ELEKTROTEHNIŠKI VESTNIK 79(5): 231–236, 2012
ENGLISH EDITION

Meta-Optimisation on a High-Performance Computing System

Árpád Bűrmen, Tadej Tuma, Iztok Fajfar

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana, Slovenija
E-mail: arpad.buermen@fe.uni-lj.si

Abstract. All optimisation algorithms have parameters that affect their performance and reliability. Usually the default values of these parameters are problem-dependent. Regardless of this fact it is common practice to use some default values that are provided with the optimisation algorithm. Finding the optimal values of these parameters is a computationally expensive optimisation problem also known as meta-optimization. The computational complexity comes from the fact that every cost-function evaluation in meta-optimisation involve several runs of an optimisation algorithm that evaluate its behavior for given values of algorithm parameters. The most common approach to making meta-optimisation feasible is the use of parallel computing. The paper presents the construction of the cost function for meta-optimisation of direct search optimisation algorithms. We demonstrate the approach by optimising the parameters of the Nelder-Mead simplex algorithm using a high-performance computing system comprising 100 processing units. The results of the meta-optimisation are surprising because the obtained values of parameters greatly differ from the values that were published 50 years ago, but are still used despite their suboptimality.

Keywords: Meta-optimisation, global optimisation, simplex algorithm

1 INTRODUCTION

Optimisation is the search for the minimum of a real-valued function of n variables. The function subject to optimisation is also referred to as the cost function (CF). The performance of an optimisation algorithm can be expressed with the number of CF evaluations (N) and the quality of the final result (CF value) returned by the optimisation algorithm. A better optimisation algorithm obtains a lower final CF value with fewer CF evaluations. Its performance directly depends on a set of real-valued algorithmic parameters also referred to as the optimisation algorithm's strategy.

The optimal strategy generally depends on the optimisation problem that is being solved. It is commonplace to find the suggested values of the (default) strategy published together with the algorithm. These values are often obtained with limited numerical trials or some (overly) simplified analysis. Despite this the published strategies are rarely challenged by optimization practitioners. Such careless choice of strategy is often the cause for optimisation algortihms being deemed as inefficient.

The optimal strategy generally depends on the nature of the optimisation problem. The only possible option is to find the optimal strategy before the optimisation itself. The problem of finding the optimal strategy is also referred to as meta-optimisation problem while the process of solving it is called meta-optimisation. The basic (optimisation) algorithm is the one for which the optimal strategy is being sought. Meta-optimisation also involves a CF that expresses the quality of a strategy with a real number. It maps vectors that represent strategies to real numbers that represent the basic algorithm's performance. Lower values of the CF correspond to better strategies. The calculation of teh CF value involves several runs of the basic algorithm that capture its performance on a family of optimisation problems. This makes meta-optimisation a computationally intensive task where even a single CF evaluation can take hours.

The CF in meta-optimisation is often discontinuous, multimodal and is littered with a numerical noise. This usually leaves no other choice, but to use a global optimisation algorithm for finding its minimum. The main disadvantage of these algorithms is the large numer of CF evaluations needed for finding the solution of a meta-optimisation problem (10000 and more). The run times can be significantly shortened by the use of parallel computing which is a viable option because many global optimisation algorithms can be efficiently parallelised.

In this paper we focus on meta-optimisation of local optimisation algorithms. These algorithms search for a local minimum of the CF. Usually, they require a significantly smaller number of CF evaluations when compared to global optimisation algorithms. Although a

Received November 19, 2012
Accepted January 9, 2013

local optimum does not represent the best possible solution of an optimisation problem, optimisation algorithms are still often used in engineering practice. In many engineering optimisation problems evaluation of the CF is computationally expensive while the computing power is limited. Is such cases local optimisation is the only viable choice. A local optimisation algorithm is also adequate if finding a decrease in the CF value is sufficient for deeming the run as successfull.

