Computing and Informatics, Vol. 27, 2008, 377–402

USING STIGMERGY TO SOLVE NUMERICAL OPTIMIZATION PROBLEMS

Peter Korošec, Jurij Šilc

Computer Systems Department
Jožef Stefan Institute
Jamova cesta 39
1000 Ljubljana, Slovenia
e-mail: {peter.korosec, jurij.silc}@ijs.si

Manuscript received 28 March 2006; revised 1 February 2007
Communicated by Marian Vajteršic

Abstract. The current methodology for designing highly efficient technological systems needs to choose the best combination of the parameters that affect the performance. In this paper we propose a promising optimization algorithm, referred to as the Multilevel Ant Stigmergy Algorithm (MASA), which exploits stigmergy in order to optimize multi-parameter functions. We evaluate the performance of the MASA and Differential Evolution – one of the leading stochastic method for numerical optimization – in terms of their applicability as numerical optimization techniques. The comparison is performed using several widely used benchmark functions with added noise.

Keywords: Ant-based algorithm, multilevel approach, numerical optimization, stigmergy

1 INTRODUCTION

Stigmergy is a method of communication in decentralized systems in which the individual parts of the system communicate with one another by modifying their local environment. It was first observed in nature as a class of mechanisms that mediate animal-animal interactions (e.g., ant trails, termite nest building, ant corpse gathering) [27]. The term stigmergy (from the Greek *stigma* = sting, and *ergon* = to work)

was originally defined by the French entomologist Pierre-Paul Grassé in his pioneering studies on the reconstruction of termite nests [9]. He defined it as: *"Stimulation of workers by the performance they have achieved."* Stigmergy provides a new paradigm for developing decentralized, complex applications such as autonomous and collective robotics [12], communication in computer networks [22], multi-agent systems [10], optimization algorithms [4], etc. In this paper we introduce a new stigmergy-based approach to the numerical optimization problem.

Numerical optimization, as described by Nocedal and Wright [19], is important in decision science and in the analysis of physical systems. An important step in optimization is the identification of some objective, i.e., a quantitative measure of the performance of the system. This objective can be any quantity or combination of quantities that can be represented by a single number. The objective depends on certain characteristics of the system called parameters, which are often restricted or constrained in some way. Furthermore, the parameters can have either continuous or discrete values. Our goal is to find values of the parameters that optimize the objective. Depending on the types of parameters, we distinguish between *continuous* optimization [19] and *discrete* optimization [3].

There is no universal optimization algorithm to solve such an optimization problem. Many of the problems arising in real-life applications are NP-hard. Hence, one usually solves large instances with the use of approximate methods that return near-optimal solutions in a relatively short time. Algorithms of this type are called *heuristics*. The upgrade of a heuristic is a metaheuristic: a set of algorithmic concepts that can be used to define a heuristic method applicable to a wider set of different problems. A particularly successful metaheuristic based on stigmergy is observed in colonies of real ants [4]. Ants communicate with one another by laying down pheromone along their trails, so one can say that an ant colony is a stigmergic system. An ant-colony metaheuristic is normally used for solving discrete, combinatorial optimization problems. A direct application of this metaheuristic for solving real-parameter optimization problem is difficult. The first algorithm designed for continuous function optimization was continuous ant-colony optimization (CACO) [1] which comprises two levels: global and local. CACO uses the ant-colony framework to perform local searches, whereas global search is handled by a genetic algorithm. Up to now, there are few other adaptations of ant-colony algorithm to continuous optimization problems: continuous interacting ant colony (CIAC) [6], ant-colony optimization for continuous and mixed-variable (eACO) [24], improved ant-colony algorithm [2], etc. In this paper we will show a new, successful implementation of an ant-colony metaheuristic on a numerical, multi-parameter optimization problem that is often solved by algorithms for continuous optimization.

The rest of the paper is organized as follows. Following a brief explanation of the transformation approach from continuous to discrete form of multi-parameter problem in Section 2, the Ant Stigmergy Algorithm is defined in Section 3. In Section 4, the multilevel approach is explained. The Multilevel Ant Stigmergy Algorithm is described in Section 5, followed by experiments in Section 6, and the conclusions in Section 7.

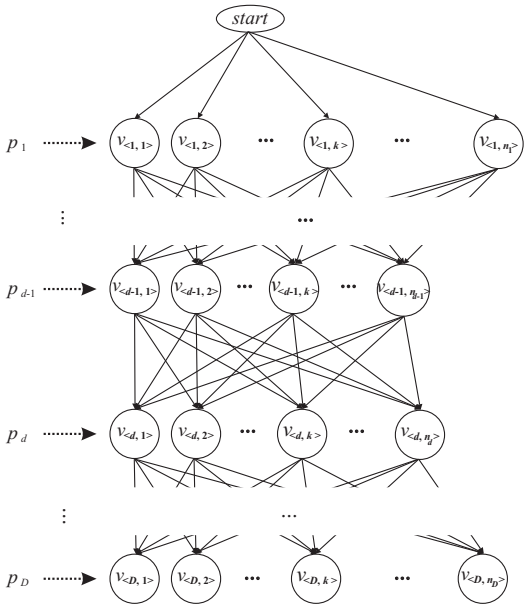


Fig. 1. Search graph representation

2 A MULTI-PARAMETER PROBLEM

Multi-parameter optimization is the process of finding the point in the parameter space $P = \{p_1, p_2, \ldots, p_D\}$ where a cost function $f(P)$ is minimized according to the feasible set Ω of parameters p_i, $i = 1, 2, \ldots, D$, that satisfy the constraints. Very often this cost function contains information about the problem target and the constraints that the solution has to meet (constrained optimization). Optimizing a multi-parameter function is usually a continuous problem.

Because of the nature of ant-based algorithms we first had to put the continuous multi-parameter problem into discrete form. More precisely, if a parameter p_i has a range from L_i to U_i and the discrete step is Δ_i then a discrete parameter p_i has $\lceil \frac{U_i - L_i}{\Delta_i} \rceil + 1$ discrete values.

Now that we have the problem discrete we need to find a way to use our ants to find a solution. Generally, ant-based algorithms solve different problems with the use of a graphical representation. In our case we decided to use a search graph.

2.1 Graph Representation

A search graph is defined as a connected, directed, non-weighted, acyclic graph. It is also rooted and ordered. We translate all the discrete parameter values into a search graph. For this purpose we define a search graph $\mathcal{G} = (V, E)$ with a set of vertices $V = \bigcup_{d=1}^{D} V_d$, $V_d = \{v_{\langle d,1 \rangle}, \ldots, v_{\langle d,n_d \rangle}\}$ and set of edges between the

vertices $E = \bigcup_{d=1}^{D} E_d$, $E_d = \{e_{\langle d-1,i\rangle,\langle d,j\rangle} = (v_{\langle d-1,i\rangle}, v_{\langle d,j\rangle}) \mid v_{\langle d-1,i\rangle} \in V_{d-1} \wedge v_{\langle d,j\rangle} \in V_d\}$, where D represents the length of the longest path in the search graph, which equals the number of parameters, and n_d represents the number of discrete values of a parameter p_d.

In Figure 1 we see that $v_{\langle 1,1\rangle}$ represents the first discrete value of the first parameter and $v_{\langle 1,2\rangle}$ represents the second discrete value, and so on. Every vertex at distance $d-1$ is connected to all the vertices at distance d. With this search graph we have covered the whole solution space of the discrete multi-parameter problem. For example, if we start in the *start* vertex ($d = 0$) and follow the search graph to the ending vertex ($d = D$) we always get a path of length D. This path consists of D vertices and each vertex belongs to one of the parameters. So what we have here is one possible solution of the multi-parameter function. In this way we can create any solution from the solution space of a discrete problem. The efficiency of the path depends on how good is the result obtained by these (found on the path) parameter values. We call this "translated problem" – the problem of finding the cheapest path. This type of solution creation is very suited to the ant-based approach. One more thing that we have to do is to define two values for each vertex. In our case each vertex has two different types of attributes: one is a constant and represents the discrete parameter value, while the other is a variable and represents the amount of pheromone, τ. On this kind of search graph we ran our optimization algorithm – the so-called *Ant Stigmergy Algorithm* (ASA).

3 THE ANT STIGMERGY ALGORITHM

The basic concept, as can be seen from the previous section, is as follows: first, we translate the multi-parameter problem into a search graph and then use an optimization technique to find the cheapest path in the constructed graph; this path consists of the values of the optimized parameters. In our case we use an ant stigmergy optimization algorithm, the routes of which can be found in the ant-colony optimization (ACO) method [5]. The ASA consists of three main phases: initialization, optimization and local search.

