# CONTINUOUS OPTIMIZATION ALGORITHMS: PERFORMANCE ON BENCHMARKING FUNCTIONS AND MODEL PARAMETERS

**Pavel Kordík, Vladimír Bičík**

Department of Computer Science, FIT,
Czech Technical University, Prague, Czech Republic
*kordikp@fit.cvut.cz(Pavel Kordík)*

**Abstract**

Estimation of continuous parameters is frequent task in modelling and simulation. There are several general purpose algorithms available for this task. We benchmarked these algorithms in order to recommend an appropriate algorithm for our model identification problem. We present results of optimization algorithms for standard benchmarking functions and show the importance of proper parameter setting. When these algorithms are applied to the estimation of model parameters, results are quite different. For this task, the gradient (quasi-Newton) and the nature inspired method (CMAES) can be efficiently combined, achieving the best optimization performance.

**Keywords: Continuous Optimization, Neural Network, Modelling, Parameter Estimation**

**Presenting Author's Biography**

Pavel Kordík works as an assistant professor and researcher at the Department of Computer Science, Faculty of Information Technology of the Czech Technical University in Prague, where he obtained his master's and Ph.D. degree in 2003 and 2007, respectively. He is the co-author of more than 40 publications. Recently, he was a coordinator of Automated Knowledge Extraction research project and member of research team of Transdisciplinary Research in the Area of Biomedical Engineering II research programme. His research interests are data mining, knowledge extraction, inductive models, neural networks, evolutionary computing, optimization methods, nature inspired continuous optimization, visualization of black-box behaviour and ensemble techniques.

# 1 Introduction

The optimization in the unconstrained space of continuous parameters is largely different from the combinatorial optimization. The search space is infinite but the gradient or hessian can be used to navigate in this space more efficiently.

However the shape of the search space can often prevent gradient based methods to converge. In this case, nature inspired methods such as Differential Evolution or Particle swarms would be more appropriate to use.

In this paper, we benchmark several continuous optimization methods. The benchmarking functions were selected to provide a wide spectrum of optimization problems.

Parameters of optimization methods were tuned for each benchmarking function, and their performance was significantly improved.

After that, we apply the methods to optimize parameters of data mining models. This is quite challenging task, as soon as the search space differs significantly for data mining models with linear, Gaussian or sigmoidal transfer functions. Also, the training data set influences the shape of the search space.

# 2 Optimization methods

In this section, we shortly describe methods for optimization of continuous parameters that are used in our experiments.

## 2.1 Gradient Based Methods

The *Quasi-Newton method* (QN) [1] is a very popular method of nonlinear continuous optimization. It computes search directions using first-order (gradient) and second-order derivatives (Hessian matrix). To reduce the computational complexity the Hessian matrix is not computed directly, but estimated iteratively using so called updates [2].

The *Conjugate Gradient* method (CG) [3], a non-linear iterative optimization algorithm, is based on an idea that the convergence can be improved by considering also all previous search directions, not only the actual one. Restarting (previous search direction are forgotten) often improves properties of CG method [4]. CG method uses only the first-order derivatives.

## 2.2 Evolutionary Algorithms

*Evolutionary Algorithms* (EAs) are inspired by Darwin's theory of evolution. EAs cover more different approaches, mainly the *Genetic Algorithms* (GAs) [5] and the *Evolution Strategies* (ES) [6] Population of individuals are evolved according to simple rules of evolution. Each individual (phenotype) has a *fitness* assigned. Phenotype is constructed using a genetic information (genotype). Individuals are crossed and mutated by genetic operators while the most fit individuals are selected to survive. After several generations, the mean fitness of individuals stagnates.

Tab. 1 Optimization methods used can be divided into several categories: gradient based, evolutionary, swarm, colony and others.

| Abbrv. | Search | Optimization method |
|---|---|---|
| **QN** | Gradient | Quasi-Newton method |
| **CG** | Gradient | Conjugate Gradient method |
| **DE-pal** | Evol. | Differtial Evolution ver. 1 |
| **DE** | Evol. | Differtial Evolution ver. 2 |
| **SADE** | Evol. | SADE genetic method |
| **CMAES** | Evol. | Covariance Matrix Adapt. ES |
| **PSO** | Swarm | Particle Swarm Optimization |
| **PSOIW** | Swarm | PSO - IW |
| **CACO** | Colony | Cont. Ant Colony Opt. |
| **ACO*** | Colony | Ext. Ant Colony Opt. |
| **DACO** | Colony | Direct ACO |
| **AACA** | Colony | Adaptive Ant Colony Alg. |
| **API** | Colony | API Ant Alg. |
| **HGAPSO** | Hybrid | Hybrid of GA and PSO |
| **SOS** | Other | Stoch. Orthogonal Search |
| **OS** | Other | Orthogonal Search ver. 1 |
| **OS-pal** | Other | Orthogonal Search ver. 2 |
| **RANDOM** | Other | Random search |

