sigma = \frac{M_q}{4W_x} \leq \frac{\Delta\sigma_n}{\gamma_{Mf}} \quad (10)$$

$$M_q = \psi_d \frac{F}{2L} \left(L - \frac{k}{2}\right)^2 \quad (11)$$

$$W_x = \frac{2I_x}{(h + t_f)} \quad (12)$$

$$I_x = \frac{h^3 \frac{t_w}{2}}{12} + 2b \frac{t_f^3}{12} + 2bt_f \left(\frac{h}{2} + \frac{t_f}{2}\right)^2 \quad (13)$$

where M_q is the varying bending moment, ψ_d is a dynamic factor, F is the varying vertical load, L is the span length of the main girder, k is the length of the trolley, W_x is the elastic section modulus, I_x is the moment of inertia, $\Delta\sigma_n$ is the fatigue stress range corresponding to the given cycles of N ($N = 3 \times 10^6$), γ_{Mf} is a safety factor (according to Eurocode 3 (2009), $\gamma_{Mf} = 1.35$).

The **static stress** (Constraint F6) is expressed by:

$$\sigma_{max} = \frac{M_s}{W_x} \leq \frac{f_y}{\gamma_{M1}} \quad (14)$$

$$M_s = \frac{\gamma_G L^2}{8} (A \rho g + g_j + g_s) + \gamma_G \frac{G_1 L}{4} \quad (15)$$

where M_s is the static stress, f_y is the yield stress, γ_{M1} , γ_G are safety factors, A is the cross-sectional area ($A = ht_w + 2bt_f$), ρ is the volumetric mass density, g is the gravitational acceleration, g_j is the uniform vertical load on the sidewalk, g_s is the uniform vertical load of the rail, G_1 is the weight of the actuating machinery (chassis, drive, engine).

The flange buckling (Constraint F7):

$$\frac{(b - 40)}{t_f} \leq \frac{1}{\delta} \quad (16)$$

$$\delta = \frac{1}{42\varepsilon} \quad (17)$$

$$\varepsilon = \sqrt{\frac{235}{f_y}} \quad (18)$$

The web buckling (Constraint F8):

$$\frac{2h}{t_w} \leq \frac{1}{\beta} \quad (19)$$

$$\beta = \frac{1}{124\varepsilon} \quad (20)$$

$$\varepsilon = \sqrt{\frac{235}{f_y}} \quad (21)$$

The deflection (Constraint F9) is defined by:

$$\frac{F(L-k)}{48E_sI_x}(3L^2 - (L-k)^2) \leq \frac{L}{600} \quad (22)$$

Where E_s is the elastic modulus of the steel.

6.1.3 OBJECTIVE FUNCTIONS

The purpose of the optimization is to define the minimum mass of the main girder m , therefore this is a single-objective optimization problem. The mass depends on the size of the cross-sectional area, which is the correlation between the objective function and the design variables.

$$A = ht_w + 2bt_f \quad (23)$$

$$m = AL\rho \quad (24)$$

6.2 SOLVING THE OPTIMUM DESIGN PROBLEM

We used the heuristic algorithm collection to solve this structural optimization problem, the input parameters were the same as in Table 3. The heuristic algorithms did 100 Monte Carlo searches, we specified the iteration limit for each search to be 1000. In order to avoid the violation of constraints, we created a penalty function. The following data are given: $\psi_d=2$, $F=240000$ [N], $L=20000$ [mm], $k=1900$ [mm],

$\Delta\sigma_n=69.8$ [MPa], $\gamma_{Mf}=1.35$, for Fe 360 steel the yield stress is $f_y = 235$ [$\frac{N}{mm^2}$],
 $\gamma_{M1}=1.1$, $\gamma_G=1.35$, $\rho=7.85$ [kg/dm^3], $g=9.81$ [m/s^2], $g_j=1$ [$\frac{N}{mm^2}$], $g_s=0.2$ [$\frac{N}{mm^2}$],
 $G_1=30$ kN, $E_s=2.1 \times 10^5$ [$\frac{N}{mm^2}$].

The optimum dimensions of welded box type girder and the corresponding constraint values are summarized in Table 6 and Table 7.