2 COST FUNCTION IN META-OPTIMISATION

Before any meta-optimisation, one has to choose the family of the optimisation problems for which the optimal basic algorithm strategy is being sought. Real-world optimisation problems are often computationally too expensive and cannot be used for constructing the CF for meta-optimisation. A much better choice are mathematical test functions that are designed for the purpose of evaluating the basic optimisation algorithm. Two sets of test functions are commonly used: the Moré-Garbow-Hillstrom set [1] and the CUTEr set [2]. The former comprises 35 unconstrained optimisation problems, while the latter offers over 1000 unconstrained and constrained problems. A subset of these problems $\{f_1, f_2, ..., f_m\}$ is used for constructing the CF in meta-optimisation. Local optimisation algorithms usually require an initial point. We denote the corresponding initial points with $x_1^0, x_2^0, ..., x_m^0$.

The CF in meta-optimisation measures the performance of a strategy with respect to the performance of some reference strategy (or algorithm). In this e,sense a possible choice is the performance of the basic algorithm using a default strategy. We denote the (reference) number of CF evaluations with $N_1', N_2', ..., N_m'$. The quality of the final result returned by a local optimisation algorithm can be expressed with the norm of the corresponding CF gradient. The lower values of this norm correspond to a better approximation of a local minimum. We denote the (reference) final results and the CF gradient with $x_1', x_2', ..., x_m'$ and $g_1', g_2', ..., g_m'$, respectively.

Denote the final results and the corresponding CF gradient obtained by the basic algorithm with x_1, x_2, ..., x_m and $g_1, g_2, ..., g_m$, respectively. Let N_1, N_2, ..., N_m be the number of CF evaluations in the basic algorithm. It can be expected that many strategies tried during the course of meta-optimisation result in an excessive number of CF evaluations. This results in prolonged meta-optimisation runs. Such situations can be avoided if one places an upper bound ($N_i^{\max}$) on the number of CF (f_i) evaluations. A possible choice would be

$$N_i^{\max} = \max\left(5N_i', 10000\right). \tag{1}$$

The goal of meta-optimisation is to find a strategy for which the basic algorithm's performance exceeds that obtained with the default strategy. This requirement represents a constraint in meta-optimisation and can be handled with a CF that comprises several penalty contributions (C_i). [3] If some strategy behaves worse than the default strategy, the corresponding penalty contribution is positive. On the other hand, if a strategy outperforms the basic strategy, a small negative penalty contribution rewards its behavior. In case a strategy exhibits same performance as the default strategy, the penalty contribution is zero. The CF in meta-optimisation can be expressed as a sum of m penalty contributions representing performance on individual test problems.

$$C = \sum_{i=1}^{m} C_i. \tag{2}$$

$C = 0$ corresponds to a strategy that exhibits the same performance on m test problems as the defautl strategy. The contribution of the i-th test problem to the CF in meta-optimisation can be expressed as

$$C_i = \begin{cases} (N_i/N_i' - 1) \cdot 10; & N_i > N_i' \\ (N_i/N_i' - 1)/10; & N_i \leq N_i' \end{cases} \tag{3}$$

$$+ \begin{cases} \log_{10}(||g_i||/||g_i'||) \cdot 10; & ||g_i|| > ||g_i'|| \\ \log_{10}(||g_i||/||g_i'||)/10; & ||g_i|| \leq ||g_i'|| \end{cases} \tag{4}$$

One can use an arbitrary constant $P > 1$ instead of the fixed value (10) in the above equations. The absolute ratio between a penalty ($C_i > 0$) and reward ($C_i < 0$) attributed to a deterioration/improvement of the same magnitude from the default strategy's performance is P^2. One CF evaluation in meta-optimisation involves m runs of the basic optimisation algorithm.

3 COMPUTER HARDWARE

Meta-optimisation is a computationally intensive procedure. To make it viable one has to take advantage of parallel processing. A sufficient computing power can be obtained by utilizing contemporary high-performance computing (HPC) systems. Such systems comprise a large number of processing units. The role of a process-sign unit can be taken by ordinary desktop computers. Nowadays, the price of a typical desktop computer with a quad-core CPU is around €600 (or €150 per CPU core). Such computers usually come with a built-in network adapter (most commonly $1\mathrm{Gb}$ Ethernet). By adding a network switch (its cost is usually lower than the price of a single desktop computer), one gets a fully functional HPC system. The price of a system comprising 100 CPU cores is around €15000.