3.1 Initialization

Let us start with initialization. Here we translate the parameters of the problem into a search graph. This way we translate the multi-parameter problem into a problem of finding the cheapest path. Figure 1 shows how this is done. We can see that for each parameter $p_d, d = 1, \ldots, D$, parameter value $v_{\langle d,i\rangle}, i = 1, \ldots, n_d$, $n_d = |p_d|$, represents one vertex in a search graph, and each vertex is connected to all the vertices that belong to the next parameter p_{d+1}. Once we have translated the multi-parameter problem into one of finding the cheapest path, we can deploy the initial pheromone values on all the vertices. Now we are ready to proceed to the next phase.

3.2 Optimization

Optimization consists of finding the cheapest path. Prior to the actual optimization an initial amount of pheromone, τ^0, is deposited uniformly in all the vertices in the search graph. There are a number of ants in a colony, all of which begin simultaneously from the *start* vertex. The probability with which they choose the next vertex depends on the amount of pheromone on the vertices. Ants use a probability rule to determine which vertex will be chosen next. More specifically, ant α in step d moves from vertex $v_{\langle d-1,i\rangle} \in \{v_{\langle d-1,1\rangle}, \ldots, v_{\langle d-1,n_{d-1}\rangle}\}$ to vertex $v_{\langle d,j\rangle} \in \{v_{\langle d,1\rangle}, \ldots, v_{\langle d,n_d\rangle}\}$ with the probability given by

$$prob_{ij,\alpha}(d) = \frac{\tau_{\langle d,j\rangle}}{\sum_{1\leq k\leq n_d} \tau_{\langle d,k\rangle}},$$

where $\tau_{\langle d,k\rangle}$ is the amount of pheromone on vertex $v_{\langle d,k\rangle}$. The ants repeat this action until they reach the ending vertex. Then, the gathered parameter values of each ant (which can be found on its path) are evaluated. Next, each ant returns to the *start* vertex and on the way it deposits pheromone in the vertices according to the evaluation result: the better the result, the more pheromone is deposited in the vertices, and vice versa. After all the ants have returned to the start vertex, a so-called daemon action is made, which in this case consists of depositing some additional pheromone on what is currently the best path and also a smaller amount in neighboring vertices. Afterwards, pheromone evaporation from all the vertices occurs, i.e., the amount of pheromone is decreased by some predetermined percentage, ρ, in each vertex $v_{\langle d,k\rangle}$ in the search graph $\mathcal{G}$:

$$\tau^{\text{NEW}}_{\langle d,k\rangle} = (1-\rho)\tau^{\text{OLD}}_{\langle d,k\rangle}.$$

The whole procedure is then repeated until some ending condition is met (e.g., some predetermined number of iterations).

3.3 Local Search

A local search has become a mandatory addition to any ant-based algorithm [8]. By using a local search it is usually possible to improve the convergence or improve the best solution, P^*, found so far. We use it because our basic search techniques is oriented more toward finding the best area of the solution space. We can say that the search is of a broader type, so a local search is used to improve the best solution. In our case a type of steepest-descent algorithm was used. The pseudo code is as follows:

$P^* = \{p_1, \ldots, p_i, \ldots, p_D\}$
change = T
while change **do**
  change = F

for $i = 1$ **to** D **do**
if Evaluate$(\{p_1, \ldots, p_i + p_i^{\text{step}}, \ldots, p_D\})$ < Evaluate(P^*) **then**
change = T
while Evaluate$(\{p_1, \ldots, p_i + p_i^{\text{step}}, \ldots, p_D\})$ < Evaluate(P^*) **do**
$P^* = \{p_1, \ldots, p_i + p_i^{\text{step}}, \ldots, p_D\}$
$p_i = p_i + p_i^{\text{step}}$
endwhile
elseif Evaluate$(\{p_1, \ldots, p_i - p_i^{\text{step}}, \ldots, p_D\})$ < Evaluate(P^*) **then**
change = T
while Evaluate$(\{p_1, \ldots, p_i - p_i^{\text{step}}, \ldots, p_D\})$ < Evaluate(P^*) **do**
$P^* = \{p_1, \ldots, p_i - p_i^{\text{step}}, \ldots, p_D\}$
$p_i = p_i - p_i^{\text{step}}$
endwhile
endif
endfor
endwhile

3.4 Algorithm

Finally, the outline of the *Ant-Stigmergy Algorithm* (ASA) pseudo code is as follows:

```
searchGraph = Initialization(parameters)
SearchGraphInitialization(initial pheromone amount)
while not current level ending condition do
    for all ants do
        path = FindPath(searchGraph)
        Evaluate(path)
    endfor
    UpdatePheromone(all ants paths vertices)
    DaemonAction(best path)
    EvaporatePheromone(all vertices)
endwhile
LocalSearch(best solution)
```

When we ran the ASA on small search graphs ($n_d \ll 100$) the results were encouraging. But when we tried it on real problems [15] or functions (see Section 6.1), which generate much larger graphs, it turned out that there the convergence was slow and the results were poor. Therefore, we decided to apply a *multilevel approach*.

4 MULTILEVEL APPROACH

We consider the multilevel approach and its potential to aid the solution of optimization problems. The multilevel approach is a simple one, which in its most basic form involves recursive coarsening to create a hierarchy of approximations to the

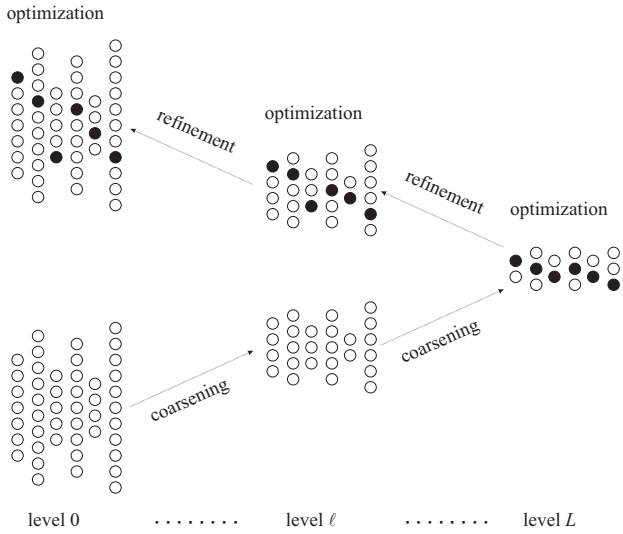


Fig. 2. Multilevel approach on graph structure (the edges are omitted to make the figure clear)

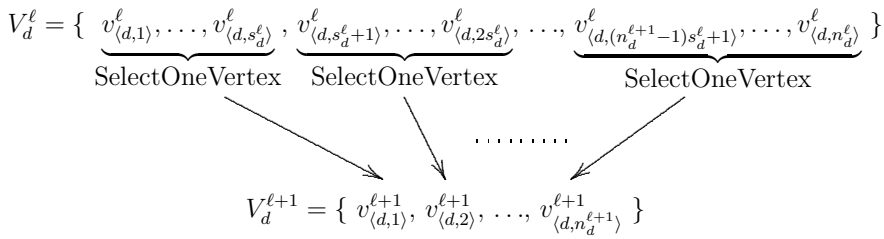
original problem. An initial solution is found (sometimes for the original problem, sometimes at the coarsest level) and then iteratively refined at each level. As a general solution strategy the multilevel procedure has been in use for many years and has been applied to many problem areas.

However, with the exception of the graph-partitioning problem [13, 17], multilevel techniques with conjunction with ant-colony optimization have not been widely applied to combinatorial optimization problems [29].

The multilevel approach consists of two main phases: coarsening and refinement. In our case we will concentrate on graph coarsening and refinement (Figure 2), but it can be used on any other structure.

4.1 Coarsening

Coarsening is done by merging two or more neighboring vertices into a single vertex; this is done in L iterations (we call them levels $\ell = 0, 1, \ldots, L$). Let us consider coarsening from level ℓ to level $\ell+1$ at a distance d. Here $V_d^\ell = \{v_{\langle d,1\rangle}^\ell, \ldots, v_{\langle d,n_d^\ell\rangle}^\ell\}$ is a set of vertices at level ℓ and distance d of the search graph $\mathcal{G}$, where $1 \leq d \leq D$. If n_d^1 is the number of vertices at a starting level of coarsening and a distance d, then for every level ℓ the equation $n_d^{\ell+1} = \lceil \frac{n_d^\ell}{s_d^\ell} \rceil$ is true, where s_d^ℓ is the number of vertices at level ℓ, which are merged into one vertex at level $\ell + 1$.