Table 6 Optimum dimensions of welded box type main girder

	h	$\frac{t_w}{2}$	b	t_f	Cross-Sectional Area [mm^2]	Mass [kg]	Run time [ms]
BFOA	1209,889	19,53007	702,8998	17,30789	47960,64	7529,82	368
BATA	1266,957	20,43493	503,0842	22,05898	48085,24	7549,382	79
BA	1240,19	20,21222	625,6542	18,49131	48205,32	7568,235	98
CEM	1210,866	19,53009	586,1365	20,74772	47970,32	7531,34	556
CS	1186,37	19,15503	675,2558	18,73322	48024,38	7539,828	252
CA	1122,725	19,51167	540,9183	22,64809	48040,19	7542,311	146
DE	1220,704	19,69891	534,5374	22,41704	48012,03	7537,889	80
FF	1216,134	19,61507	708,8209	16,9796	47925,53	7524,309	1859
HS	1161,865	18,98102	538,3703	24,45729	48387,55	7596,845	8
MA	1305,94	21,20183	320,3388	32,5539	48544,88	7621,546	8309
MSO	1212,164	19,55104	699,1941	17,32737	47929,45	7524,924	274
NM	1216,954	19,62829	684,4668	17,56559	47932,84	7525,456	7
PSO	1212,432	19,55535	687,6858	17,61221	47932,87	7525,461	90
RS	1245,286	20,12348	666,9302	17,37803	48239,35	7573,578	165
SA	1217,185	19,63202	704,3912	17,05795	47926,74	7524,498	180

The best solution was found by Firefly Algorithm (FF), however the differences between the results were relatively small. The weight difference between the best solution (Firefly Algorithm with 7524.309 kg) and the worst solution (Memetic Algorithm with 7621,546 kg) was only 97.237 kg, which is less than 2% of the main girder's mass. The small difference between the results indicates that all the algorithms worked properly. The values of the constraints in Table 7 further verify

the heuristic algorithms’ correct operation, because the maximum limits were respectively $\text{fatigue} \leq 66.48$, $\text{static stress} \leq 213.64$, $\text{flange buckling} \leq 42$, $\text{web buckling} \leq 124$, $\text{deflection} \leq 66.67$.

Table 7 Corresponding constraint values to optimum dimensions

	Fatigue	Static stress	Flange buckling	Web buckling	Deflection
BFOA	56,28	213,64	35,31	124,00	31,56
BATA	56,28	213,63	41,78	123,97	31,09
BA	56,23	213,50	27,95	123,11	30,30
CEM	56,24	213,53	21,88	123,94	30,49
CS	56,20	213,34	37,22	123,96	31,45
CA	56,26	213,63	12,78	124,00	29,26
DE	55,56	211,04	17,75	123,74	30,35
FF	<u>56,28</u>	<u>213,64</u>	<u>40,38</u>	<u>124,00</u>	<u>31,23</u>
HS	56,28	213,64	37,75	124,00	31,08
MA	55,22	213,64	32,98	123,69	30,60
MSO	56,28	213,36	23,76	124,00	29,77
NM	56,28	213,64	40,87	124,00	31,41
PSO	56,28	213,27	38,97	124,00	31,55
RS	55,12	213,11	29,92	123,51	29,83
SA	56,24	213,64	40,25	124,00	31,86

To further investigate the results, we determined the optimal dimensions of the overhead crane’s main girder with Brute Force method. Brute Force evaluated all feasible solutions, however to get the results in a rational time span, we had to discretize the continuous variables. Still, the evaluation of 3.237.480.000 possible solutions took 70 minutes (4149129.594 msec). The certainly best solution is $h = 1240\text{ mm}$, $t_w/2 = 20\text{ mm}$, $b = 681\text{ mm}$, $t_f = 17\text{ mm}$, and the minimum weight of the main girder is 7528.778 kg. Comparing the heuristic algorithms’ results with the Brute Force method’s solution revealed the heuristic algorithms provided very good results. Firefly algorithm has found a better solution in less than two seconds than the Brute Force method in 70 minutes. The difference of 4 kg was caused by

the rounding of continuous variables. It also worth mentioning that if we would like to manufacture the main beam according to the Firefly Algorithm's results, the continuous values should also be discretized.

7. SUMMARY

The field of numerical optimization is an always evolving scientific field. Several new evolutionary optimization techniques appeared lately. We created a software solution, which provides practically endless possibilities to create arbitrarily difficult complex test functions. We benchmarked fifteen optimization algorithms with thirteen test functions and a real-word design problem, where determined the optimal dimensions of an overhead crane's main girder. We have made comparison with Brute Force method. Both the test functions and the crane girder design show, that the evolutionary optimization techniques are very efficient tools of the design. In the future, we would like to create more difficult, self-made test problems, and benchmark further heuristic techniques. Based on the benchmark result, we will try to create novel, more efficient hybrid heuristic algorithms, which could be utilized in real-life structural and logistics optimization problems.