The interconnection between the processing units is the main bottleneck. Despite its speed ($1\mathrm{Gbit/s}$), the latency in quite high (around $20\mu\mathrm{s}$). Part of the latency is inherent to Ethernet, while the rest (around $10\mu\mathrm{s}$, [4]) is caused by inefficient drivers. This is quite large

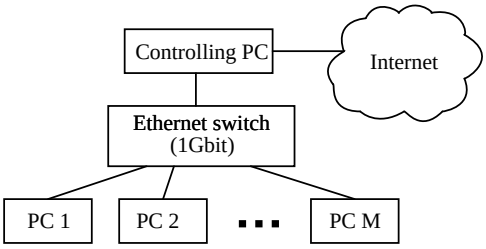

Figure 1. Block diagram of a HPC system based on desktop computers and gigabit Ethernet.

when compared to the latency of the HPC systems using InfiniBand (e.g. [5]) where it can be as low as 1μs. The throughput is also much higher in the InfiniBand systems ($2\mathrm{Gb/s}$ for a basic SDR link). The cost, however, is much greater if the InfiniBand links are used for interconnecting the processing units.

Regardless of the lower throughput and higher latency, a HPC system with the Ethernet interconnection is capable of running parallel optimisation algorithms. Let R denote the throughput of the interconnection while τ denotes the latency. The interconnection does not behave as the system's bottleneck if two conditions are satisfied. The time it takes for a processing unit to complete a task (T) must be significantly higher than the latency. Secondly, the total amount of data D associated with a task (data sent to a processing unit describing the task and data sent from a processing unit with the results of a task) must be small enough so that it can be transferred in the fraction of time it takes for the task to complete. Both conditions can be formulated mathematically as

$$T \gg \tau, \tag{5}$$

$$D/R \ll T. \tag{6}$$

Meta-optimisation along with most engineering optimisation problems satisfies conditions (5)-(6). Condition (5) is satisfied because T is in the $1\mathrm{s}$ order of the magnitude or greater, while τ is in the 10μs order of the magnitude (even for "slow" gigabit Ethernet links). Because every task (CF evaluation) can be specified with n' real numbers (where n' is the dimension of the meta-optimisation problem, i.e. the number of the basic algorithm's parameters). A task produces a result which is a CF value and can be represented as a single real-valued number. Assuming that 64-bit real numbers are used, the amount of data transferred to and from every task is $64\cdot(n'+1)$. This gives us $D/R = (n'+1)\cdot 64\mathrm{ns}$. If completing one task takes $T = 1\mathrm{ms}$, condition (6) is satisfied for all $n' < 15624$.

4 Optimisation software

Parallel optimisation algorithms on HPC systems consist of multiple programs (tasks) running in parallel. Every program runs on a single processor and solves a part of the problem. Programs communicate among themselves and coordinate their work. The communication is usually implemented as messages. When a task receives a message, it processes the message by performing a (lengthy) computation upon which it sends back a message with results. Implementing parallel algorithms can be significantly simplified if one uses a library that wraps the process of sending and receiving messages into a simple application programming interface (API). This hides the details of communication protocols. Currently, two solutions are available: PVM [6] and MPI [7]. PVM is a somewhat older library, while MPI is only an API specification. Several implementations of the MPI specification are available.

Developement of optimisation algorithms requires an environment that 1) simplifies implementation of complex matematical concepts and 2) speeds up debugging of the implemented code. The first requirement is satisfied by several programming laguages in combination with appropriate mathematical libraries. The second requirement is usually fulfilled by various scripting languages. We chose Python [8] in combination with NumPy/SciPy [9] mathematical libraries for implementing all the described algorithms due to its simplicity, generality and the large number of available extensions.

Meta-optimisation can be parallelised using two approaches. The first approach distributes the evaluation of the basic algorithm's performance for one candidate strategy among multiple processors. Every processor runs the basic algorithm on one of the m test problems. When the m optimisation runs are complete, the corresponding value of the CF is computed using (2). The meta-optimisation algorithm then determines the next candidate strategy and distributes its evaluation among parallel processors. This way a speedup factor of up to m can be achieved. Unfortunately the speedup is limited by the fact that not all of the m optimisation runs are finished at the same time. This way some processors remain idle until all of the m processors finish the evaluation (synchronisation penalty). As consequence, the actual speedup factor is smaller than m.