So what we do is we divide V_d^ℓ into $n_d^{\ell+1}$ subsets, where $V_d^\ell = \bigcup_{k=1}^{n_d^{\ell+1}} V^\ell_{\langle d,k\rangle}$, $\forall i,j \in \{1,\ldots,n_d^{\ell+1}\} \wedge i \neq j : V^\ell_{\langle d,i\rangle} \cap V^\ell_{\langle d,j\rangle} = \emptyset$. Each subset is defined as follows:

$$\begin{aligned} V^\ell_{\langle d,1\rangle} &= \{v^\ell_{\langle d,1\rangle}, \ldots, v^\ell_{\langle d,s^\ell_d\rangle}\}, \\ V^\ell_{\langle d,2\rangle} &= \{v^\ell_{\langle d,s^\ell_d+1\rangle}, \ldots, v^\ell_{\langle d,2s^\ell_d\rangle}\}, \\ &\vdots \\ V^\ell_{\langle d,n^{\ell+1}_d\rangle} &= \{v^\ell_{\langle d,(n^{\ell+1}_d-1)s^\ell_d+1\rangle}, \ldots, v^\ell_{\langle d,n^\ell_d\rangle}\}. \end{aligned}$$

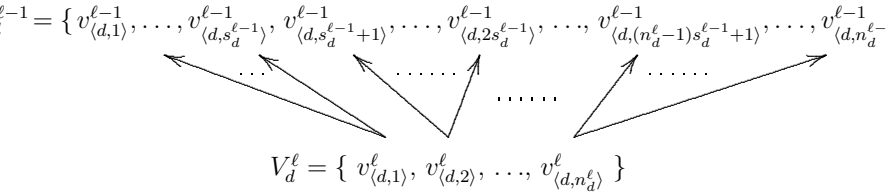
Set $V_d^{\ell+1} = \{v^{\ell+1}_{\langle d,1\rangle}, \ldots, v^{\ell+1}_{\langle d,n^{\ell+1}_d\rangle}\}$ is the set of vertices at distance d at level $\ell+1$, where $v^{\ell+1}_{\langle d,k\rangle} \in V^\ell_{\langle d,k\rangle}$ is selected on some predetermined principle. For example, random pick, the most left/right/centered vertex in the subset, etc.

The outline of the coarsening pseudo code from V_d^ℓ to $V_d^{\ell+1}$ is as follows:

for $k = 1$ **to** $n_d^{\ell+1}$ **do**
  $v^{\ell+1}_{\langle d,k\rangle} = \text{SelectOneVertex}(V^\ell_{\langle d,k\rangle})$
endfor

4.2 Refinement

Because of the simplicity of the coarsening, the refinement itself is very trivial. Let us consider refinement from level l to level $l-1$ at distance d.



The outline of the refinement pseudo code is as follows:

for $k = 1$ **to** n_d^ℓ **do**
  for each $v^{\ell-1}_{\langle d,i\rangle} \in V^{\ell-1}_{\langle d,k\rangle}$ **do**
    $v^{\ell-1}_{\langle d,i\rangle} = \text{CopyVariables}(v^\ell_{\langle d,i\rangle})$ // in our case $\tau^{\ell-1}_{\langle d,i\rangle} = \tau^\ell_{\langle d,k\rangle}$
  endfor
endfor

Here the variable vertex attributes (in our case the amount of pheromone), as a result of optimization at level ℓ, are transferred to level $\ell - 1$ with the use of the CopyVariables function. Therefore, each vertex of subset $V^{\ell-1}_{\langle d,k\rangle}$ is assigned with the same value of variable attributes, which corresponds to vertex $v^{\ell}_{\langle d,k\rangle}$ that was chosen in the coarsening from level $\ell - 1$ to level ℓ, while the constant vertex attributes remain the same.

4.3 The Algorithm

Finally, the outline of the multilevel algorithm pseudo code could look like this:

```
structure[0] = Initialization
for ℓ = 0 to L − 1 do
    structure[ℓ + 1] = Coarsening(structure[ℓ])
endfor
for ℓ = L downto 0 do
    Solver(structure[ℓ])                    // e.g., optimization algorithm
    if ℓ > 0 then
        structure[ℓ − 1] = Refinement(structure[ℓ])
    endif
endfor
```

5 THE MULTILEVEL ANT STIGMERGY ALGORITHM

It is now time to merge the previously mentioned algorithms into one. This approach is called the *Multilevel Ant Stigmergy Algorithm* (MASA). The MASA consists of five main phases: initialization, coarsening, optimization, refinement, and local search. Each phase is exactly the same as described in the previous sections.

The outline of the MASA pseudo code is as follows:

```
searchGraph[0] = Initialization(parameters)
for ℓ = 0 to L − 1 do
    searchGraph[ℓ + 1] = Coarsening(searchGraph[ℓ])
endfor
SearchGraphInitialization(initial pheromone amount)
for ℓ = L downto 0 do
    while not current level ending condition do
        for all ants do
            path = FindPath(searchGraph[ℓ])
            Evaluate(path)
        endfor
        UpdatePheromone(all ants paths vertices)
        DaemonAction(best path)
```

```
        EvaporatePheromone(all vertices)
    endwhile
    if ℓ > 0 then
        searchGraph[ℓ − 1] = Refinement(searchGraph[ℓ])
    endif
endfor
LocalSearch(best solution)
```

6 PERFORMANCE EVALUATION

In this section we analyze the performance of the MASA and compare the MASA to what are currently the best algorithms for solving multi-parameter optimization problems: differential evolution and its descendant, discrete differential evolution. The evaluation is performed on a set of numerical benchmark functions.

6.1 Benchmark Functions

For the benchmark functions we have decided to use sphere, f_{Sp}, Griewangk, f_{Gr}, Rastrigin, f_{Rt}, Rosenbrock, f_{Rb}, Krink, f_{Kr} and negative Krink, f_{nK}. These functions have been used in a number of earlier investigations on performance evaluation for optimization problems, e.g., see [18]. For evaluation purposes we used three different function dimensions $D = |P| = 5$, 25, and 50. The function definitions are as follows (see also Table 1):

$$f_{\mathrm{Sp}}(P) = \sum_{i=1}^{D} p_i^2,$$

$$f_{\mathrm{Gr}}(P) = \frac{1}{4000}\sum_{i=1}^{D}(p_i - 100)^2 - \prod_{i=1}^{D}\cos\left(\frac{p_i - 100}{\sqrt{i}}\right) + 1,$$

$$f_{\mathrm{Rt}}(P) = \sum_{i=1}^{D}(10 + p_i^2 - 10\cos(2\pi p_i)),$$

$$f_{\mathrm{Rb}}(P) = \sum_{i=1}^{D-1}(100(p_{i+1} - p_i^2)^2 + (p_i - 1)^2),$$

$$f_{\mathrm{Kr}}(P) = \sum_{i=1}^{D}(-37.816415 - |p_i - 50| + 40\sin(\frac{5\pi p_i}{18})),$$

$$f_{\mathrm{nK}}(P) = \sum_{i=1}^{D}(-89.016293 + |p_i - 50| - 40\sin(\frac{5\pi p_i}{18})).$$

The optimization of noisy functions is a common task occurring in various applications. In some applications the function to be minimized is only known to a low

Function	L_i	U_i	Δ	**Minimum value**
$f_{\mathrm{Sp}}(P)$	-100	100	10^{-3}	$f_{\mathrm{Sp}}(\overrightarrow{0}) = 0$
$f_{\mathrm{Gr}}(P)$	-600	600	10^{-2}	$f_{\mathrm{Gr}}(\overrightarrow{100}) = 0$
$f_{\mathrm{Rt}}(P)$	-5.12	5.12	10^{-4}	$f_{\mathrm{Rt}}(\overrightarrow{0}) = 0$
$f_{\mathrm{Rb}}(P)$	-50	50	10^{-3}	$f_{\mathrm{Rb}}(\overrightarrow{1}) = 0$
$f_{\mathrm{Kr}}(P)$	0	100	10^{-3}	$f_{\mathrm{Kr}}(\approx \overrightarrow{52.167}) \approx 0$
$f_{\mathrm{nK}}(P)$	0	100	10^{-3}	$f_{\mathrm{nK}}(\approx \overrightarrow{99.031}) \approx 0$

Table 1. Function constraints and minimum values

precision. For the purpose of simulating this problem we introduce noisy versions of the benchmark functions that are defined as

$$\tilde{f}(P, s) = \frac{1}{s} \sum_{i=1}^{s} (f(P) + \mathrm{Gauss}(0, 1)),$$

where s is the number of samples (evaluations with added noise) needed to compute a noisy function, and $\mathrm{Gauss}(0, 1)$ is a Gaussian distribution with a mean of 0 and a standard deviation of 1. For evaluation purposes we used three different degrees of sampling $s = 10$, 50, and 100.