The *Differential Evolution* (DE) [7] is a Genetic Algorithm with a special crossover scheme. It adds a weighted difference between two individuals to a third individual. For each individual in the population, an offspring is created using the weighted difference of parent solutions. The offspring replaces the parent in a case it is fitter. Otherwise, the parent survives and is copied to the next generation. The pseudocode describing, how the offspring is created, can be found e.g. in [8]. In our experiments we use two different implementations (DE, DE-pal [9]) of the method.

The *Simplified Atavistic Differential Evolution* (SADE) algorithm [10] is an Evolutionary Algorithm using a real-valued encoding. It works with three genetic operators: the mutation (gaussian noise), the local mutation (final tuning) and the DE crossover scheme mentioned above.

The *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [11] is the state-of-the-art Evolution Strategy where new individuals are sampled according to a continually updated covariance matrix of a multivariate normal mutation distribution.

## 2.3 Swarm Methods

The *Particle Swarm Optimization* method (PSO) uses a swarm of particles to locate an optimum. According to [12] particles "communicate" information they find about each other by updating their velocities in terms of local and global champions; when a new champion is found, the particles will change their positions accordingly so that the new information is "broadcasted" to the swarm. The particles are always drawn back both

to their own personal best positions and also to the best position of the entire swarm. They also have stochastic exploration capability via the use of random constants. Here, we use to modifications of the algorithm: the canonical one (PSO) and the *Particle Swarm Optimization with Stochastic Inertia Weight* (PSO-IW) [13].

### 2.4 Colony Optimization Methods

The *Ant Colony Optimization* (ACO) algorithm is primarly used for discrete problems (e.g. Traveling Salesman Problem, packet routing). However many modifications of the original algorithm for continuous problems have been introduced recently [14]. These algorithms mimic the behavior of real ants and their communication using pheromones. We have so far implemented the following ACO based algorithms:

The *Continuous Ant colony optimization* (CACO) was proposed in [15] and it works as follows. There is an ant nest in a center of a search space with several search vectors leading from the nest. Ant chooses a direction using a roulette wheel selection applied to the pheromone amounts attached to each vector. From the position determined by the selected vector the ant performs a random walk inside a local search radius. This radius can shrink in time to do more detailed search around the point. The quantity of pheromone associated with explored vector is increased proportionally to the quality of the solution. If a better solution is found, the vector is changed to point to actual ant position.

The *Ant Colony Optimization for Continuous Spaces* (ACO*) [16] was designed for the training of feedforward neural networks. However, it is able to solve mixed discrete-continuous optimization problems. In case of continuous variables the pheromone is represented by probability density function (PDF) - a mixture of Gaussian functions. Each time an ant walks through a continuous variable, corresponding PDF is sampled and so the part of a new solution candidate is generated. Other features of ACO like pheromone reinforcement and evaporation are retained.

The *Direct Ant Colony Optimization* (DACO) [17] uses two types of pheromones - one for mean values and one for standard deviations. These values are used by ants to create new solutions and are updated in the ACO way.

The *Adaptive Ant Colony Algorithm* (AACA) [18] encodes solutions into binary strings. Ants travel from the most significant bit to the least significant bit, choosing nodes 0 or 1 for each step. After finishing the trip, the resulting binary string is converted into a solution candidate and the pheromone on the path is reinforced according to the quality of the solution. The probability of a change in more significant bits decreases during algorithm run.

The API algorithm [20] is named after *Pachycondyla apicalis* and it simulates the foraging behaviour of these ants. Ant moves from a nest to a one of several hunting sites, generated in its neighborhood. It performs a random move in the small neigborhood of the selected hunting site to generate a new solution. If an improvement occurs, the next search leads to the same hunting site. If the hunt is unsuccessful more than $p$ times for a single hunting site, the hunting site is forgotten and ant randomly generates a new one. After some time period the nest is moved to the best of all solutions.