To avoid synchronisation penalty, one can use an asynchronous optimisation algorithm for meta-optimisation. Such algorithms keep all processors busy all the time. The evaluation the meta-optimisation CF is not distributed among processors. This brings an additional advantage by greatly increasing T thus making conditions (5) and (6) satisfied even for the HPC systems with slow interconnections. An example of such algorithm can be found in [10].

5 Example

The Nelder-mead simplex algorithm [11] searches for a local minimum of a function of n variables by moving

$n+1$ points (also referred to as simplex). We assume that the points are ordered so that the largest and smallest function value corresponds to x_{n+1} (worst point) and x_1 (best point), respectively. Most of the time the simplex moves around by replacing the worst point with one of the points lying on a ray with x_{n+1} for origin that runs through the centroid $\overline{x}$ of the n best points ($\overline{x} = (x_1 + ... + x_n)/n$). The value $\gamma \geq -1$ determines a point x on this ray according to

$$x = \overline{x} + \gamma(\overline{x} - x_{n+1}). \tag{7}$$

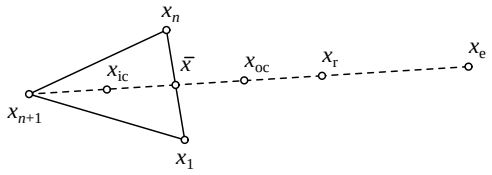

Figure 2. Reflection, expansion, outer contraction, and inner contraction steps in the Nelder-Mead simplex algorithm.

The values of γ denoted by γ_r, γ_e, γ_oc, and γ_ic correspond to points (x_r, x_e, x_oc, and x_ic) that represent the possible replacements for the worst point (see Fig. 2). The corresponding steps are also referred to as reflection, expansion, outer contraction, and inner contraction. If none of these steps improves the worst function value (at x_{n+1}), the simplex is shrinked towards its best point (x_1) according to

$$x_i \rightarrow x_1 + \gamma_\mathrm{s}(x_i - x_1), \;\; i = 2, ..., n+1. \tag{8}$$

γ_s denotes the shrink factor. The values of γ that correspond to the above-mentioned steps must satisfy $0 < \gamma_\mathrm{r} < \gamma_\mathrm{e}$ in $0 < \gamma_\mathrm{oc}, -\gamma_\mathrm{ic}, \gamma_\mathrm{s} < 1$. The recommended values of these parameters found in [11] represent the default strategy

$$[\gamma_\mathrm{r}, \gamma_\mathrm{e}, \gamma_\mathrm{oc}, \gamma_\mathrm{ic}, \gamma_\mathrm{s}] = [1, 2, 1/2, -1/2, 1/2]. \tag{9}$$

The algorithm can be described with the following steps.

1) Order points so that they satisfy $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$.
2) Evaluate $f_\mathrm{r} = f(\vec{x}_\mathrm{r})$.
   If $f_\mathrm{r} < f_1$, evaluate $f_\mathrm{e} = f(x_\mathrm{e})$.
   If $f_\mathrm{e} < f_\mathrm{r}$, replace x_{n+1} with x_e, otherwise replace it with x_r.
3) If $f_1 \leq f_\mathrm{r} < f_n$, replace x_{n+1} with x_r.
4) If $f_n \leq f_\mathrm{r} < f_{n+1}$, evaluate $f_\mathrm{oc} = f(x_\mathrm{oc})$.
   If $f_\mathrm{oc} < f_{\mathrm{n}+1}$, replace x_{n+1} with x_oc,
5) If $f_{n+1} \leq f_\mathrm{r}$, evaluate $f_\mathrm{ic} = f(x_\mathrm{ic})$.
   If $f_\mathrm{ic} < f_{\mathrm{n}+1}$, replace x_{n+1} with x_ic.
6) Steps 2-5 did not replace x_{n+1}, shrink the simplex.
7) If stopping conditions are not satisfied, return to step 1.

The description in [11] does not explicitly state the point ordering in case multiple points share the same value of f. In such case, the points are ordered with the oldest point appearing last.