Function	D	**Best**	**Mean**	**Std**	**Avg iter**
f_{Sp}	5	0	0	0	9 693
f_{Gr}	5	0	$0.616\ 10^{-1}$	$0.598\ 10^{-1}$	11 337
f_{Rt}	5	0	0	0	8 875
f_{Rb}	5	$0.133\ 10^{-1}$	$0.282\ 10^{-1}$	$0.104\ 10^{-1}$	80 227
f_{Kr}	5	$0.136\ 10^{-5}$	4.733	3.514	15 741
f_{nK}	5	$-0.406\ 10^{-3}$	5.613	5.334	21 610
f_{Sp}	25	$0.500\ 10^{-5}$	$0.920\ 10^{-5}$	$0.202\ 10^{-5}$	22 747
f_{Gr}	25	$0.296\ 10^{-4}$	$0.148\ 10^{-1}$	$0.140\ 10^{-1}$	30 653
f_{Rt}	25	$0.119\ 10^{-4}$	0.696	0.911	31 979
f_{Rb}	25	9.712	31.340	36.310	466 687
f_{Kr}	25	$0.980\ 10^{-4}$	3.547	3.955	58 965
f_{nK}	25	$-0.159\ 10^{-2}$	4.691	5.996	56 518
f_{Sp}	50	$0.340\ 10^{-2}$	$0.460\ 10^{-2}$	$0.839\ 10^{-3}$	27 343
f_{Gr}	50	$0.142\ 10^{-3}$	$0.356\ 10^{-2}$	$0.610\ 10^{-2}$	46 228
f_{Rt}	50	$0.714\ 10^{-4}$	0.663	1.149	55 601
f_{Rb}	50	38.844	80.789	46.993	348 816
f_{Kr}	50	$0.480\ 10^{-3}$	3.828	4.502	87 853
f_{nK}	50	$-0.155\ 10^{-2}$	3.224	4.464	86 534

Table 2. Experimental results of the MASA without local search on non-noisy functions

The problem of dealing with noisy functions has been addressed by various researchers, mainly for evolution strategies [21], evolution programming [7], genetic algorithms [11], particle swarm optimization [14], and differential evolution [18].

6.2 Performance of the MASA

We ran the MASA 30 times on each experiment. The number of maximum function evaluations per experiment N was set to 500 000. The number of ants was 10, $\rho = 0.1$, and the coarsening was implemented by merging two vertices into one (which defines the number of levels L). With regard to the ending condition for each level, we have two different policies. In the case of non-noisy functions, the ending condition was set to "no best solution found for the last 50 iterations", while in the case of noisy functions we limited the number of evaluations per level to $\frac{N}{L}$. This way we ensure that the algorithm does not stay too long on coarse-grained graphs, i.e., levels with high ℓ. We must note that during the experimentation we did not fine-tune the algorithms parameters, but only made a limited number of experiments to find satisfying settings.

Function	D	**Best**	**Mean**	**Std**	**Avg iter**
f_{Sp}	5	0	0	0	9 703
f_{Gr}	5	0	$0.616\ 10^{-1}$	$0.598\ 10^{-1}$	11 347
f_{Rt}	5	0	0	0	8 885
f_{Rb}	5	$0.133\ 10^{-1}$	$0.280\ 10^{-1}$	$0.102\ 10^{-1}$	80 246
f_{Kr}	5	$0.136\ 10^{-5}$	4.733	3.514	15 751
f_{nK}	5	$-0.609\ 10^{-3}$	5.613	5.334	21 626
f_{Sp}	25	0	0	0	22 852
f_{Gr}	25	0	$0.148\ 10^{-1}$	$0.140\ 10^{-1}$	30 761
f_{Rt}	25	0	0.696	0.911	32 084
f_{Rb}	25	$0.174\ 10^{-1}$	0.949	2.636	500 000
f_{Kr}	25	$0.681\ 10^{-5}$	3.547	3.955	59 069
f_{nK}	25	$-0.304\ 10^{-2}$	4.690	5.997	56 639
f_{Sp}	50	0	0	0	27 562
f_{Gr}	50	0	$0.328\ 10^{-2}$	$0.608\ 10^{-2}$	46 472
f_{Rt}	50	0	0.663	1.149	55 824
f_{Rb}	50	$0.744\ 10^{-1}$	5.126	18.595	500 000
f_{Kr}	50	$0.136\ 10^{-4}$	3.827	4.502	88 073
f_{nK}	50	$-0.609\ 10^{-2}$	3.221	4.465	86 784

Table 3. Experimental results of the MASA with local search on non-noisy functions

The evaluation results of the MASA (with and without local search) on non-noisy functions are presented in Table 2 and Table 3, where the best and the average solutions obtained in 30 runs are shown for each experiment. The standard deviation of the solutions and the average number of function evaluations per experiment are also included in the table. The global minimum of functions f_{Sp}, f_{Gr}, f_{Rt}, and f_{Rb} is exactly zero, while for f_{Kr} and f_{nK} we set constants so that the global minimum is as close to zero possible, see Table 1.

In Tables 2 and 3 we can see that on almost all functions and dimensions the MASA found an optimal or near-optimal solution. The only function where it

performed worse was f_{Rb}. The main reason for this is that the first expression $100(p_{i+1} - p_i^2)^2$ has two global minima at $p_i = 0$ and $p_i = 1$, $i = 1, 2, \ldots, D$, while the second expression $(p_i - 1)^2$ has only one global minimum at $p_i = 1$. The p_i^2 in the first expression prefers $p_i = 0$ over $p_i = 1$. Since the first expression dominates over the second, the MASA is at first misled into solution $p_i = 0$, from where it can slowly move toward the global minimum at $p_i = 1$, $i = 1, 2, \ldots, D$.

Function	s	**Best**	**Mean**	**Std**	**Avg iter**
$\tilde{f}_{\mathrm{Sp}}$	10	−0.433	0.261	0.314	500 000
$\tilde{f}_{\mathrm{Gr}}$	10	0.791	1.646	0.527	500 000
$\tilde{f}_{\mathrm{Rt}}$	10	6.113	10.589	2.511	500 000
$\tilde{f}_{\mathrm{Rb}}$	10	46.428	527.665	825.313	500 000
$\tilde{f}_{\mathrm{Kr}}$	10	78.918	119.566	21.214	500 000
$\tilde{f}_{\mathrm{nK}}$	10	132.751	164.080	22.010	500 000
$\tilde{f}_{\mathrm{Sp}}$	50	76.441	212.538	91.326	500 000
$\tilde{f}_{\mathrm{Gr}}$	50	2.946	5.131	1.235	500 000
$\tilde{f}_{\mathrm{Rt}}$	50	22.213	74.578	19.105	500 000
$\tilde{f}_{\mathrm{Rb}}$	50	3 051.539	57 664.280	43 795.415	500 000
$\tilde{f}_{\mathrm{Kr}}$	50	352.753	425.406	26.116	500 000
$\tilde{f}_{\mathrm{nK}}$	50	667.893	764.757	54.007	500 000
$\tilde{f}_{\mathrm{Sp}}$	100	1 062.354	2 337.157	761.205	500 000
$\tilde{f}_{\mathrm{Gr}}$	100	17.040	32.121	8.081	500 000
$\tilde{f}_{\mathrm{Rt}}$	100	114.638	167.781	20.200	500 000
$\tilde{f}_{\mathrm{Rb}}$	100	1 270 827	3 740 176	2 279 957	500 000
$\tilde{f}_{\mathrm{Kr}}$	100	502.813	671.239	68.903	500 000
$\tilde{f}_{\mathrm{nK}}$	100	866.699	1 070.813	80.812	500 000

Table 4. Experimental results of the MASA without local optimization on noisy functions with $D = 50$

To evaluate the performance of the MASA on noisy functions we decided to test it on functions with dimension $D = 50$. Table 4 shows the results of the MASA without local search on the noisy functions. One can see that with an increase of the degree of sampling, s, the results deteriorate noticeably.

6.3 Performance of the DE

The evolutionary algorithm, the particle swarm optimization, and differential evolution (DE) are very popular numerical optimization procedures. The results reported in [18, 28] show that DE generally outperforms the other algorithms. Therefore, we decided to compare the MASA with DE.