### 2.5 Hybrid Search

The *Hybrid of the GA and the PSO* (HGAPSO) algorithm was proposed in [12]. PSO works based on a social adaptation of knowledge, and all individuals are considered to be of the same generation. On the contrary, GA is based on the evolution over successive generations, so the changes of individuals during a single generation are not considered. In nature, individuals will grow up and become more suitable to the environment before producing offspring. To incorporate this phenomenon into GA, PSO is adopted to enhance the top-ranking individuals of each generation.

### 2.6 Other Methods

The *Orthogonal Search* (OS) optimizes a multivariate problem by selecting one dimension at a time, minimizing the error at each step. The OS, also known as the *Powell's method*, was used in [21] to train single layered neural networks.

In our experiments, we use two implementations designated the OS and the OS-pal (see [9]) differing in a type of linesearch method and stopping criteria. The *Stochastic Orthogonal Search* (SOS) differs from OS just by random permutations of dimension order.

As the last method we have employed a simple Random search (RANDOM), where in each iteration the most fit solution out of multiple randomly generated is kept.

In the next section, implemented optimization methods are evaluated in terms of convergence on standard benchmarking functions and their parameters are adjusted for each function.

## 3 Performance on Benchmarking functions

In our experiments, we use a representative set of unimodal and multimodal functions (Table 2). For the detailed description of functions, their visualization and references see [22].

### 3.1 Search space visualization

Most of the benchmarking functions can be easily visualized, as soon as they are two dimensional. Figure 1 shows the example of PSO algorithm optimizing the Rosenbrock's valley function. Particles are marked by circles, the solution by a cross.

### 3.2 Algorithm parameter tuning

For each algorithm, we examined the effect of all parameters on the success rate and number of iterations for each benchmarking function. All results were obtained as an average values for 100 algorithm runs with constant parameter setting and each run was limited to

Tab. 2 Unimodal and multimodal benchmaring functions used in our experiments.

| Abbrv. | Modality | Name of the function |
|---|---|---|
| **BE** | Unimodal | Beale's function |
| **BO** | Unimodal | Booth's function |
| **DJ** | Unimodal | De Jong's (sphere) function |
| **EA** | Unimodal | Easom's function |
| **MA** | Unimodal | Matyas' function |
| **RO** | Unimodal | Rosenbrock's valley |
| **TR** | Unimodal | Trid function |
| **ZA** | Unimodal | Zakharov's function |
| **AC** | Multimodal | Ackley's path |
| **BR** | Multimodal | Branin's function |
| **GP** | Multimodal | Goldstein–Price function |
| **GR** | Multimodal | Griewangk's function |
| **HI** | Multimodal | Himmelblau function |
| **LA** | Multimodal | Langerman's function |
| **L3** | Multimodal | Levy function no. 3 |
| **L5** | Multimodal | Levy function no. 5 |
| **MI** | Multimodal | Michalewicz's function |
| **RN** | Multimodal | Rana's function |
| **RA** | Multimodal | Rastrigin's function |
| **SH** | Multimodal | Shekel's foxholes |
| **SB** | Multimodal | Shubert's function |
| **SW** | Multimodal | Schwefel's function |
| **WH** | Multimodal | Whitley's function |

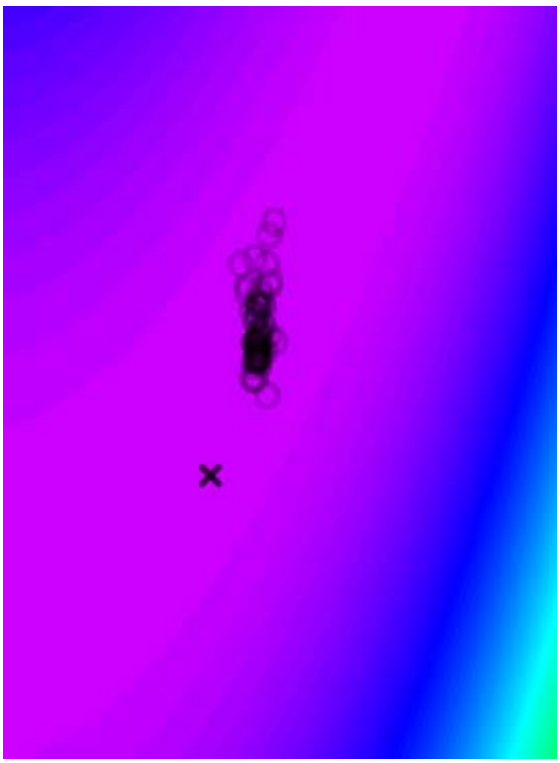

Fig. 1 PSO: particles inside a valley of the Rosenbrock benchmarking function.