8. ACKNOWLEDGEMENTS

The research was supported by the TÁMOP 4.2.4.A/2-11-1-2012-0001 priority project entitled 'National Excellence Program - Development and operation of domestic personnel support system for students and researchers, implemented within the framework of a convergence program, supported by the European Union, co-financed by the European Social Fund. The research was supported also by the Hungarian Scientific Research Fund OTKA T 109860 project and was partially carried out in the framework of the Center of Excellence of Innovative Engineering Design and Technologies at the University of Miskolc.

9. REFERENCES

Barcsák C, Jármai K (2013) Benchmark for testing evolutionary algorithms, WCSMO10, Orlando, USA, Abstract book 5124, E-Proceedings 5124.pdf, 10 p. ISBN-13:978-0-615-85240-9

Brooks SH (1958) A discussion of random methods for seeking maxima. Operations Research, 6(2), pp. 244–251.

Brownlee J (2011) Clever Algorithms: Nature-Inspired Programming Recipes, Lulu, pp. 27-336, ISBN: 978-1-4467-8506-5

Corana A, Marchesi M, Martini C, Ridella S (1987) Minimizing Multimodal Functions of Continuous Variables with the ‘Simulated Annealing’ Algorithm, Istituto per i Circuiti Elettronici-C.N.R., ACM Transactions on Mathematical Software, Vol. 13, No. 3, September, pp. 262-280.

Eurocode 3 (2009) Design of steel structures. Part 1-1: General structural rules. Brussels, CEN

Farkas J, Jármai K (1997) Analysis and Optimum Design of Metal Structures, Taylor & Francis, pp. 236-239, ISBN: 978-9-0541-0669-2

Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: Harmony search. Simulation, 76(2), pp. 60–68.

Kennedy J, Eberhart RC (1995) Particle swarm optimization, In Proceedings IEEE Int’l Conf. on Neural Networks, Vol. IV, pp. 1942–1948.

Kirkpatrick S (1983) Optimization by simulated annealing: Quantitative studies. Journal of Statistical Physics, 1983. pp. 975–986.

Liang J, Suganthan N, Deb K (2005) Novel composition test functions for numerical global optimization, Swarm Intelligence Symposium, Proceedings, pp. 68-75.

Liu Y, Passino KM (2002) Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviours, Journal of Optimization Theory and Applications, pp. 603–628.

- 1
2
3 Marcsák G, Jármai K (2014) C# source code of heuristic algorithms,
4 [https://drive.google.com/folderview?id=0BxE6yHbGFZBAOHBpN2VIV08yS0k&u](https://drive.google.com/folderview?id=0BxE6yHbGFZBAOHBpN2VIV08yS0k&usp=sharing)
5 [sp=sharing](https://drive.google.com/folderview?id=0BxE6yHbGFZBAOHBpN2VIV08yS0k&usp=sharing) (access July 29, 2014)
6
7
8
9 McCaffrey J (2013) Amoeba Method Optimization using C#, MSDN Magazine 2013 June.
10 <http://msdn.microsoft.com/en-us/magazine/dn201752.aspx>
11
12
13 McCaffrey J (2013) Multi-Swarm Optimization, MSDN Magazine 2013 September.
14 <http://msdn.microsoft.com/en-us/magazine/dn385711.aspx>
15
16
17 Mologa M, Smutnicki C (2014) Test functions for optimization needs, pp. 1-10,
18 [http://www.bioinformaticslaboratory.nl/twikidata/pub/Education/NBICResearchScho](http://www.bioinformaticslaboratory.nl/twikidata/pub/Education/NBICResearchSchool/Optimization/VanKampen/BackgroundInformation/TestFunctions-Optimization.pdf)
19 [ol/Optimization/VanKampen/BackgroundInformation/TestFunctions-](http://www.bioinformaticslaboratory.nl/twikidata/pub/Education/NBICResearchSchool/Optimization/VanKampen/BackgroundInformation/TestFunctions-Optimization.pdf)
20 [Optimization.pdf](http://www.bioinformaticslaboratory.nl/twikidata/pub/Education/NBICResearchSchool/Optimization/VanKampen/BackgroundInformation/TestFunctions-Optimization.pdf) (access July 29, 2014)
21
22
23
24 Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts:
25 Towards memetic algorithms. Technical report, California Institute of Technology
26
27
28 Nelder JA, Mead R (1965) "A simplex method for function minimization". Computer
29 Journal 7, pp. 308–313.
30
31
32
33 Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S, Zaidi M (2005) The bees algorithm.
34 Technical report, Manufacturing Engineering Centre, Cardiff University
35
36
37 Reynolds RG (1994) An introduction to cultural algorithms. In Proceedings of the 3rd
38 Annual Conference on Evolutionary Programming, World Scientific Publishing, pp.
39 131–139.
40
41
42
43 Rubinstein RY (1997) Optimization of computer simulation models with rare events.
44 European Journal of Operations Research, 99, pp. 89–112.
45
46
47 Storn R, Price K (1995) Differential evolution: A simple and efficient adaptive scheme for
48 global optimization over continuous spaces, Technical Report TR-95-012,
49 International Computer Science Institute, Berkeley, CA
50
51
52
53 Sun C, Tan Y, Zeng J, Pan J, Tao Y (2011) The structure optimization of main beam for
54 bridge crane based on an improved PSO. Journal of computers, 6(8), 1585-1590.
55
56
57
58
59
60

