OPTIMIZATION OF MODELS: LOOKING FOR THE BEST STRATEGY

Pavel Kordík, Oleg Kovářík, Miroslav Šnorek

Department of Computer Science and Engineering, FEE,
Czech Technical University, Prague, Czech Republic
kordikp@fel.cvut.cz (Pavel Kordík)

Abstract

When parameters of model are being adjusted, model is learning to mimic the behaviour of a real world system. Optimization methods are responsible for parameters adjustment. The problem is that each real world system is different and its model should be of different complexity. It is almost impossible to decide which optimization method will perform the best (optimally adjust parameters of the model). In this paper we compare the performance of several methods for nonlinear parameters optimization. The gradient based methods such as Quasi-Newton or Conjugate Gradient are compared to several nature inspired methods. We designed an evolutionary algorithm selecting the best optimization methods for models of various complexity. Our experiments proved that the evolution of optimization methods for particular problems is very promising approach.

Keywords: Continuous Optimization, Ant Colony Optimization, Particle Swarm Optimization, Genetic Algorithm, Quasi-Newton Method

Presenting Author's Biography

Pavel Kordík works as an assistant professor and researcher at the Department of Computer Science and Engineering, FEE, 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 20 publications. He is 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 question "Which optimization method is the best for our problem?" has not a simple answer. There is no method superior to others for all possible optimization problems. However there are popular methods performing well on whole range of problems.

Among these popular methods, we can include so called gradient methods - the Quasi Newton method, the Conjugate Gradient method and the Levenberg-Marquardt method. They use an analytical gradient (or its estimation) of the problem error surface. The gradient brings them faster convergence, but in cases when the error surface is jaggy, they are likely to get stuck in a local optima.

Other popular optimization methods are genetic algorithms. They search the error surface by jumping on it with several individuals. Such search is usually slower, but more prone to get stuck in a local minima. The Differential Evolution (DE) perform genetic search with an improved crossover scheme.

The search performed by swarm methods can be imagined as a swarm of birds flying over the error surface, looking for food in deep valleys. You can also imagine that for certain types of terrain, they might miss the deepest valley. Typical examples of swarm methods are Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) that mimics the behavior of real ants and their communication using pheromone.

Optimization methods with different behavior are often combined in one algorithm such as Hybrid of the Genetic Algorithm and the Particle Swarm Optimization (HGAPSO).

We use optimization methods to adjust parameters of inductive models. An inductive model can be created from a data set using some algorithm - in our case the Group of Adaptive Models Evolution (GAME) algorithm [1]. An example of inductive model created by GAME algorithm is depicted on the Figure 1. Similarly to Multi-Layered Perceptron (MLP) neural networks, GAME units (neurons) are connected in a feedforward network (model). The structure of the model is evolved by special niching genetic algorithm, layer by layer. Parameters of the model (coefficients of units' transfer functions) are optimized independently. The problem is to decide which optimization method should be used.

Each data set have different complexity. The surface of a model's RMS error depends on the data set, transfer functions of optimized unit and also on preceding units in the network. Therefore we might expect, there is no universal optimization method performing optimally on all data sets.

In the next section you can find the short description of optimization methods used. Then we compare their performance and discuss the question how the best method for certain data set can be found. We also experiment with evolution of optimization methods (see Section 4). Finally we summarize results over several data sets and recommend the best optimization strategy.

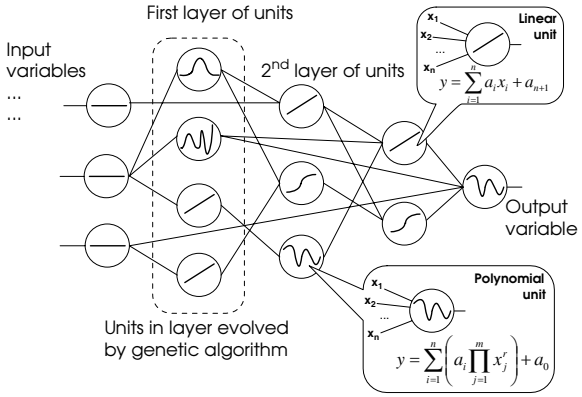

Fig. 1 The structure of the GAME model is evolved layer by layer using special genetic algorithm. During the training of a unit, selected optimization method adjusts coefficients $(a_1, ..., a_n)$ in the transfer function of the unit.