There are several published examples (e.g. [12]), for which the Nelder-Mead simplex algorithm fails to converge to a local minimum. Despite this fact, it is still very popular due to its simplicity. The goal of meta-optimisation will be to find the optimal strategy for the Nelder-Mead simplex algorithm that is given by its $n' = 5$ parameters.

The set of test functions comprised 39 problems from [13] of which most come from [1]. According to the recommendations in [1] every problem was tested with three different starting points: x^0, $10x^0$, and $100x^0$, where x^0 denotes the problem's standard starting point given in [1] or [13]. An exception to this rule was the Jennrich and Sampson function which is not defined at $100x^0$. The set comprised $m = 3 \cdot 39 - 1 = 116$ test problems. The reference number of the CF evaluations and final gradient norm were obtained with the convergent variant of the algorithm given in [13] by using the standard starting points (x^0). The initial simplex and the stopping conditions were identical to those published in [13].

The following constraints were imposed on the values of the algorithm's parameters (strategy)

$$0.001 \leq \quad \gamma_\mathrm{r} \quad \leq 2, \tag{10}$$
$$0 \leq \quad \gamma_\mathrm{e} - \gamma_\mathrm{r} \quad \leq 4, \tag{11}$$
$$0.001 \leq \quad \gamma_\mathrm{oc}/\gamma_\mathrm{r} \quad \leq 0.999, \tag{12}$$
$$0.001 \leq \quad -\gamma_\mathrm{ic} \quad \leq 0.999, \tag{13}$$
$$0.001 \leq \quad \gamma_\mathrm{s} \quad \leq 0.999. \tag{14}$$

The meta-optimisation was performed on a HPC system comprising $M = 25$ computers with an Intel Core i5 750 processor running at $2,66\mathrm{GHz}$ and one control computer that was responsible for connecting the system to the internet (Fig. 1). The theoretical performance limit is $1\mathrm{Tflops}$ (assuming that every processor core is capable of performing up to four floating point operations per clock cycle). The set of the problems for which such a system can be used is limited by conditions (5)-(6).

The role of the meta-optimisation algorithm was taken by the global optimisation algorithm described in [10]. This algorithm is a combination of simulated annealing and differential evolution. The algorithm was stopped after 200000 CF evaluations. The results were obtained after two days. The following optimal strategy was obtained

$$\gamma_\mathrm{r} = 0.983, \tag{15}$$
$$\gamma_\mathrm{e} = 1.27, \tag{16}$$
$$\gamma_\mathrm{oc} = 0.734, \tag{17}$$

$$\gamma_{\mathrm{ic}} = -0.609, \tag{18}$$

$$\gamma_{\mathrm{s}} = 0.385. \tag{19}$$

Parameters γ_{r} and γ_{ic} are close to the values suggested by Nelder and Mead in [11]. It is interesting to note that the obtained value of $\gamma_{\mathrm{e}} = 1.27$ is close to 1.2 which is the value suggested in [14]. The obtained value of γ_{oc} and γ_{s} differs from the default value (0.5).

To verify the obtained optimal strategy, the basic algorithm was run on the 39 test problems with unscaled initial points x^0. The number of the CF evaluations was limited to 100000. The results are given in Table 1. For two test problems the default strategy used all the available CF evaluations without satisfying the stopping conditions. The optimal strategy satisfied the stopping conditions for all test problems. Cases where one strategy outperforms the other (in terms of the number of CF evaluations and final gradient norm) are doneted by asterisks. The default strategy made fewer CF evaluations than the optimal strategy for 23 test problems, while the opposite was true for 16 test problems. With respect to the final gradient norm, the default strategy outperformed the optimal strategy on 14 problems, while the optimal strategy was better on 25 problems. Considering both CF evaluations and the final gradient norm, the default strategy was better on eight problems, while the optimal strategy outperformed the default one on ten problems.

Columns marked with #r, #e, #oc, and #ic list the relative number of successfull reflection, expansion, outer contraction and inner contraction steps. The columns marked with #s list the number of shrink steps. If the optimal strategy is used, the relative number of successfull reflection steps slightly decreases, while the number of successful expansion steps increases. We assume that this is part of the reason why the optimal strategy outperforms the default strategy. The number of functions on which the optimal strategy outperforms the defautl strategy (density of the asterisk symbols) increases as the dimension of the problem increases. This suggests that the optimal strategy depends on the problem dimension. The number of shrink steps is small and decreases even further if the obtained optimal strategy is used. The outer contraction steps constitute only a small percentage of CF evaluations. The obtained value of γ_{oc} is therefore not reliable. On the other hand, the inner contraction steps represent $10\% - 20\%$ of all the CF evaluations which validates the obtained value of $\gamma_{\mathrm{ic}} = -0.609$.