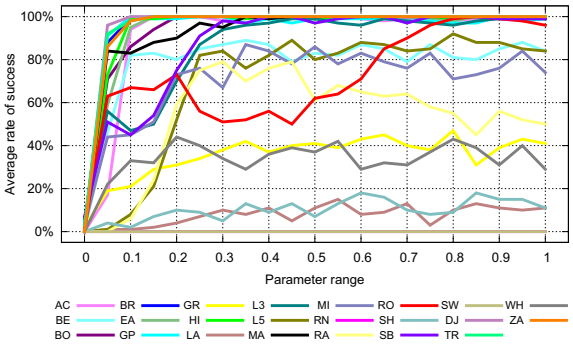


Fig. 2 Average success rate for PSO–original (social acceleration coefficient $\phi_2$ = 0.00, ..., 1.00, step = 0.05).

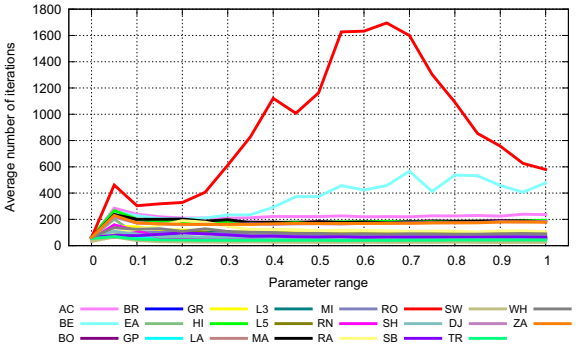


Fig. 3 Average number of iterations taken by PSO–original ($\phi_2$ = 0.00, ..., 1.00, step = 0.05).

2000 iterations.

For the stopping condition adds its limit to the iteration counter, we subtracted this number from the observed data if the number of iterations was lower than the overall limit of iterations taken. By doing this we get better idea about the speed of convergence for tested algorithms as the number used in stopping criterion is very high for numerical methods but proved to be useful for the methods inspired by nature. As an example, we show the sensitivity of the original PSO algorithm to the change of its *social acceleration* parameter $\phi_2$. The other parameters *cognitive acceleration coefficient* $\phi_1$ and *population size* were constant. The higher this parameter was set, the better results we observed (Figure 2).

We also recorded two exceptions, the Rosenbrock's valley and the Rastrigin's function. For the Rosenbrock's valley the success rate was decreasing up to the value of $\phi_2 = 0.5$ and then followed the rising trend. This might be caused by having $\phi_2 < \phi_1$ means the particles are staying in the valley disregarding whether it is the correct side; the valley is curved. But as the parameter starts to outweight the local attraction rate, particles are more easily shifted to the lower areas of the valley and keep "sliding" towards the global optimum as the particle with the lowest value leads the way.

The latter function is highly multimodal and setting the

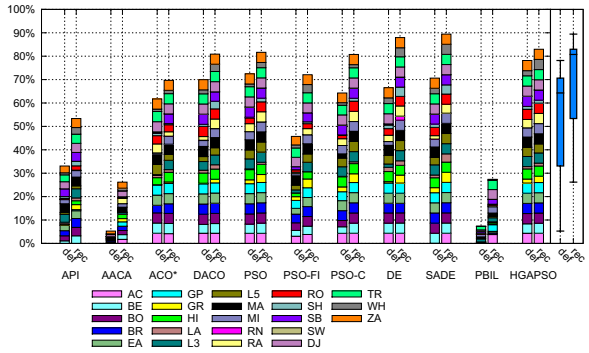


Fig. 4 Comparison of the average success rate for default parameter setting and for recommended setting.