Differential evolution is a stochastic, population-based optimization algorithm. It was introduced by Storn and Price [25] and was developed to optimize the real (float) parameters of a real-valued function. DE resembles the structure of an evolutionary algorithm, but differs from traditional evolutionary algorithms in its ge-

Function	D	Best	Mean	Std	Avg iter
f_{Sp}	5	0	0	0	8 785
f_{Gr}	5	0	0	0	39 697
f_{Rt}	5	0	0	0	18 222
f_{Rb}	5	0	$0.315\ 10^{-7}$	$0.131\ 10^{-6}$	84 646
f_{Kr}	5	$0.742\ 10^{-4}$	$0.742\ 10^{-4}$	0	500 000
f_{nK}	5	0.418	8.100	8.466	500 000
f_{Sp}	25	0	0	0	52 230
f_{Gr}	25	0	$0.986\ 10^{-3}$	$0.308\ 10^{-2}$	100 140
f_{Rt}	25	0.995	14.307	14.083	500 000
f_{Rb}	25	0	$0.139\ 10^{-1}$	$0.745\ 10^{-1}$	476 097
f_{Kr}	25	14.000	209.900	86.522	500 000
f_{nK}	25	19.795	87.365	39.688	500 000
f_{Sp}	50	0	0	0	105 560
f_{Gr}	50	0	$0.493\ 10^{-3}$	$0.188\ 10^{-2}$	132 061
f_{Rt}	50	11.940	98.290	43.517	500 000
f_{Rb}	50	15.188	37.273	14.522	500 000
f_{Kr}	50	600.382	921.951	156.130	500 000
f_{nK}	50	97.104	228.734	65.940	500 000

Table 5. Experimental results of the DE on non-noisy functions

neration of new candidate solutions and by its use of a "greedy" selection scheme. The basic idea of DE is outlined as follows:

```
population = RndCreate(parameters)
Evaluate(population)
while ending condition do
    for each parent from population do
        candidates[1..3] = RndSelect(population)
        newCandidate = Calculate(candidates[1..3])
        BinomialCrossover(parent, newCandidate)
        Evaluate(newCandidate)
        if (newCandidate better than parent) then
            parent = newCandidate
        endif
    endfor
    RndEnumerate(population)
endwhile
```

The newCandidate is calculated as a weighted sum of three randomly chosen candidates that are different from the parent. Only then does the parent participate in the creation of the candidate – the candidate is modified by a crossover with its parent. Finally, the candidate is evaluated and compared to the parent. The candidate replaces the parent in the population, only if it is better than the parent. The described procedure (body of the for loop in the above algorithm) is repeated for

all the parent individuals from the population. When it is finished the individuals from the population are randomly enumerated and the procedure is repeated.

Recall that DE is used on continuous problems, while the MASA works on discrete problems.

The DE has three parameters, which were set to the following values, as proposed by Krink et al. [18]: the population size was 50, the crossover constant was 0.8, and the scaling factor was 0.5. We ran the DE on each test function 30 times. The maximum number of function evaluations per experiment was set to 500 000.

In Table 5 we can see that for f_{Sp} and f_{Gr} the DE returns near optimal results for all dimensions; for f_{Rt} and f_{Kr} it returns near optimal results only for $D = 5$; for f_{Rb} it returns near optimal results for $D < 50$; for f_{nK} the DE does not produce any good results in 500 000 evaluations.

Function	s	**Best**	**Mean**	**Std**	**Avg iter**
$\tilde{f}_{\mathrm{Sp}}$	10	−0.656	−0.393	0.117	500 000
$\tilde{f}_{\mathrm{Gr}}$	10	$0.387\ 10^{-1}$	0.436	0.130	500 000
$\tilde{f}_{\mathrm{Rt}}$	10	338.419	372.889	16.961	500 000
$\tilde{f}_{\mathrm{Rb}}$	10	40.450	66.248	32.114	500 000
$\tilde{f}_{\mathrm{Kr}}$	10	1 549.370	1 700.032	72.681	500 000
$\tilde{f}_{\mathrm{nK}}$	10	106.612	252.194	92.002	500 000
$\tilde{f}_{\mathrm{Sp}}$	50	807.386	1 358.840	463.151	500 000
$\tilde{f}_{\mathrm{Gr}}$	50	9.055	13.071	3.079	500 000
$\tilde{f}_{\mathrm{Rt}}$	50	409.340	453.605	21.390	500 000
$\tilde{f}_{\mathrm{Rb}}$	50	1, 708, 481	4 523 049	2 358 303	500 000
$\tilde{f}_{\mathrm{Kr}}$	50	1, 715.753	1 963.043	81.910	500 000
$\tilde{f}_{\mathrm{nK}}$	50	634.912	910.486	121.962	500 000
$\tilde{f}_{\mathrm{Sp}}$	100	7 057.189	12 035.640	2 571.304	500 000
$\tilde{f}_{\mathrm{Gr}}$	100	62.169	108.015	21.525	500 000
$\tilde{f}_{\mathrm{Rt}}$	100	439.637	500.555	24.734	500 000
$\tilde{f}_{\mathrm{Rb}}$	100	39 842 415	122 003 535	57 879 863	500 000
$\tilde{f}_{\mathrm{Kr}}$	100	1 977.308	2 108.498	73.489	500 000
$\tilde{f}_{\mathrm{nK}}$	100	954.527	1 165.232	95.198	500 000

Table 6. Experimental results of the DE on noisy functions with $D = 50$

Table 6 shows the results of the DE on noisy functions. Like in the case of the MASA, we tested the DE on functions with the dimension $D = 50$. Here we also notice a deterioration of the results. The impact of the higher degree of sampling, $s = 50$ and $s = 100$, on the performance of the DE is greater than for the MASA.

A detailed comparison between the presented algorithms will be given in the next section.

6.4 Algorithms Convergence Comparison

A convergence comparison of the DE and the MASA on non-noisy functions can be seen in Figures 3–5, where the graphs show the mean performance of 30 runs for each function.

Figure 3 shows the functions with $D = 5$. Here we do not notice any large performance differences between the algorithms. The algorithms' average returned results and convergence are approximately the same. For higher dimensions, $D = 25$ (Figure 4) and $D = 50$ (Figure 5), we observe, with the exception of f_{Rb}, that the MASA outperforms the DE.

The convergence comparison of the DE and the MASA on noisy functions with $D = 50$ can be seen in Figures 6–8. For a higher degree of sampling, $s = 50$ (Figure 7) and $s = 100$ (Figure 8), we observe that the MASA outperforms the DE. With the MASA one can see a cascading approach toward the optimal solution. The reason for this is the constant number of function evaluations at each level.

6.5 Comparison to Other Ant Methods

As we mentioned in the introduction, there are few other adaptations of ACO algorithm to real-parameter optimization. Here, the MASA is compared to results presented by Socha [24]. In order to have comparable results, the same accuracy level was chosen.

The results presented in Table 7 are based on 30 independent runs of the MASA and show number of function evaluations to achieve the fixed accuracy level. The experimental results show that the MASA (without local optimization) has much higher convergence speed than that of CACO [1] and comparable with eACO [24].

Test Function*	D	**accuracy**	**CACO**	**CIAC**	**eACO**	**MASA**
Sphere	6	10^{-4}	22 050	50 000	695	361
Goldstein & Price	2	10^{-4}	5 320	23 391	364	142
Rosenbrock	2	10^{-3}	6 842	11 797	2 905	188
Zakharov	2	10^{-4}	—	—	401	448

* http://iridia.ulb.ac.be/~ksocha/extaco04.html

Table 7. Comparison of average number of function evaluations until the accuracy is reached

6.6 Algorithm Complexity

The algorithm's complexity is estimated as suggested in [26] by calculating the following formula $\frac{\widehat{T}_2 - T_1}{T_0}$, where computing time T_0 is independent of the function dimension and is calculated by running the program below:

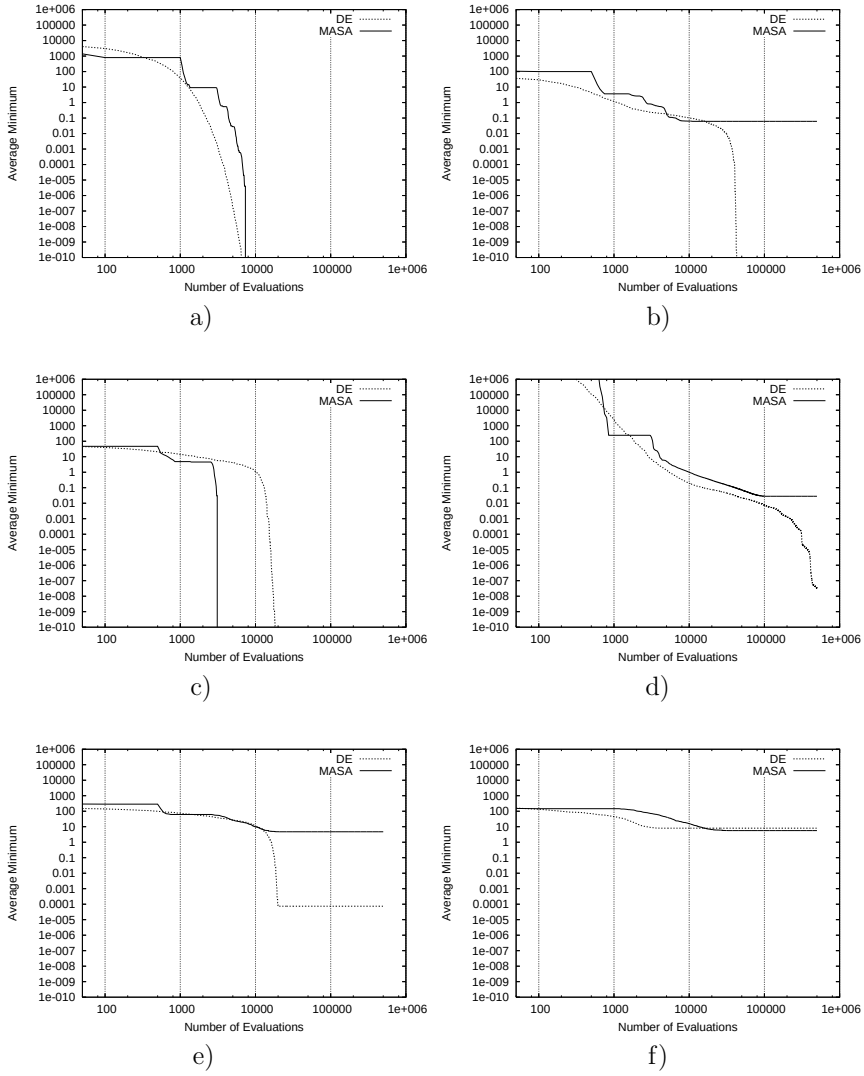


Fig. 3. Average minimum of a) sphere, b) Griewangk, c) Rastrigin, d) Rosenbrock, e) Krink, and f) negative Krink functions with $D = 5$