6 Conclusions

The values of algorithmic parameters (strategy) determine the behavior of a (basic) optimisation algorithm. The process of finding their optimal values can be automated by applying an outer optimisation loop. This outer optimisation is also referred to as meta-optimisation. We gave an example of how to construct the cost function for meta-optimisation. The cost function is based on the basic algorithm's performance measured on several test problems.

Meta-optimisation is a computationally intensive task that usually cannot be finished in a feasible time without the help of high-performance computing systems. We gave a low-cost example of such system with 100 CPU cores. The system is based on standard desktop computers connected with a gigabit Ethernet network. Such systems can be efficient if the time a task takes to complete is significantly longer than the latency of the network and the time it takes for the data to be trasferred to/from the task.

We gave an example of meta-optimisation that searches for the optimal values of the Nelder-Mead simlex algorithm's five algorithmic parameters. The results show that the optimal strategy significantly differs from that published by Nelder and Mead. An interesting fact is that the optimal value of the expansion parameter is almost identical to that suggested in the paper describing the convergent variant of the algorithm based on grid-restrainment [14]. The results also suggest that the optimal strategy depends on the dimension of the optimisation problem.

An interesting task for future research would be to find the dependence of the optimal strategy on the problem dimension and whether that dependence can be explained theoretically.

Acknowledgement

The research has been supported by the Ministry of Education, Science, Culture and Sport of the Republic of Slovenia within the research program P2-0246 - Algorithms and optimisation methods in telecommunications.

References

[1] J.J. Moré, B.S. Garbow, K.E. Hillstrom, "Testing Unconstrained Optimization Software," *ACM Transactions on Mathematical Software*, vol. 7, no. 1, pp. 17–41, 1981.

[2] N.I.M. Gould, D. Orban, P.L. Toint, "CUTEr and SifDec: a constrained and unconstrained testing environment, revisited," *ACM Transactions on Mathematical Software*, vol. 29, no. 4, pp. 373–394, 2003.

[3] A. Bűrmen et al., "Automated robust design and optimization of integrated circuits by means of penalty functions," *AEÜ, International Journal of Electronics and Communications*, vol. 57, no. 1, pp. 47–56, 2003.

[4] "Myrinet Express over Generic Ethernet Hardware," http://open-mx.gforge.inria.fr, april 2012.

[5] "Visokozmogljivi računski sestav HPCFS," http://hpc.fs.uni-lj.si, april 2012.

[6] V.S. Sunderam, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience*, vol. 2, no. 4, pp. 315–339, 1990.

Table 1. Comparison of the default strategy and optimal strategy for the Nelder-Mead simplex algorithm.