Tab. 1 Optimization methods summary

Abbrv.	Search	Optimization method
QN	Gradient	Quasi-Newton method
CG	Gradient	Conjugate Gradient method
PalDE	Genetic	Differtial Evolution ver. 1
DE	Genetic	Differtial Evolution ver. 2
SADE	Genetic	SADE genetic method
PSO	Swarm	Particle Swarm Optimization
CACO	Swarm	Cont. Ant Colony Opt.
ACO*	Swarm	Ext. Ant Colony Opt.
DACO	Swarm	Direct ACO
AACA	Swarm	Adaptive Ant Colony Opt.
API	Swarm	ACO with API heur.
HGAPSO	Hybrid	Hybrid of GA and PSO
SOS	Other	Stoch. Orthogonal Search
OS	Other	Orthogonal Search

2 Optimization methods

In this paper, we use optimization methods, that are implemented in the GAME engine [1] (see Table 1).

2.1 Gradient based methods

The most popular optimization method of nonlinear programming is the Quasi-Newton method (QN) [2]. It computes search directions using gradients of an energy surface. To reduce their computational complexity, second derivatives (Hessian matrix) are not computed directly, but estimated iteratively using so called updates [3].

The Conjugate gradient method (CG) [4], a non-linear iterative method, is based on the 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 [5].

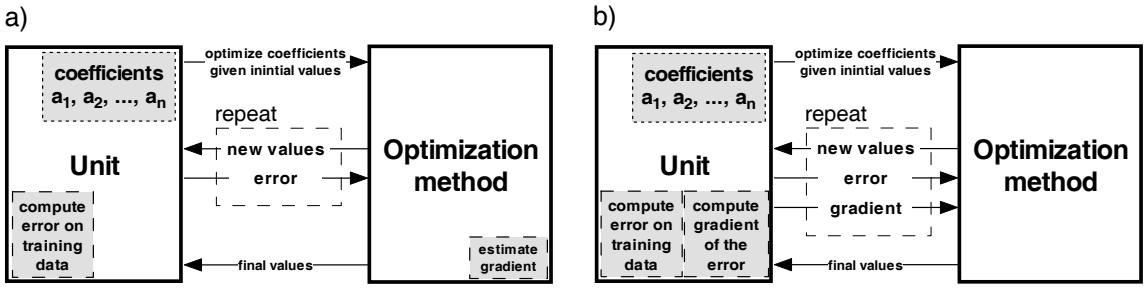

Fig. 2 There are two optimization modes available in the GAME engine. First mode (Figure a) is less effective and it is used for units which do not supply analytic gradient of their error. In this mode, gradient methods (QN, CG) have to compute an estimate of the gradient. In the second (Figure b) mode the analytic gradient supplied by the unit reduces the number of optimization steps required.

2.2 Genetic search

Genetic Algorithms (GA) [6] are inspired by Darwin's theory of evolution. Population of individuals are evolved according simple rules of evolution. Each individual has a *fitness* that is computed from its genetic information. Individuals are crossed and mutated by genetic operators and the most fit individuals are selected to survive. After several generations the mean fitness of individuals is maximized.

Niching methods [7] extend genetic algorithms to domains that require location of multiple solutions. They promote the formation and maintenance of stable subpopulations in genetic algorithms (GAs). The GAME engine uses the *Deterministic Crowding* (DC) [8] niching method to evolve structure of models. There exist several other niching strategies such as fitness sharing, islands, restrictive competition, semantic niching, etc.

The Differential Evolution (DE) [9] is a genetic algorithm with special crossover scheme. It adds the 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 case it is fitter. Otherwise, the parent survives and is copied to the next generation. The pseudocode, how offsprings are created, can be found e.g. in [10].

The Simplified Atavistic Differential Evolution (SADE) algorithm [11] is a genetic algorithm improved by one crossover operator taken from differential evolution. It also prevents premature convergence by using so called radiation fields. These fields have increased probability of mutation and they are placed to local minima of the energy function. When individuals reach a radiation field, they are very likely to be strongly mutated. At the same time, the diameter of the radiation field is decreased. The global minimum of the energy is found when the diameter of some radiation field descend to zero.