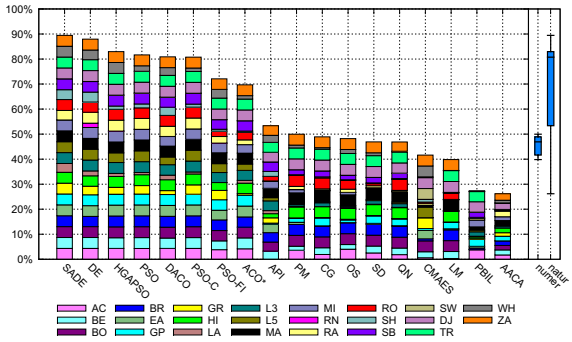


Fig. 5 Average success rate for implemented methods

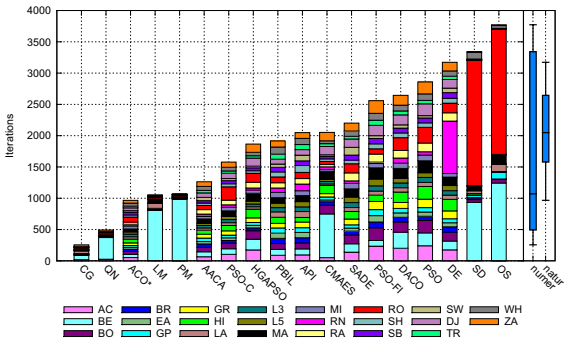


Fig. 6 Average number of iterations taken by implemented methods.

*global attraction* to a high value leads to a premature convergence into a shared global candidate which is unfortunately one of the many local minima. This is exactly the opposite case than with the previous parameter as the decrease is clearly visible for values when $\phi_2 > \phi_1$, meaning that this parameter outweighs the *local attraction* parameter. Therefore the particles are more easily dragged into a local minimum.

Considering the iteration count we can notice that again the "valley" functions reflect change of $\phi_2$ significantly: the Beale's function and the Rosenbrock's valley (Figure 3). For these the iteration count taken by PSO increases to a certain point and then decreases, both times with the same rate. Global optimum of Beale's function is hidden in a hard to find valley, that might be the cause of the particles missing it when strongly attracted to the global optimum. However the Rosenbrock's valley is easy to find and the iterations taken by PSO on this function correspond to the success rate: as soon as the success rate increases the iteration count decreases. This follows the idea of particles forming a kind of a snake or a queue with head of this formation having the lowest value and speeding it towards the solution as illustrated by Figure 1.

In [22], you can find the the sensitivity of all algorithms to all their parameters.

Based on the sensitivity evaluation, we have changed the default setting of parameters to a *recommended setting* (the same for all benchmarking functions).

Two columns if Figure 4 correspond to each algorithm. The one on the left is for the default parameter setting (labeled "def"), the one on the right represents results obtained for the setting recommended in this chapter (labeled "rec"). Different recommended settings were used for different function types if appropriate. Column height represents summation of success rates as recorded for each benchmark function, these are represented by individual color boxes. The ideal algorithm would reach 100% which would mean a 100% success rate for each test function. On the far right there are two boxplots, the left one for average success rates with default parameter setting, the right one for average success rates with recommended setting. We can clearly see that there was a significant improvement.

Average increase in the success rate was 16.09%, the lowest 4.78% for the HGAPSO algorithm, which was already giving very good results, and the highest was 26.35% for PSO with FI velocity update formula.

### 3.3 Overall comparison

The numerical methods rank poorly in terms of average success rate because of multimodal functions (Figure 5). The iteration count of numerical methods is lower (Figure 6), also thanks to their premature convergence to local minima on multimodal functions. The average convergence of numerical methods is also worsen by poor results on some specific misleading functions, such as the GP function with multiple local optima near the global optimum. In our comparison, nature inspired methods dominated. The SADE method was the most accurate in average and the ACO* method [19] had the best accuracy/iterations ratio. However, with better stopping conditions and multiple restarts, numerical methods might score better.

## 4 Optimization of model parameters

In the previous section, we show that, not surprisingly, the performance of optimization methods depends strongly on the shape of function surface. If we would like to optimize parameters of data mining models, we cannot simply choose the best performing method SADE and apply it to this problem.