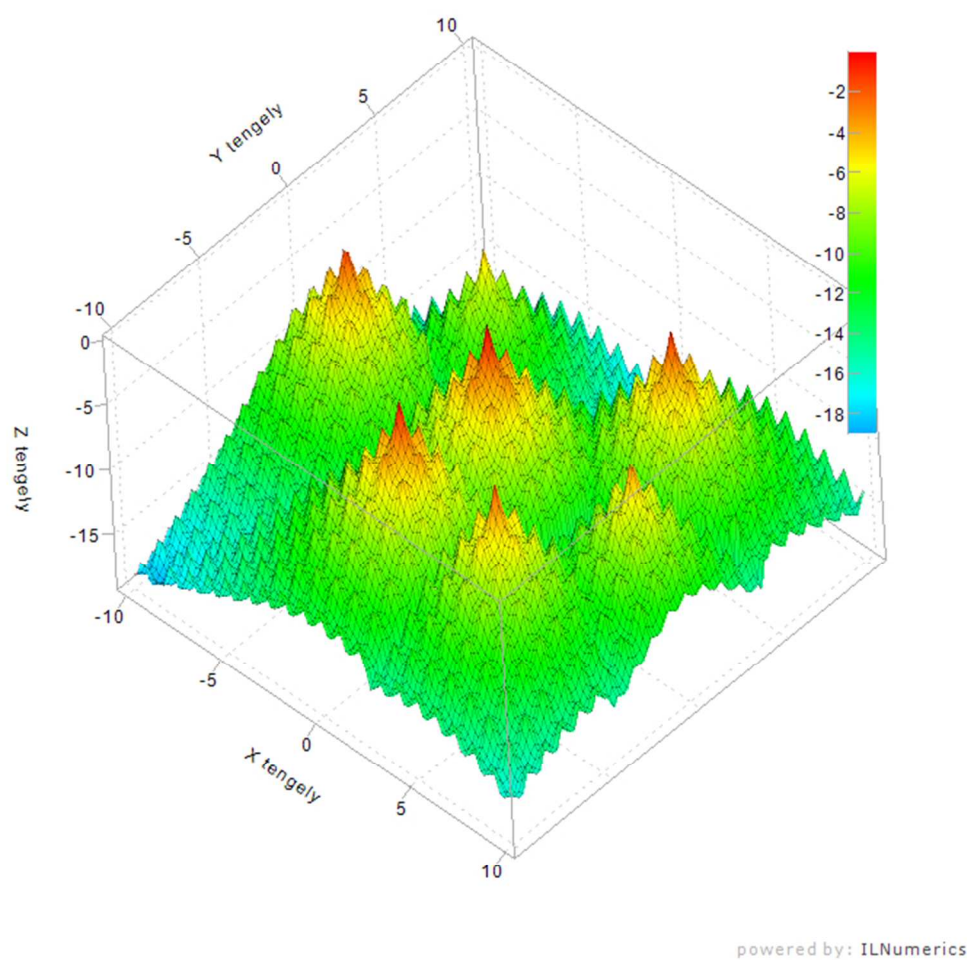
Yang XS (2009) Firefly algorithms for multimodal optimization. Stochastic Algorithms: Foundations and Applications, SAGA 2009. Lecture Notes in Computer Sciences 5792. pp. 169-178.

Yang XS (2010) A New Metaheuristic Bat-Inspired Algorithm, Nature Inspired Cooperative Strategies for Optimization (NISCO 2010), (Eds. J. R. Gonzalez et al.), Studies in Computational Intelligence, Springer Berlin, Springer, pp. 65-74

Yang XS, Deb S (2009) Cuckoo search via L'evy flights, Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, India. IEEE Publications, pp. 210-214.