2.3 Swarm methods

The Particle Swarm Optimization method (PSO) use a swarm of particles to locate the optimum. According to [19] particles "communicate" information they find about each other by updating their velocities in terms of local and global bests; when a new best is found, the particles will change their positions accordingly so that the new information is "broadcast" 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 the random constants.

The Ant colony optimization (ACO) algorithm is primary 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 [12]. These algorithms mimic the behavior of real ants and their communication using pheromone. We have so far implemented the following ACO based algorithms:

The Continuous Ant colony optimization (CACO) was proposed in [13] and it works as follows. There is an ant nest in a center of a search space. Ants exits the nest in a direction given by quantity of pheromone. When an ant reaches the position of the best ant in the direction, it moves randomly (the step is limited by decreasing diameter of search. If the ant find better solution, it increases the quantity of pheromone in the direction of search [14].

The Ant Colony Optimization for Continuous Spaces (ACO*) [15] was designed for the training of feed forward neural networks. Each ant represents a point in the search space. The position of new ants is computed from the distribution of existing ants in the state space.

Direct Ant Colony Optimization (DACO) [16] uses two types of pheromone - one for mean values and one for standard deviation. These values are used by ants to create new solutions and are updated in the ACO way.

The Adaptive Ant Colony Algorithm (AACA) [17] encodes solutions into binary strings. Ants travel from least significant bit to the most significant bit and back. After finishing the trip, the binary string is converted to the solution candidate. The probability of change decreases with significance of bit position by boosting pheromone deposits.

The API algorithm [18] is named after Pachycondyla

apicalis and it simulates the foraging behaviour of these ants. Ants move from nest to its neighborhood and randomly explore the terrain close to their hunting sites. If an improvement occurs, next search leads to the same hunting site. If the hunt is unsuccessful for more than p times for one hunting site, the hunting site is forgotten and ant randomly generates a new one.

2.4 Hybrid search

The Hybrid of the GA and the PSO (HGAPSO) algorithm was proposed in [19]. PSO works based on social adaptation of knowledge, and all individuals are considered to be of the same generation. On the contrary, GA works based on evolution from generation to generation, so the changes of individuals in 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 on each generation.

2.5 Other methods

The Orthogonal Search (OS) optimizes multivariate problem by selecting one dimension at a time, minimizing the error at each step. The OS can be used [20] to train single layered neural networks.

We use minimization of a real-valued function of several variables without using gradient, optimizing variables one by one. The Stochastic Orthogonal Search (SOS) differs from OS just by random selection of variables.

3 Which optimization method is the best?

The goal of optimization methods is to find optimal values of coefficients $a_1, a_2, ..., a_n$ in transfer functions of GAME units (see Figure 2).

The complexity of this task is dependent on many factors. First of all, the transfer function of actual unit can be of different type and complexity. Then, all preceding units contribute to the complexity of optimization task. The most significant factor is probably the character of data set.

For this reason we assume that even within single model, different optimization methods might be appropriate to adjust coefficients of units. We designed experiments to find out, if a combination of different optimization methods within single model is better than using just one optimization method. Later in this paper (see Section 4), we present technique which selects appropriate method for each unit automatically, by means of evolutionary algorithm.

To find out the best optimization method for individual data sets, we performed following experiments.

3.1 Comparison of individual optimization methods on different data sets

Several different real world data set were involved in the comparison. The detailed description of these data sets can be found in [1]. For each data set, we generated inductive models using the GAME algorithm. Coefficients of all units in models were optimized by a single optimization method from the Table 1. In the configuration *All*, we combined optimization methods using the technique presented in the Section 4. Because these experiments were computationally expensive (optimization methods not utilizing the analytic gradient need many more iterations to converge), we repeated the experiment 5 times for each configuration in case of Building data set and 20 times (20 models) for each configuration for Boston and Spiral data sets.