		default strategy							optimal strategy						
function	dim.	N	$\|\|g\|\|$	#r	#e	#oc	#ic	#s	N	$\|\|g\|\|$	#r	#e	#oc	#ic	#s
Rosenbrock	2	* 231	4.8e-09	0.13	0.08	0.07	0.25	0.00	366	* 1.4e-09	0.15	0.10	0.06	0.23	0.00
Freudenstein&Roth	2	* 174	*7.1e-07	0.09	0.07	0.05	0.27	3.45	315	1.3e-06	0.16	0.10	0.05	0.20	1.90
Powell	2	* 752	*2.7e-08	0.24	0.11	0.02	0.18	0.00	1073	5.4e-08	0.22	0.16	0.03	0.14	0.00
Brown	2	* 320	*7.0e-02	0.09	0.12	0.06	0.25	0.00	713	1.1e-01	0.18	0.16	0.03	0.16	0.00
Beale	2	* 170	1.7e-09	0.11	0.05	0.08	0.28	0.00	241	* 3.8e-10	0.09	0.07	0.05	0.30	0.00
Jennrich&Sampson	2	* 154	*1.2e-05	0.10	0.03	0.05	0.31	2.60	191	1.8e-05	0.07	0.04	0.06	0.34	1.05
McKinnon	2	* 195	*1.7e-08	0.12	0.10	0.04	0.24	2.05	318	2.4e-08	0.12	0.17	0.02	0.21	1.26
McKinnon (alt)	2	247	1.0e+00	0.00	0.00	0.00	0.49	0.00	* 217	* 3.3e-08	0.11	0.01	0.05	0.34	1.84
Helical Valley	3	* 369	9.6e-09	0.15	0.11	0.04	0.25	0.00	561	* 8.5e-09	0.11	0.16	0.04	0.22	0.00
Bard	3	* 333	4.6e-09	0.14	0.08	0.08	0.24	1.80	439	* 4.0e-09	0.10	0.11	0.05	0.24	2.73
Gaussian	3	* 245	*1.3e-10	0.12	0.06	0.07	0.29	0.00	250	2.9e-10	0.11	0.00	0.05	0.36	0.00
Meyer	3	100002	*2.2e+01	0.22	0.00	0.11	0.00	32.8	* 2525	8.9e+01	0.26	0.17	0.03	0.10	2.61
Gulf R&D	3	3759	*9.8e-15	0.27	0.14	0.03	0.13	0.00	* 3317	1.8e-14	0.25	0.20	0.02	0.11	0.00
Box 3D	3	* 514	4.7e-09	0.13	0.14	0.03	0.23	2.33	656	* 3.2e-11	0.12	0.16	0.04	0.22	0.00
Powell Singular	4	* 1070	6.3e-16	0.27	0.09	0.04	0.19	0.00	1444	* 4.4e-16	0.20	0.14	0.04	0.18	0.00
Wood	4	* 683	5.4e-08	0.27	0.08	0.05	0.20	0.00	945	* 1.2e-08	0.25	0.11	0.05	0.19	0.00
Kowalik&Osborne	4	* 431	9.7e-10	0.24	0.05	0.06	0.21	3.71	551	* 9.1e-10	0.19	0.08	0.07	0.22	2.18
Brown&Dennis	4	* 490	9.8e-04	0.21	0.07	0.04	0.21	5.71	573	* 6.1e-04	0.20	0.08	0.06	0.21	4.19
Quadratic	4	* 359	*5.0e-10	0.17	0.05	0.06	0.28	0.00	510	9.0e-10	0.15	0.07	0.05	0.29	0.00
Penalty I	4	1404	5.4e-11	0.28	0.12	0.04	0.15	1.71	* 1227	* 9.5e-12	0.28	0.14	0.03	0.15	1.30
Penalty II	4	3768	1.5e-10	0.33	0.13	0.03	0.13	0.64	* 2924	* 9.2e-11	0.30	0.17	0.02	0.11	0.55
Osborne 1	5	* 1136	*2.5e-08	0.34	0.07	0.03	0.15	3.08	1277	2.9e-07	0.31	0.11	0.03	0.16	1.96
Brown	5	820	*2.2e-10	0.27	0.08	0.05	0.20	0.00	* 791	1.2e-09	0.23	0.08	0.03	0.25	0.00
Biggs EXP6	6	* 1127	6.6e-08	0.34	0.07	0.04	0.16	3.19	1184	* 1.1e-08	0.31	0.10	0.03	0.17	2.53
Rosenbrock	6	4321	*6.8e-09	0.38	0.11	0.02	0.13	0.00	* 2342	7.0e-09	0.36	0.13	0.02	0.14	0.00
Brown	7	1872	*7.6e-10	0.41	0.07	0.03	0.15	0.00	* 1248	1.2e-09	0.29	0.07	0.04	0.21	0.00
Quadratic	8	1783	1.8e-09	0.46	0.05	0.03	0.14	0.00	* 1166	* 1.7e-09	0.25	0.07	0.04	0.25	0.00
Rosenbrock	8	6181	7.0e-01	0.46	0.09	0.02	0.11	0.65	* 4046	* 1.0e-08	0.41	0.13	0.02	0.12	0.00
Var. Dim.	8	4044	6.6e-09	0.46	0.09	0.02	0.12	0.00	* 2368	* 5.6e-09	0.28	0.15	0.02	0.16	0.00
Powell	8	* 2579	1.3e-04	0.42	0.07	0.03	0.14	0.00	4770	* 6.9e-16	0.40	0.12	0.02	0.12	0.00
Watson	9	2953	*3.3e-08	0.35	0.13	0.03	0.13	1.22	* 2183	7.0e-08	0.23	0.20	0.02	0.14	1.37
Rosenbrock	10	* 6786	3.0e+00	0.51	0.08	0.02	0.10	0.74	11656	* 8.7e-09	0.47	0.14	0.01	0.08	0.00
Penalty I	10	* 5535	4.5e-05	0.50	0.08	0.02	0.11	1.08	6786	* 1.2e-10	0.46	0.13	0.01	0.09	0.88
Penalty II	10	* 6461	1.5e-04	0.47	0.08	0.02	0.13	0.62	9869	* 1.5e-05	0.45	0.15	0.01	0.08	0.61
Trigonometric	10	3196	1.7e-08	0.52	0.06	0.02	0.10	0.94	* 1457	* 3.8e-09	0.37	0.05	0.03	0.19	2.06
Osborne 2	11	4945	6.5e-08	0.55	0.05	0.02	0.09	0.89	* 2870	* 4.8e-08	0.44	0.08	0.02	0.14	1.53
Powell	12	* 7300	7.5e-04	0.55	0.06	0.02	0.10	0.00	12253	* 1.1e-15	0.50	0.12	0.01	0.08	0.00
Quadratic	16	9243	7.5e-09	0.66	0.03	0.01	0.06	0.00	* 3040	* 2.5e-09	0.42	0.05	0.02	0.19	0.00
Quadratic	24	100000	1.4e+00	0.76	0.03	0.01	0.02	0.00	* 6795	* 3.5e-09	0.60	0.04	0.02	0.11	0.00