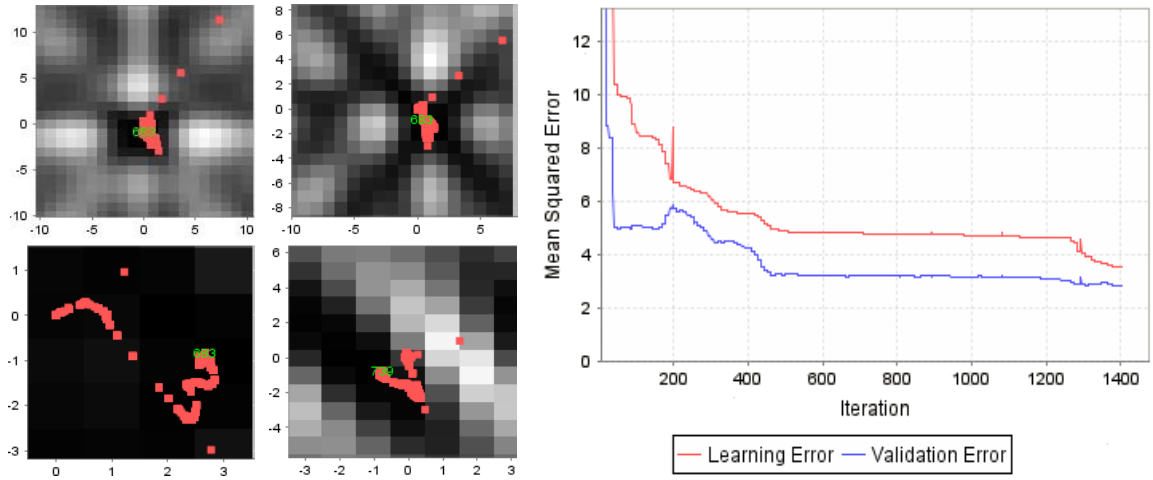


Fig. 7 The Sine neuron optimized by Quasi-Newton algorithm with numerical estimates of the analytic gradient. Visualization of the training error surface from different perspectives (left) and the convergence history (right).

First we visualize the shape of "error" surface. Each parameter of a data mining model to be optimized is one dimension of the error surface. By adjusting parameters, the error of the model on training data can be minimized.

### 4.1 Search space visualization

The training error is dependent variable and model parameters are independent variables. To be able to see, how the training error surface looks like, we need to use projections of the multidimensional space. We use a scatterplot matrix of training error plots. For each plot, two parameters are varied in the interval $(-15, 15)$, all other parameters are fixed (values in actual iteration) and the training error is computed in each point of the plot. The darker the background is, the lower the training error. We visualize also the iteration history in each plot and observe, how the process converges.

This visualization is particulary useful for exploring the training error surface complexity in each dimension (how a change of individual parameter influences the error).

In the Figure 7, you can observe the training log of the single Sine neuron on the Bosthouse data set [23]. The Quasi-Newton method (QN) needed almost 1400 iteration to find optimal values of 24 parameters (Bosthouse has 12 features, the output of the model is $y = \sum_{i=1}^{12} \sin(a_i * x_i + a_{12+i})$). The algorithm managed to escape from a local minima (iteration 170) and converged to a global optima.

For each model, and also for each data set, the shape of the error surface is different. The most significant is the type of transfer function. Also, the surface gets more complex for a model combined from several models.

### 4.2 Performance on individual data sets

The combination of models (or neurons) into one model increases the complexity of the error surface and makes the optimization task harder. Our GAME modeling strategy [24], generates the final model using elementary models such as the Sine model examined above. The elementary models are optimized independently, one by one. For classification tasks, we generate one model for each category and the predicted class is given by the model with a highest response.

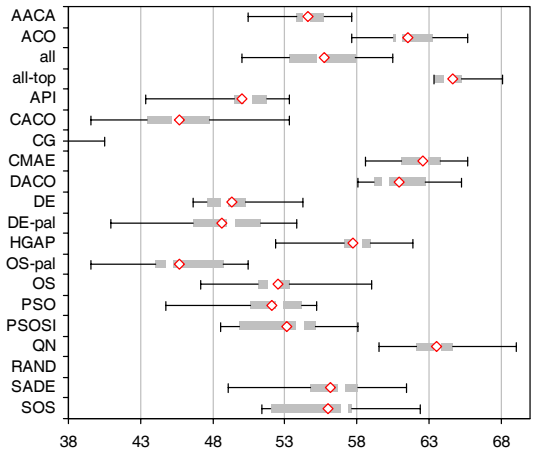


Fig. 8 Box plots showing classification accuracy of models optimized by individual algorithms on the Glass data sets.

Next experiment was designed to find the most suitable algorithm for optimizing parameters of models. We have evaluated the performance of individual algorithms on two different data sets. The Glass data set was obtained from the UCI repository and the Spiral data set is described in [25].