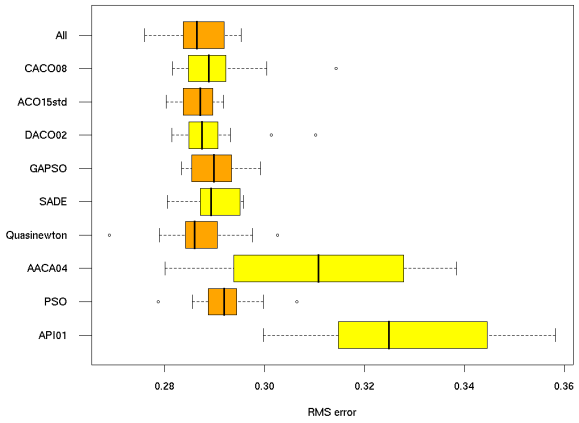

Fig. 3 The RMS error of models on the Boston data set. Units of models were optimized by individual methods.

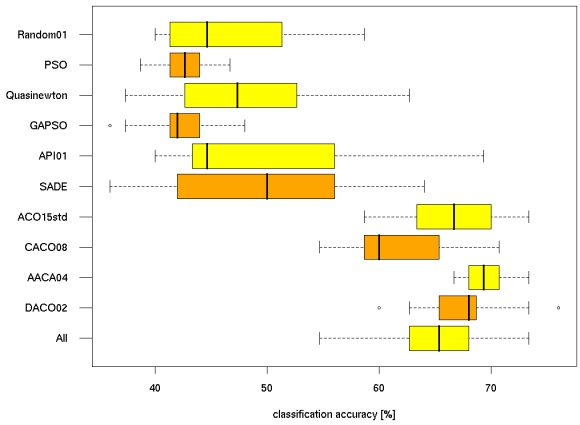

Fig. 4 The classification accuracy of models optimized by individual methods on two intertwined spirals problem.

The results on Boston data set (Figure 3) show that almost all methods demonstrated equal performance. Just two Ant Colony derivatives AACA and API performed significantly worse.

On the Two intertwined spirals problem (Figure 4) results are completely different. Ant colony methods showed better classification accuracy than other methods.

The average root mean squared errors of individual methods on the Building data set for it's three output variables are shown in the Figure 5. There is no signif-

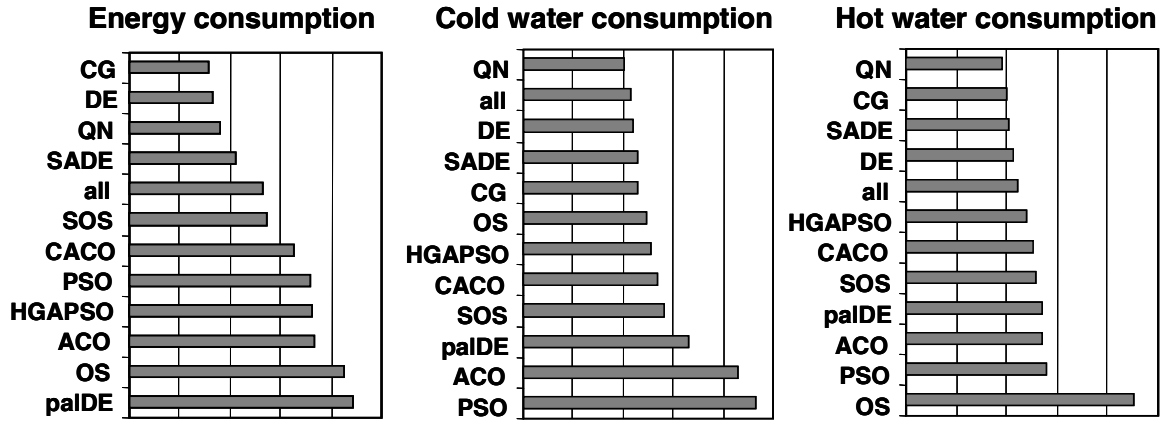

Fig. 5 The performance comparison of optimization methods on the Building data set. The size of bars for individual methods is proportional to the average testing RMS error of models generated using these methods on the Building data set. Models were generated individually for each output variable.

icant difference between results for the noisy variable (Energy consumption) and the other two. We can divide optimization methods into the good and bad performing classes. Good performers are Conjugate Gradient, Quasi Newton, SADE genetic algorithm, Differential Evolution, and the *all* configuration standing for all methods participation in models evolution. On the other hand badly performing optimization methods for the Building data set are Particle Swarm Optimization, PAL- Differential Evolution and the Ant Colony Optimization. PalDE is the second version of the Differential Evolution algorithm implemented in the GAME engine. The result when the first version of DE performed well and the second version badly is pellicular. It signifies that the implementation and the proper configuration of a method is of crucial importance. In accordance with results published in [10], our version of differential evolution outperformed swarm optimization methods on this data set.