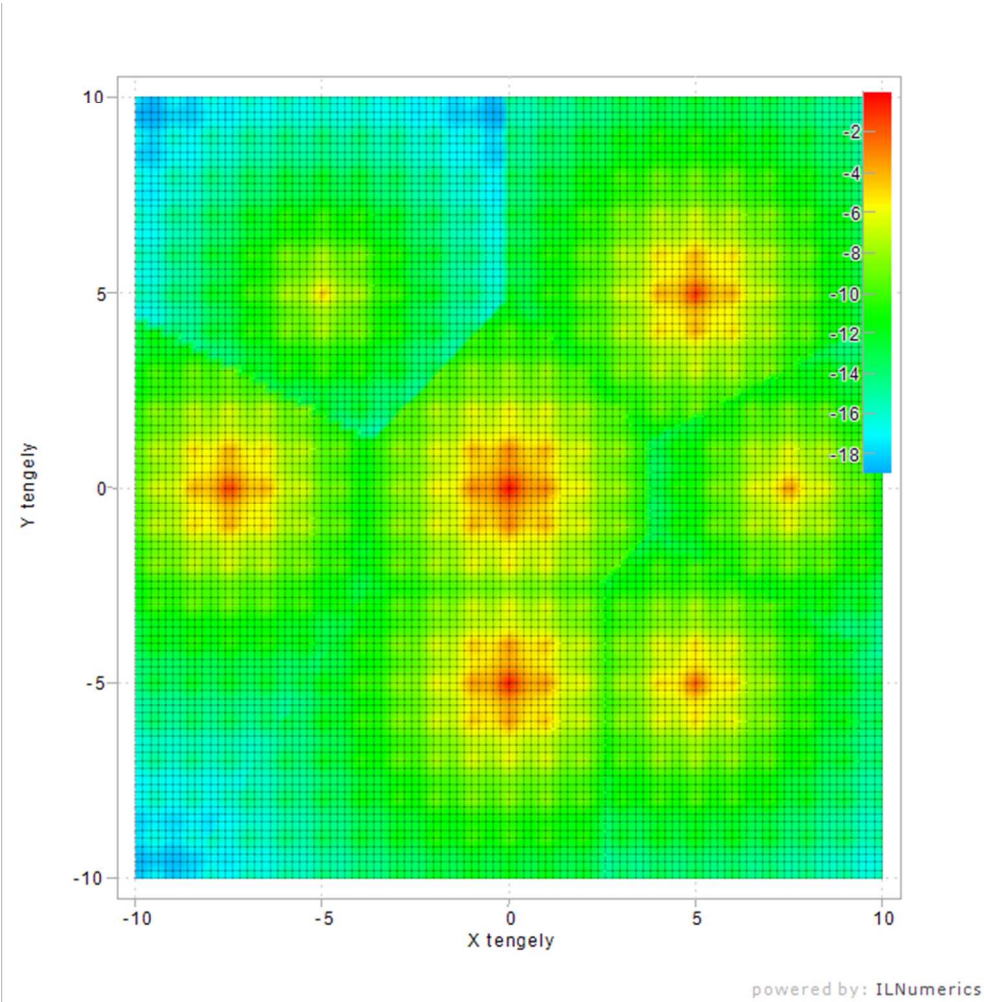
Zhao SZ, Liang JJ, Suganthan PN, Tasgetiren MF (2008) Dynamic Multi-Swarm Particle Swarm Optimizer with Local Search for Large Scale Global Optimization, in Proceedings IEEE Congress on Evolutionary Computation, pp. 3845–3852.

Zuberi RH, Kai L, Zhengxing Z (2008). Design optimization of EOT crane bridge. Eng Opt, 192-201.