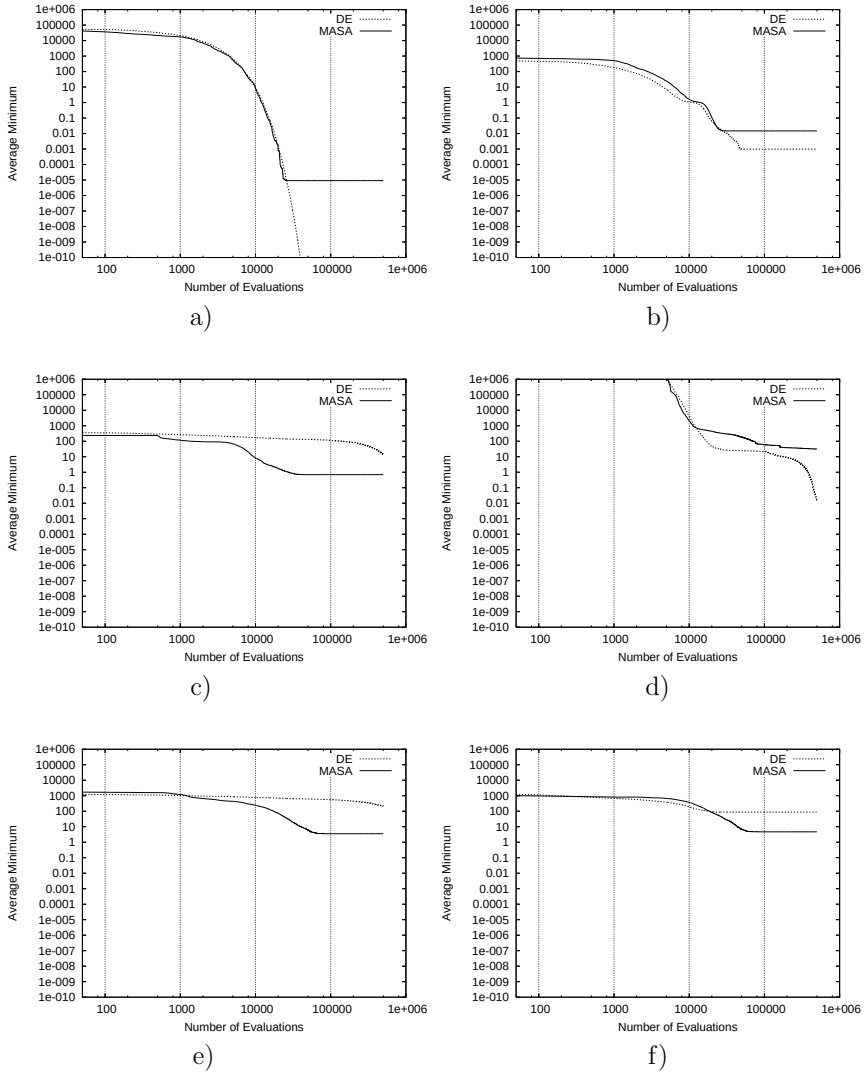


Fig. 4. Average minimum of a) sphere, b) Griewangk, c) Rastrigin, d) Rosenbrock, e) Krink, and f) negative Krink functions with $D = 25$

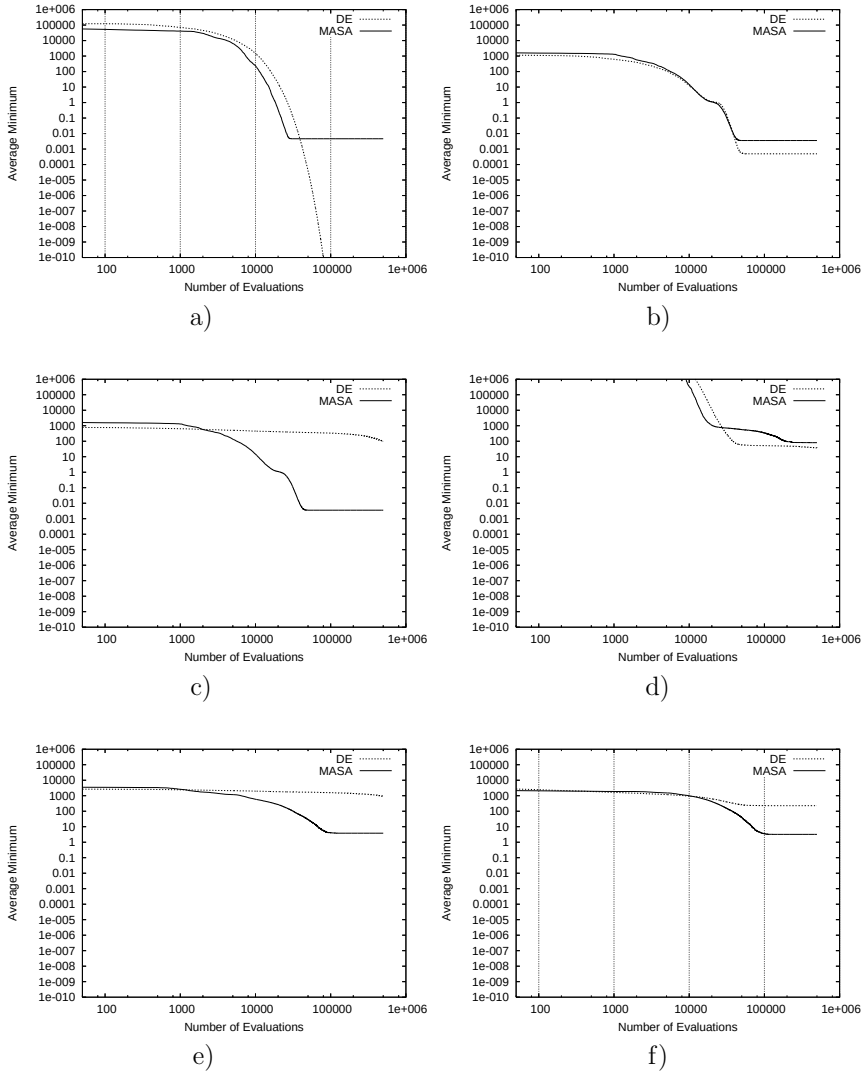


Fig. 5. Average minimum of a) sphere, b) Griewangk, c) Rastrigin, d) Rosenbrock, e) Krink, and f) negative Krink functions with $D = 50$