4 Evolution of optimization methods

In this section we explain the *All* configuration, that can be found in Figures in previous Section. We assumed that for each data set, some optimization methods are more efficient than others. If we select appropriate method to optimize coefficients of each unit within single GAME network, the accuracy will increase. The problem is to find out which method is appropriate (and most effective).

In the "All" configuration, we used simple strategy. When a new unit was generated, random method was assigned to optimize the coefficients of units. In case the optimization method was inappropriate, coefficients were not set optimally and unit did not survived in the genetic algorithm evolving units in the layer of the GAME model. Only appropriate optimization methods were able to generate fittest units.

The question is if it is better to assign optimization method randomly or inherit it from parent units.

The type of optimization method can be easily inherited from parents, because units are evolved by means of niching genetic algorithm. This genetic algorithm can also assign appropriate optimization methods to units being evolved. We added the type of the optimization into the chromosome (see Figure 6). When new units are generated by crossover to the next generation, they also inherit type of optimization from their parent units. The result should be that methods, training successful units, ale selected more often than methods, training poor performers on a particular data set.

Again, an experiment was designed to prove this assumption.

4.1 Inheritance of methods

We prepared configurations of the GAME engine with several different inheritance settings. In the configuration $p0\%$ new units inherit their optimization method from their paren units. In the configuration $p50\%$ offsprings have 50% chance to get random method assigned. In the configuration $p100\%$ nothing is inherited, all optimization methods are set randomly.

We have been experimenting with the Mandarin, Antro and Boston data sets. For each configuration 30 models were evolved. The maximum, minimum and mean of their RMS errors for each configuration are displayed in the Figure 7. Results are very similar for all configurations and data sets. There is no configuration significantly better than others. For all data sets we can observe that the $p50\%$ and the $p100\%$ configuration have slightly better mean error values and lower dispersion of errors. We chose the $p50\%$ configuration to be default in the GAME engine. It means offspring units have 50% chance to get random optimization method assigned otherwise their methods are inherited from parent units.

Finally, we would like to answer the question "Which optimization method is the best?" To be able to do it, we performed hundreds of experiments on several data sets. Next section presents our results.

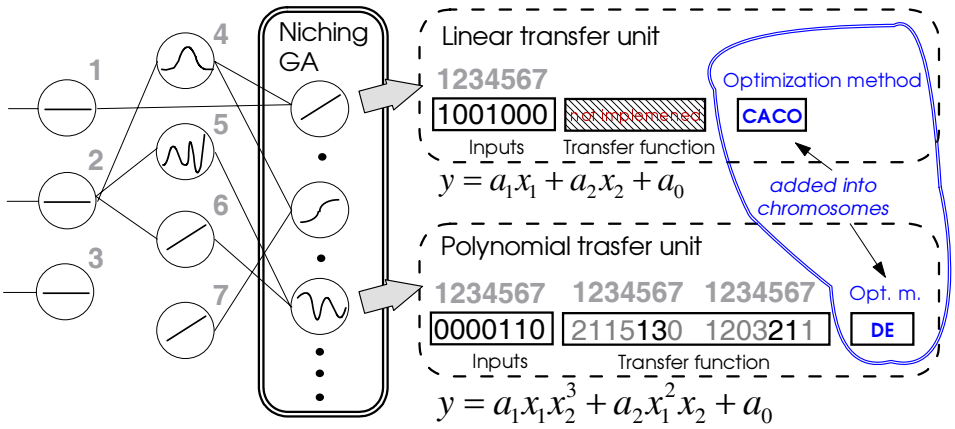

Fig. 6 The example of chromosomes for GAME units with linear and polynomial transfer function. Chromosomes contain encoded input connections and for some units, the structure of the transfer function is also encoded to be able to evolve it. The type of the optimization method was appended to the chromosome.