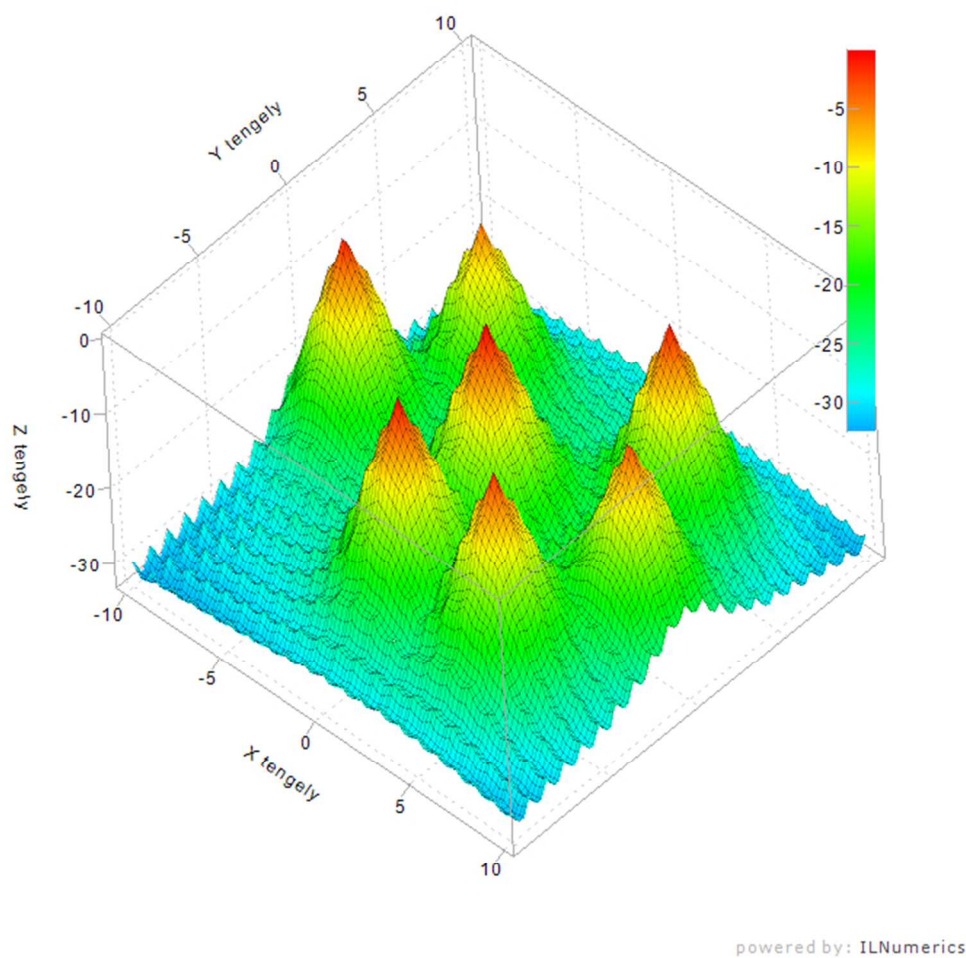


Complex function with Euclidean distance-based weighting (F11)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