For the Glass data set (Figure 8), the best algorithm was the QN followed by CMA-ES and DACO. On the Two intertwined spirals problem (Figure 9) results were completely different. The only optimization algorithms capable of training successful networks were

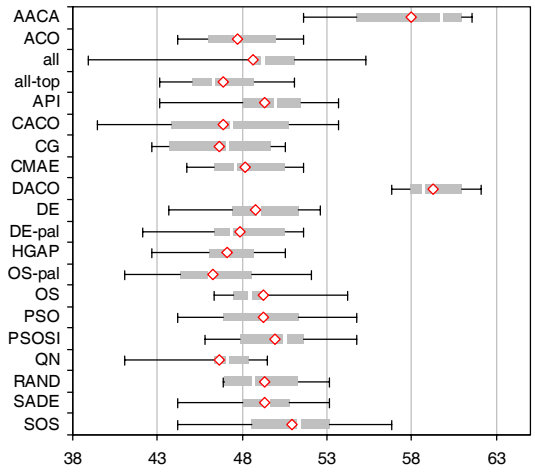


Fig. 9 The classification accuracy of models optimized by individual algorithms on the Spiral data sets.

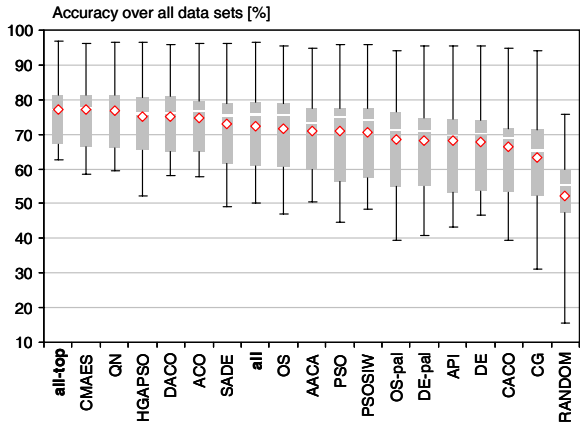


Fig. 10 The mean accuracy of methods over all data sets

Ant colony based DACO and AACA. All other algorithms failed to produce useful classifiers.

There are more methods employed in the evaluation than in previous benchmarks. The *DE-pal* and *OS-pal* are reference implementations of Differential evolution and Orthogonal search methods from the PAL library [9]. When their results are compared with the results of the same algorithms implemented in our environment, you can see, that good implementation and well tuned parameters improve accuracy significantly.

The *all* algorithm pseudo-randomly chooses the optimization algorithm for each elementary model. The *all-top* algorithm selects just from two algorithms the QN and CMA-ES and the final classifier is therefore optimized by both of them.

### 4.3 Averaged performance

Figure 10 averages the results over following group of data sets: Cancer, Diabetes, Gene, Glass, Heart, Heartc, Horse. The final accuracy was computed as average from all results of ten fold cross validation over all data sets (70 numbers averaged for each method).

For detailed description of data sets and the methodology of experiments, you can refer to [26].

The differences among the best three methods *all-top*, *CMA-ES* and *QN* are not significant. The QN requires significantly lower number of iterations and we use it as the default optimization method in our FAKE GAME open source software [27].

## 5 Conclusion

In this paper, we present results of several algorithms for optimization of continuous parameters. Our results on benchmarking functions indicate, that nature inspired methods are more accurate for multimodal functions. Numerical methods, on the other hand, converge faster with much lower number of function evaluations, when properly configured.

Appropriate configuration of algorithms is very important and we show that it can significantly improve their convergence. We plan to experiment with meta-learning methods in order to configure algorithms for given problem described by meta-data.

Estimation of parameters for data mining models is quite different problem. Visual inspection tools allowed us to explore the multi-dimensional search space. The transfer function of the model is the dominant factor influencing the complexity of the search space. Also the data set is important.

For simple problems from the UCI repository, the QN method seems to be a reasonable choice for the data mining model optimization. For certain harder problems such as fitting Sine models to the Spirals problem, only ACO based methods converge.

We have implemented all algorithms and benchmarks into the open source Java COntinuous Optimization Library (JCOOL). Currently, we plan to extend JCOOL with an algorithm recommendation system based on a knowledge base and meta-data.

## 6 Acknowledgement

Thanks to our colleagues from the Computational Intelligence Research Group that help to develop the Java COntinuous Optimization Library (JCOOL). Special thanks goes to Oleg Kovářík for implementing Ant Colony methods for continuous domain and Martin Hvizdoš to design and implement the architecture of the JCOOL.

This research was partially supported by the grant Automated Knowledge Extraction (KJB201210701) of the Grant Agency of the Academy of Science of the Czech Republic and the research program "Transdisciplinary Research in the Area of Biomedical Engineering II" (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic.