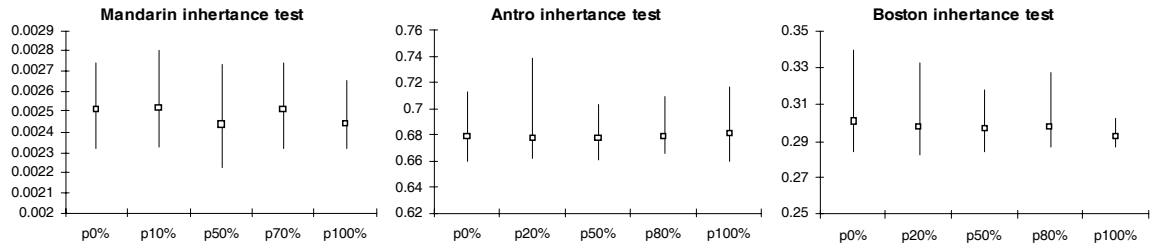

Fig. 7 The experiments with the inheritance of transfer function and learning method. For all three data sets, the fifty percent inheritance level is a reasonable choice.

5 Final comparison of methods

We used the same methodology as for previous experiments on Building data set, Boston and Spiral data sets. Along with these data sets, we used diverse real world data sets described in [1].

Optimization methods are ranked according to the accuracy of their models on several data sets. Figure 8 displays the results.

Final ranking shows, that the Quasi-Newton optimization method was the most successful from individual method. It was also the fastest. The evolution of appropriate(*All*) clearly outperformed all individual methods, but it was much slower than Quasi-Newton method. The reason is that computing time was wasted by inefficient methods that do not use analytic gradient of the error surface (such as PSO). Possible solution is to exclude the least efficient methods (accuracy will decrease just marginally), or to enhance these methods by hybridizing them with gradient based methods.

6 Conclusion

The experiments showed that gradient methods like Quasi Newton and Conjugate Gradients performed very well for all data sets we have been experimenting with. When *All* methods are used, superb performance is guarantied, but the computation is significantly slower (some methods need many iterations to converge). At this stage of the research and implementation, we recommend using the Quasi Newton (QN) optimization method only, because it is the fastest and very reliable. If the computing time is not important for you, the evolution of optimization methods is the best choice.

The evolution of optimization methods is very promising, and we believe that the performance might increase, when we improve properties of individual nature inspired optimization methods.

In our further research we plan to use the analytic gradient to improve the performance of gradient and swarm methods. We also plan to experiment with switching of optimization methods (switch to a different method when a convergence is slow).

7 Acknowledgement

This research is 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 Ed-

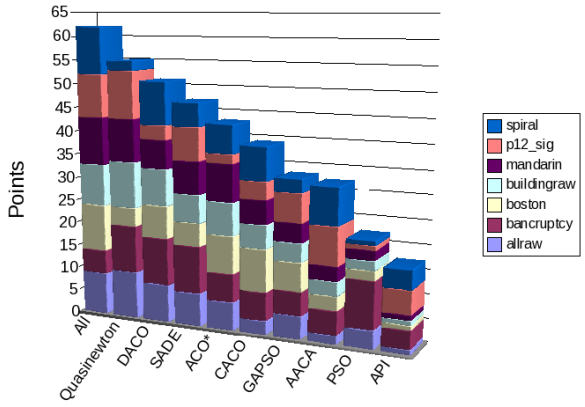

Fig. 8 Final comparison of all tested methods. Points are derived from the ranking for each data test - better position means more points.

ucation, Youth and Sports of the Czech Republic.

8 References

[1] P. Kordík. *Fully Automated Knowledge Extraction using Group of Adaptive Models Evolution*. PhD thesis, Czech Technical University in Prague, FEE, Dep. of Comp. Sci. and Computers, FEE, CTU Prague, Czech Republic, September 2006.

[2] 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.

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

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

[5] 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.

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

[7] Samir W. Mahfoud. Niching methods for genetic algorithms. Technical Report 95001, Illinois Genetic Algorithms Laboratory (IlliGaL), University of Ilinios at Urbana-Champaign, May 1995.

[8] S. W. Mahfoud. A comparison of parallel and sequential niching methods. In *Sixth International Conference on Genetic Algorithms*, pages 136–143, 1995.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] L. Kuhn. *Ant Colony Optimization for Continuous Spaces*. PhD thesis, The Department of Information Technology and Electrical Engineering The University of Queensland, October 2002.

[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] 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.

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

[18] 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.

[19] 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.

[20] 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.