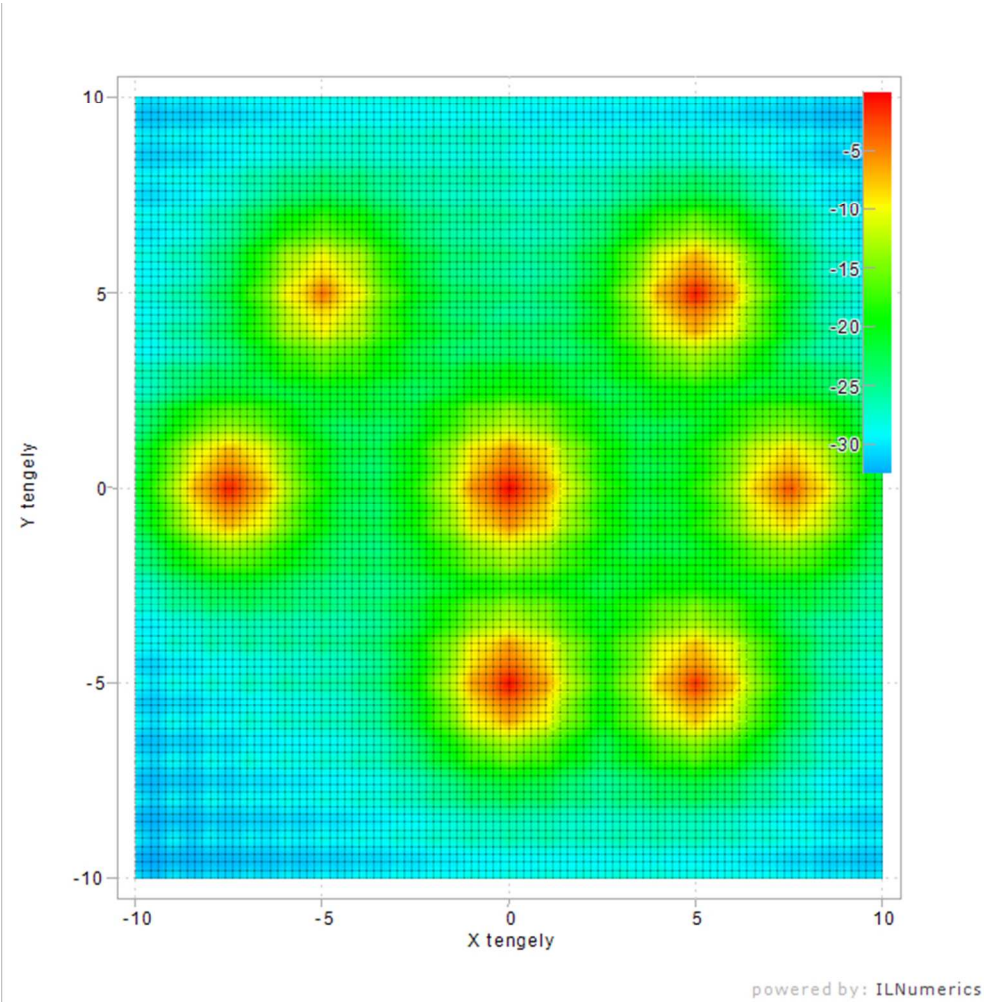


Complex function with Euclidean distance-based weighting (F11)

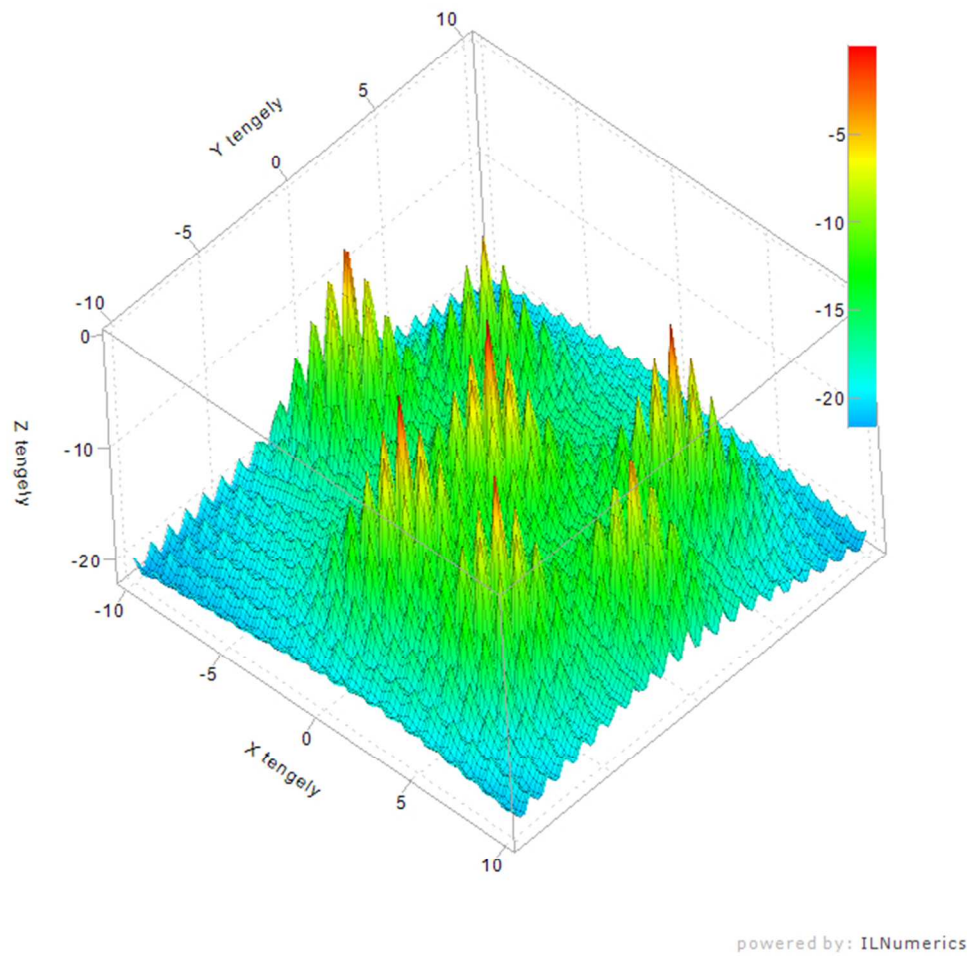


Complex function with Gaussian weighting (F12)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