[7] "MPI: A Message-Passing Interface Standard, Version 2.2, September 4, 2009", http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, april 2012. Version 2.2

[8] "Python Programming Language", http://www.python.org/, april 2012.

[9] "SciPy", http://www.scipy.org/, april 2012.

[10] J. Olenšek, T. Tuma, J. Puhan, Á. Bűrmen, "A new asynchronous parallel global optimization method based on simulated annealing and differential evolution," *Applied Soft Computing Journal*, vol. 11, no. 1, pp. 1481–1489, 2011.

[11] J.A. Nelder, R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[12] K.I.M. McKinnon, "Convergence of the Nelderâ€"Mead simplex method to a non-stationary point,", *SIAM Journal in Optimization*, vol. 9, no. 1, pp. 148–158, 1999.

[13] Á. Bűrmen, T. Tuma, "Unconstrained derivative-free optimization by successive approximation," *Journal of computational and applied mathematics*, vol. 223, no. 1, pp. 62–74, 2009.

[14] Á. Bűrmen, j. Puhan, T. Tuma, "Grid restrained Nelder-Mead algorithm," *Computational optimization and applications*, vol. 72, no. 5, pp. 359–375, 2006.

Árpád Bűrmen received his Ph.D. degree in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia, in 2003. Currently, he is an associate professor at the same Faculty. His research interests include continuous and event-driven simulation of circuits and systems, optimization methods and their convergence theory and applications, and algorithms for parallel and distributed computation. He is one of the principal developers of the SPICE OPUS circuit simulator and has published over 20 papers in peer-reviewed journals.

Tadej Tuma received his B.Sc., M.Sc. and Ph.D. degrees from University of Ljubljana, Faculty of Electrical Engineering, in 1988, 1991, and 1995, respectively. He is a Professor at the same faculty where he teaches four undergraduate and three postgraduate courses. His research interests are mainly in the field of computer-aided circuit analysis and design.

Iztok Fajfar received his B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia in 1991, 1994, and 1997, respectively. In 1991 he was researcher at the Jozef Stefan Institute in Ljubljana. At the end of the same year he was granted a research position at the Faculty of Electrical Engineering, University of Ljubljana. Currently he holds the position of an associate professor at the same faculty. He teaches several introductory and advanced courses in computer programming. He has also participated in several industrial software projects. His research interests include design and optimisation of electronic circuits.