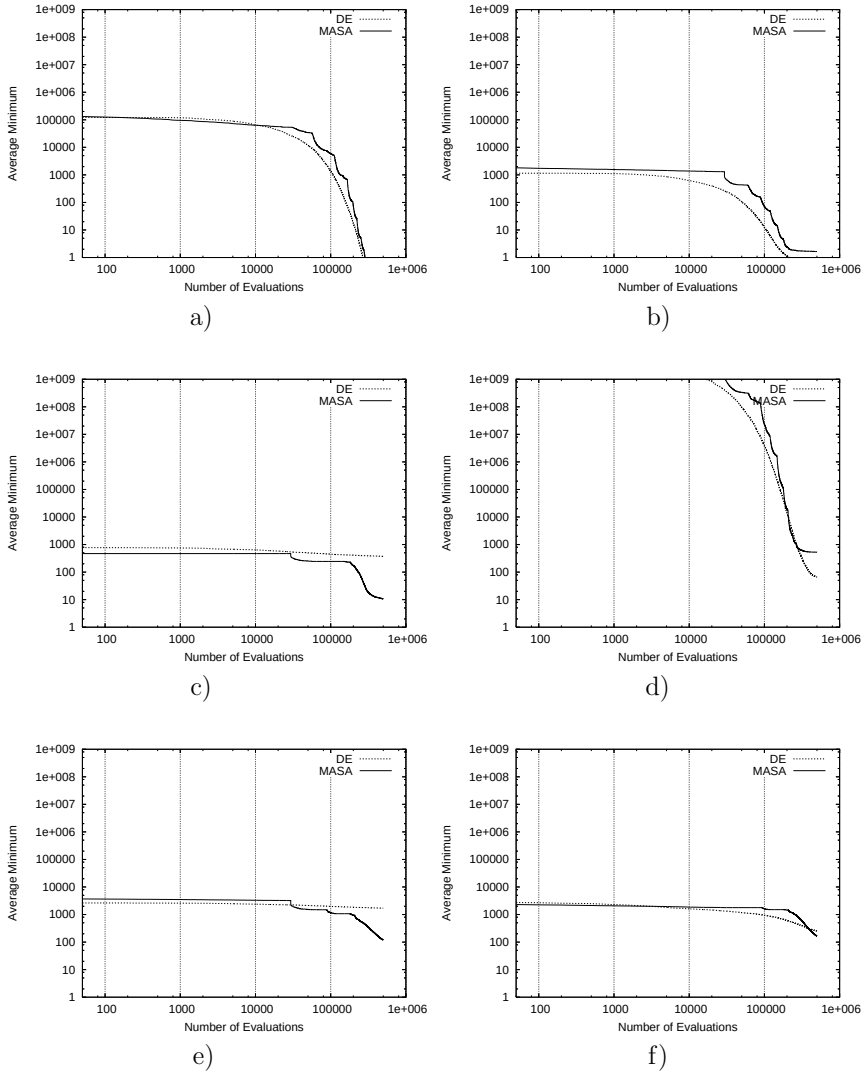


Fig. 6. Average minimum of a) sphere, b) Griewangk, c) Rastrigin, d) Rosenbrock, e) Krink, and f) negative Krink functions with $s = 10$ and $D = 50$

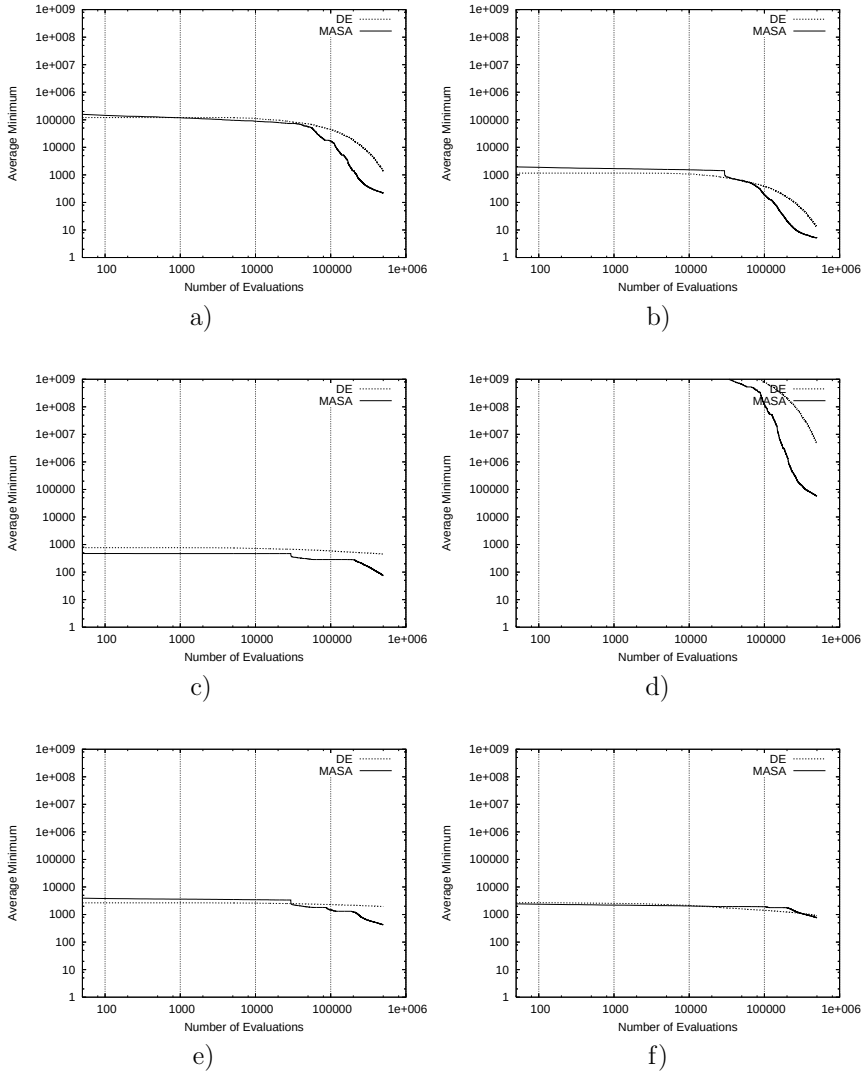


Fig. 7. Average minimum of a) sphere, b) Griewangk, c) Rastrigin, d) Rosenbrock, e) Krink, and f) negative Krink functions with $s = 50$ and $D = 50$

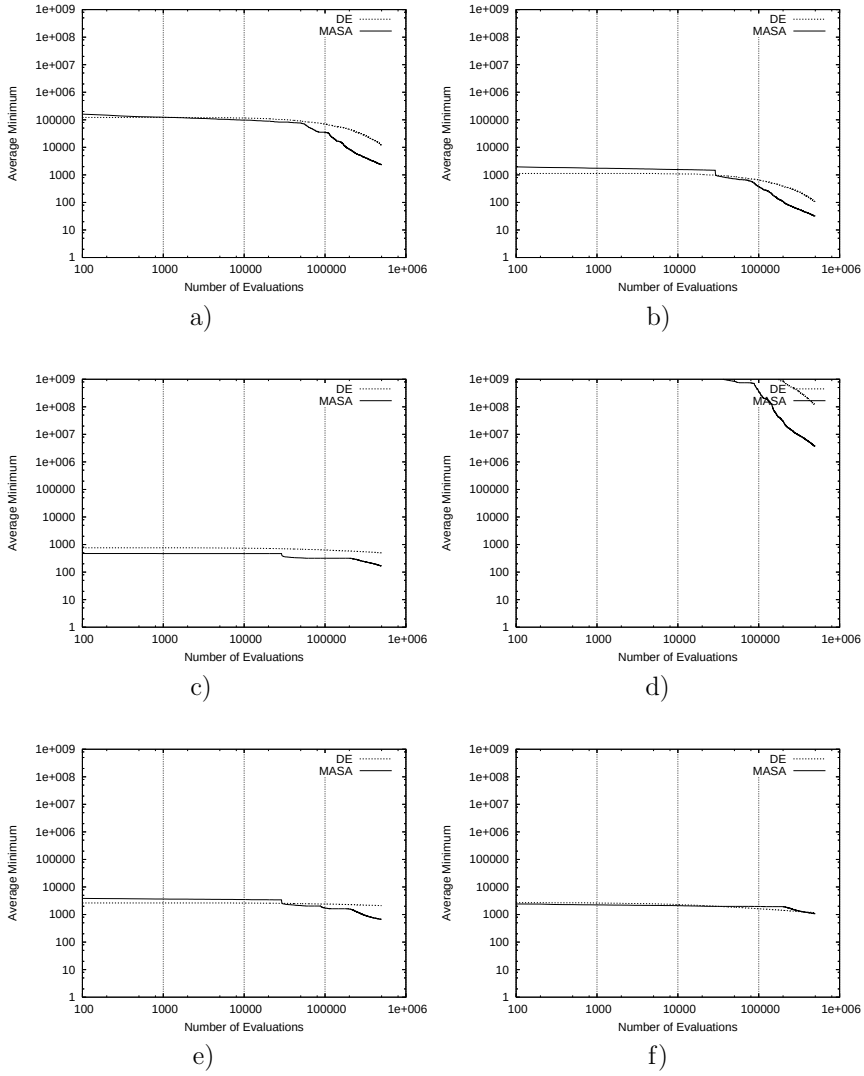


Fig. 8. Average minimum of a) sphere, b) Griewangk, c) Rastrigin, d) Rosenbrock, e) Krink, and f) negative Krink functions with $s = 100$ and $D = 50$

```
for i = 1 to 1 000 000 do
    x = (double) 5.55
    x = x + x
    x = x * x
    x = sqrt(x)
    x = ln(x)
    x = exp(x)
    y = x/x
endfor
```

T_1 is the computing time for 200 000 evaluations just for function f_{Rb} and $\widehat{T}_2$ is the mean time of five executions, but now considering the complete computing time of the algorithm for function f_{Rb}. The results presented in Table 8 show that MASA has a higher complexity then the DE, but when dealing with real-world problems this deficiency becomes insignificant compared to time needed to compute a single evaluation of cost function [15].

Algorithm	T_0 [s]	T_1 [s]	$\widehat{T}_2$ [s]	$\frac{\widehat{T}_2 - T_1}{T_0}$
MASA w/o LS	0.2	3.0	44.0	205
DE	0.2	3.0	6.0	15

Table 8. Algorithm complexity (function f_{Rb}, $D = 50$)

7 CONCLUSIONS

In this paper we presented a new method based on stigmergy for solving numerical (multi-parameter) optimization problems. Stigmergy is a type of collective work that can be observed in an ant colony.