## 7 References


[1] R.B. Schnabel, J.E. Koontz, and B.E. Weiss. A modular system of algorithms for unconstrained minimization. Technical Report CU-CS-240-82, Comp. Sci. Dept., University of Colorado at Boulder, 1982.

[2] Salane and Tewarson. A unified derivation of symmetric quasi-newton update formulas. *Applied Math*, 25:29–36, 1980.

[3] J. G. Wade. Convergence properties of the conjugate gradient method. available at www-math.bgsu.edu/ gwade/tex_examples/example2.txt, September 2006.

[4] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213, August 1994.

[5] J. Holland. *Adaptation in Neural and Artificial Systems*. University of Michigan Press, 1975.

[6] I. Rechenberg. *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.

[7] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[8] J. Vesterstrom and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation*, volume 2, pages 1980–1987, 2004.

[9] Alexei Drummond and Korbinian Strimmer. Pal: an object-oriented programming library for molecular evolution and phylogenetics. *Bioinformatics*, 17(7):662–663, 2001.

[10] Ondřej Hrstka and Anna Kučerová. Improvements of real coded genetic algorithms based on differential operators preventing premature convergence. *Advances in Engineering Software*, 35(3-4):237–246, March-April 2004.

[11] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[12] Chia-Feng Juang and Yuan-Chang Liou. On the hybrid of genetic algorithm and particle swarm optimization for evolving recurrent neural network. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 3, pages 2285–2289, Dept. of Electr. Eng., Nat. Chung-Hsing Univ., Taichung, Taiwan, 25-29 July 2004.

[13] Marco Antonio Montes de Oca, Thomas Stützle, Mauro Birattari, and Marco Dorigo. A comparison of particle swarm optimization algorithms based on run-length distributions. In *ANTS Workshop*, pages 1–12, 2006.

[14] S. Tsutsui, M. Pelikan, and A Ghosh. Performance of aggregation pheromone system on unimodal and multimodal problems. In *The IEEE Congress on Evolutionary Computation, 2005 (CEC2005)*, volume 1, pages 880–887. IEEE, 2-5 September 2005.

[15] George Bilchev and Ian C. Parmee. The ant colony metaphor for searching continuous design spaces. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 25–39, London, UK, 1995. Springer-Verlag.

[16] Christian Blum and Krzysztof Socha. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In *Proceedings of Hybrid Intelligent Systems Conference, HIS-2005*, pages 233–238, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[17] M. Kong and P. Tian. A direct application of ant colony optimization to function optimization problem in continuous domain. In *In Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7*, 2006.

[18] Y. Li and T. Wu. An adaptive ant colony system algorithm for continuous-space optimization problems. *J Zhejiang Univ Sci*, 4(1):406, 2003.

[19] O. Kovářík and P. Kordík. Optimizing Models Using Continuous Ant Algorithms. In *Proceedings of the 2nd International Conference on Inductive Modelling*, volume 1, pages 124–128, Kiev, 2008. Ukr. INTEI.

[20] N. Monmarché, G. Venturini, and M. Slimane. On how pachycondyla apicalis ants suggest a new search algorithm. *Future Gener. Comput. Syst.*, 16(9):937946, 2000.

[21] K.M. Adeney and M.J. Korenberg. An easily calculated bound on condition for orthogonal algorithms. In *IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, volume 3, page 3620, 2000.

[22] Vladimír Bičík. Continuous optimization algorithms. Master's thesis, Czech Technical University in Prague, 2010.

[23] P. Kordík. *Fully Automated Knowledge Extraction using Group of Adaptive Models Evolution*. PhD thesis, Czech Technical University in Prague, Praha, 2006.

[24] P. Kordík. *GAME - Hybrid Self-Organizing Modeling System Based on GMDH*, volume 1, pages 233–280. Springer, Berlin, 2009.

[25] Hugues Juille and Jordan B. Pollack. Co-evolving intertwined spirals. In *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, Evolutionary Programming V, pages 461–467. MIT Press, 1996.

[26] P. Kordík, J. Koutník, J. Drchal, O. Kovářík, M. Čepek, and M. Šnorek. Meta-learning approach to neural network optimization. *Neural Networks*, 2010 (23)(4):568–582, May 2010.

[27] GAME. The fake game environment for the automatic knowledge extraction. available online at http://sourceforge.net/projects/fakegame/.