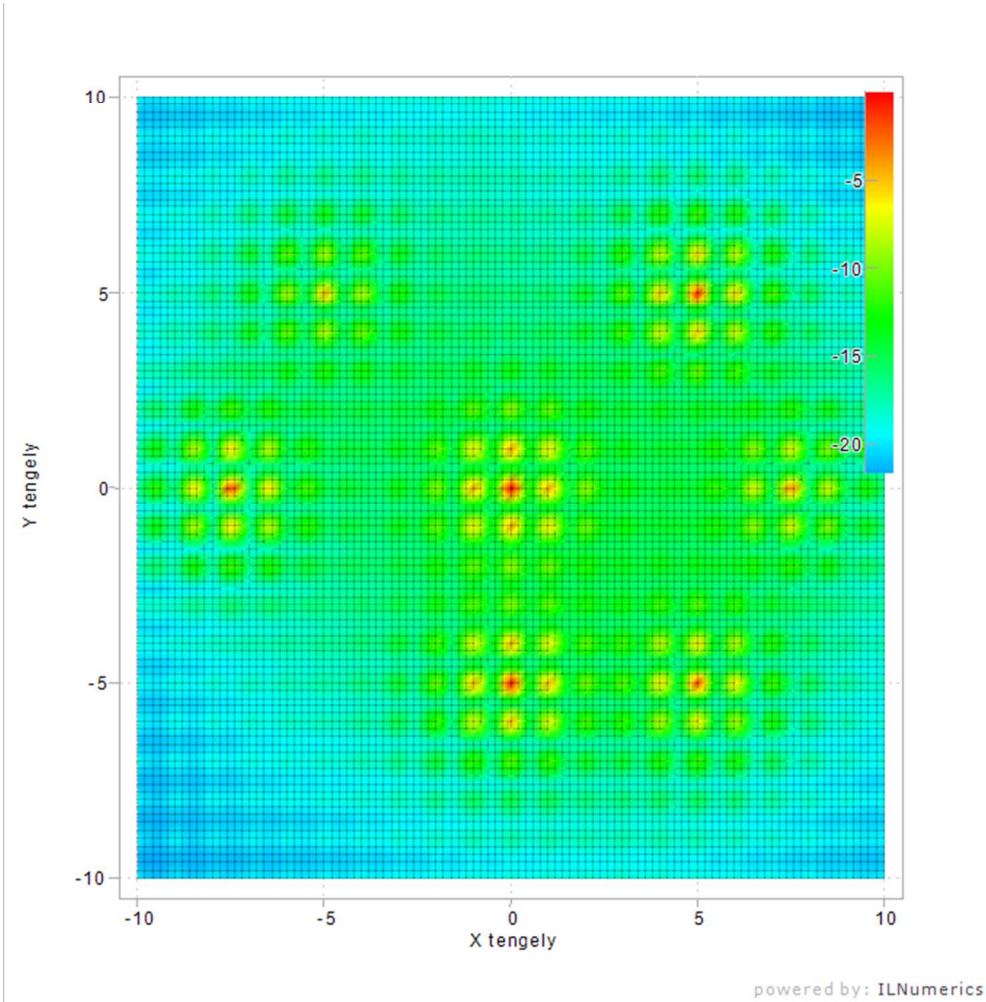


Complex function with Gaussian weighting (F12)

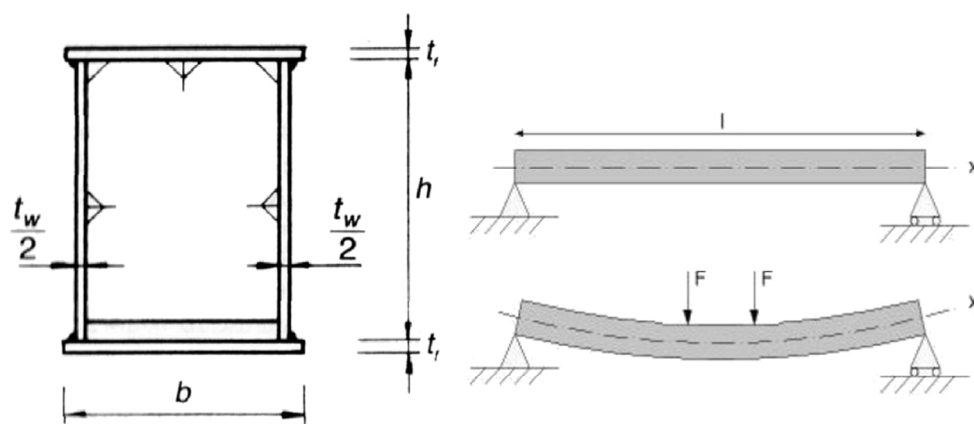


Complex function with Gabor-like weighting ($\tau_1=3, \tau_2=8$) (F13)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60



Complex function with Gabor-like weighting ($\tau_1=3, \tau_2=8$) (F13)



The decision variables of the design problem