We proposed a general approach for the translation of a multi-parameter problem into a search graph representation. Each longest path in the search graph represents one possible solution, and all longest paths together represent the whole solution space of the multi-parameter problem. For an efficient search of the solution space we used a multilevel approach. We call this method the Multilevel Ant Stigmergy Algorithm.

We evaluated the performance of the Multilevel Ant Stigmergy Algorithm and Differential Evolution in terms of their applicability as numerical optimization techniques. The comparison is performed with several widely used benchmark functions. It was determined that for lower-dimension functions the performance was comparable, while for higher dimensions the Multilevel Ant Stigmergy Algorithm outperformed Differential Evolution in all functions with the exception of one.

Since the optimization of noisy functions is a common problem occurring in various applications we also evaluated the Multilevel Ant Stigmergy Algorithm and Differential Evolution on noisy versions of the benchmark functions. For evaluation

purposes we used three different degrees of sampling: low, middle, and high. We observed that the impact of the higher degree on the performance of Differential Evolution is greater than for the Multilevel Ant Stigmergy Algorithm.

Here we would like to mention that we used the Multilevel Ant Stigmergy Algorithm in the computer-assisted design of a universal AC or DC motor rotor/stator geometry [15]. The efficiency of an electric motor is defined as the ratio of the output power to the input power and depends on various power losses. They include copper and iron losses, which are significantly affected by the geometry of the rotor and the stator. The optimization task is to find the geometry parameter values that would generate the rotor and the stator geometry with minimum power losses. The average solution obtained with the algorithm was 24.9 % better than the solution recently found using a genetic algorithm [20], and 44.3 % better than the expert's solution currently in industrial production.

However, even if the Multilevel Ant Stigmergy Algorithm offers good quality of solution, it still needs considerable computational time (due to time consuming finite-element method simulation package that takes a couple of minutes per run). With distributed implementation of the Multilevel Ant Stigmergy Algorithm the computation time is drastically decreased (from one day to few hours) without any noticeable loss in solution quality [23].

Following our industrial case studies [15, 16] we can conclude that the Multilevel Ant Stigmergy Algorithm can be used on any real-world problem that can be put into discrete form and have corresponding graph representation.

REFERENCES

[1] Bilchev, G.—Parmee, I. C.: The Ant Colony Metaphor for Searching Continuous Design Spaces. Lecture Notes in Computer Science, Vol. 993, 1995, pp. 25–39.

[2] Chen, L.—Shen, J.—Qin, L.—Fan, J.: A Method for Solving Optimization Problem in Continuous Space Using Improved Ant Colony Algorithm. Lecture Notes in Computer Science, Vol. 3327, 2004, pp. 61–70.

[3] Cook, W. J.—Cunningham, W. H.—Pulleyblank, W. R.—Schrijver, A.: Combinatorial Optimization. John Wiley & Sons, New York, 1997.

[4] Dorigo, M.—Stützle, T.: Ant Colony Optimization. The MIT Press, Cambridge, Massachusetts, 2004.

[5] Dorigo, M.—Di Caro, G.—Gambardella, L. M.: Ant Algorithms for Discrete Optimization. Artificial Life, Vol. 5, 1999, No. 2, pp. 137–172.

[6] Dréo, J.—Siarry, P.: A New Ant Colony Algorithm Using the Heterarchical Concept Aimed at Optimization of Multiminima Continuous Functions. Lecture Notes in Computer Science, Vol. 2463, 2002, pp. 216–227.

[7] Fogel, L. J.—Owens, A. J.—Walsh, M. J.: Artificial Intelligence Through Simulated Evolution. John Wiley & Sons, New York, 1966.

[8] Gambardella, L. M.—Dorigo, M.: An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. INFORMS Journal on Computing, Vol. 12, 2000, No. 3, pp. 237–255.

[9] Grassé, P. P.: La Reconstruction du nid et les Coordinations Inter-Individuelle Chez Bellicositermes Natalensis et Cubitermes sp. La théorie de la Stigmergie: Essai d'Interprétation du Comportement des Termites Constructeurs. Insect Sociaux, Vol. 6, 1959, pp. 41–83.

[10] Hadeli, K.—Valckenaers, P.—Kollingbaum, M.—van Brussel, H.: Multi-Agent Coordination and Control Using Stigmergy. Computers in Industry, Vol. 53, 2004, No. 1, pp. 75–96.

[11] Holland, J. H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.

[12] Holland, O.—Melhuish, C.: Stimergy, Self-Organization, and Sorting in Collective Robotics. Artificial Life, Vol. 5, 1999, No. 2, pp. 173–202.

[13] Karypis, G.—Kumar, V.: Analysis of Multilevel Graph Partitioning. In Proceedings of the ACM/IEEE Supercomputing Conference, pp. 658–677, 1995.

[14] Kennedy, J.—Eberhart, R. C.: Particle Swarm Optimization. In Proceedings of the IEEE Intenational Conference on Neural Networks, pp. 1942–1848, 1995.

[15] Korošec, P.—Šilc, J.: The Multilevel Ant Stigmergy Algorithm: An Industrial Case Study. In Proceedings of the 7$^{\text{th}}$ International Conference on Computational Intelligence and Natural Computing, pp. 475–478, 2005.

[16] Korošec, P.—Šilc, J.—Filipič, B.—Erkki, L.: Ant Stigmergy on the Grid: Optimizing the Cooling Process in Continuous Steel Casting. In Proceedings of the 9th International Workshop on Nature Inspired Distributed Computing, 2006.

[17] Korošec, P.—Šilc, J.—Robič, B.: Solving the Mesh-partitioning Problem with an Ant-colony Algorithm. Parallel Computing, Vol. 30, 2004, No. 5–6, pp. 785–801.

[18] Krink, T.—Filipič, B.—Fogel, G. B.—Thomsen, R.: Noisy Optimization Problems – A Particular Challenge for Diferential Evolution? In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 332–339, 2004.

[19] Nocedal, J.—Wright, S. J.: Numerical Optimization. Springer, New York, Berlin, Heidelberg, 1999.

[20] Papa, G.—Koroušić-Seljak, B.—Benedičič, B.—Kmecl, T.: Universal Motor Efficiency Improvement Using Evolutionary Optimization. IEEE Transactions on Industrial Electronics, Vol. 50, 2003, No. 3, pp. 602–611.

[21] Rechenberg, I.: Evolution Strategies: Optimization of Technical Systems by Means of Biological Evolution. Fromman-Holzboog, Studgart, 1973.

[22] Roth, M.—Wicker, S.: Termite: Ad-Hoc Networking with Stigmergy. In Proceedings of the IEEE Global Communications Conference, 2003.

[23] Šilc, J.—Korošec, P.: The Distributed Stigmergic Algorithm for Multiparameter Optimization. Lecture Notes in Computer Science, Vol. 3911, pp. 92–99, 2006.

[24] Socha, K.: ACO for Continuous and Mixed-Variable Optimization. Lecture Notes in Computer Science, Vol. 3172, pp. 25–36, 2004.

[25] Storn, R.—Price, K.: Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. Technical Report TR-95-012, ICSI, Berkley, CA, 1995.

[26] Sunganthan, P. N.—Hansen, N.—Liang, J. J.—Chen, Y. P.—Auger, A.—Tiwari, S.: Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report, Nanyang Technological University, Singapore, May 2005.

[27] Theraulaz, G.—Bonabeau, E.: A Brief History of Stigmergy. Artificial Life, Vol. 5, 1999, No. 2, pp. 97–116.

[28] Vesterstrøm, J.—Thomsen, R.: A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems. In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1980–1987, 2004.

[29] Walshaw, C.: Multilevel Refinement for Combinatorial Optimisation Problems. Annals of Operations Research, Vol. 131, 2004, No. 1–4, pp. 325–372.

Peter Korošec received the B. Sc. and M. Sc. degrees in computer science from the University of Ljubljana, Slovenia, in 2001 and 2004, respectively. In 2006, he received his PhD from the Jožef Stefan International Postgraduate School. Since winter 2002 he has been a researcher at the Jožef Stefan Institute in Ljubljana, Slovenia. His current areas of research include combinatorial and numerical optimization with ant-based metaheuristics.

Jurij Šilc received his Ph. D. degree in electrical engineering from the University of Ljubljana, Slovenia, in 1992. In 1980 he joined the Jožef Stefan Institute, where he is now a senior researcher. At the Institute, he served as the head of the Computer Architecture Laboratory from 1986 to 1994. He is presently the deputy head of the Computer Systems Department and an assistant professor at the Jožef Stefan Postgraduate School. His research interests include processor architecture, parallel computing, combinatorial and numerical optimization.