Designing Particle Swarm Optimization with Regression Trees

Th. Bartz-Beielstein[1], M. de Vegt[1], K.E. Parsopoulos[2], and M.N. Vrahatis[2]

[1] University of Dortmund,
Department of Computer Science,
44221 Dortmund, Germany
{thomas.bartz-beielstein, marcel.devegt}@udo.edu
[2] Department of Mathematics,
University of Patras Artificial Intelligence Research
Center (UPAIRC), University of Patras,
GR–26110 Patras, Greece
{kostasp, vrahatis}@math.upatras.gr

Abstract. A new approach for the determination of the parameters of the Particle Swarm Optimization algorithm, by using regression trees, is proposed, and applied to a complex real–world optimization problem, namely the optimization of an elevator group controller, as well as to two well–known benchmark problems. The results support the claim that the proposed technique can enhance the performance of the algorithm and provide proper parameter values. Advantages and drawbacks are discussed and conclusions are derived.

1 Introduction

Modern search heuristics have proved to be very useful for solving complex real–world optimization problems that cannot be tackled through classical optimization techniques [1]. Many of these search heuristics involve a set of exogenous parameters that affect their convergence properties. An optimal parameter setting depends on the problem at hand, as well as on the restrictions posed by the environment (i.e. time and hardware constraints).

Particle Swarm Optimization (PSO) is a swarm intelligence optimization algorithm [2]. The main inspiration for its development was the flocking behavior of swarms and fish schools. PSO has proved to be very efficient in numerous applications in science and engineering [3–7]. PSO's convergence is controlled through a set of parameters that are usually determined empirically, or they are set equal to widely used default values.

We suggest an approach for determining the values of these parameters, tailored for the optimization problem at hand. The proposed technique is based on statistical experimental design and it is applicable to all PSO variants. It can be also applied to any parameterizable search algorithm, such as evolutionary algorithms (EA). To justify the usefulness of our approach, we analyze the properties of PSO algorithms from the viewpoint of an optimization practitioner, in

the context of a real–world optimization problem, namely the optimization of an elevator group controller, as well as to two well–known test functions, extending the approach proposed in [8], which was based on statistical design of experiments and classical regression analysis.

This article is organized as follows: the elevator group control problem, the PSO algorithm, as well as the goals of this optimization problem are introduced in Section 2. Section 3 is devoted to the description of the proposed approach for the specific control problem. Experimental results are reported in Section 4 and discussed in Section 5. A summary and an outlook are given in Section 6.

2 Problem, Algorithm and Environment

Experimental analysis of algorithms deals with the analysis of interactions among problems, algorithms, and environments, based on experiments. To illustrate the usefulness of this approach, we consider an elevator group controller (problem), PSO (algorithm), and restrictions, such as the number of fitness function evaluations, as well as the number of available processors, that appear in real–life applications (environment).

2.1 The Elevator Control Problem

Almost all new office buildings in modern cities are high rise buildings and require efficient elevator systems. The elevator group controller assigns elevators to customer service calls, based on a specific policy. This policy should be optimal with respect to the overall throughput, waiting times, energy consumption, and other different goals. Different building types and traffic situations make controllers and related policies incomparable. This motivated the development of a simplified elevator group control model, the *Sequential Ring* (S–ring). It can be reproduced easily and is well suited as a benchmark problem for elevator group controllers [9].

When passengers give a hall call, they simply press a button. Therefore, only one bit of information for each floor is sent to the group controller and the whole state of the system is mapped to a binary string. The system's dynamic can be represented by a state transition table and it is controlled according to a policy. The S–ring has only a few parameters: the number of elevator cars, m; the number of queues, n; and the passenger arrival rate, p [10]. A 2–bit state (s_i, c_i) is associated with each site, where s_i is the *server bit*, while c_i is the *customer bit*. The server bit is set to 1 if a server is present on the i–th floor, and 0 otherwise, while the customer bit is set to 0, or 1 if there is no waiting passenger or at least one waiting passenger, respectively. Fig. 1 illustrates a typical S–ring configuration. The state of the system at time t is given as

$$x(t) = (s_0(t), c_0(t), \ldots, s_{n-1}(t), c_{n-1}(t)) \in \mathbb{B}^{2n}, \tag{1}$$

where $\mathbb{B} = \{0, 1\}$. The sites are numbered from 0 to $n - 1$.

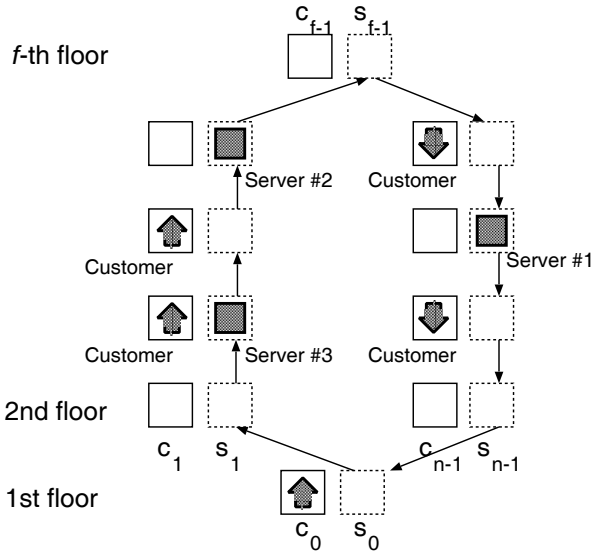

Fig. 1. The S–ring as an elevator system.

The state evolution is sequential, scanning the sites from $n-1$ down to 0, and then again around from $n-1$. The up and down elevator movements can be regarded as a loop. This motivates the ring structure. At each time step, one of the floor queues is considered, where passengers may arrive with probability p. Therefore, the triple $\xi(t) = (c_k(t), s_k(t), s_{k'}(t))$, $k \in \{0, \ldots, n-1\}$, and $k' \equiv (k+1) \mod n$, is updated as follows: if the queue has both a customer and a server present, the server makes a decision to "take" (that corresponds to the bit 1) or "pass" (that corresponds to the bit 0) the customer, according to a policy π. In case of a "take" decision, the customer enters the car, and the server stays there; in the case of "pass", or if there is no customer, the server steps to the next site. As the rules of operation are very simple this model is easily reproducible and suitable for benchmark testing, since it can be easily modeled as a minimization problem[3].

A look–up table, such as Table 1, can be used to represent the dynamic of the system in a compact manner. The triplet $\xi(t)$, in the first column of the table, represents the state of the actual site: customer waiting, server present, and server present on the next floor. The probability of a state change to the state in the fourth column is given in the second column. Columns three and five denote the decision and the reward respectively. The reward is defined as the change in the number of floors with waiting customers. For example, consider the situation in the 7–th row (110). In this case, there is a customer waiting

[3] A reference implementation of the S–ring model can be requested from the authors: thomas.bartz-beielstein@udo.edu.

Table 1. The S–ring lookup table.

$\xi(t)$	Prob	$\pi(x)$	$\xi(t+1)$	Δr
000	$1-p$		000	0
	p		100	-1
100	1		100	0
010	$1-p$		001	0
	p	0	101	-1
		1	010	0
110	1	0	101	0
		1	010	$+1$
001	$1-p$		001	0
	p		101	-1
101	1		101	0
011	1		011	0
111	1		011	$+1$

($1xx$) and the server has to make a decision ("take" or "pass" the customer). There is a server present on the same floor ($11x$) but no server on the next floor (110). In case of a "pass" decision, $\pi(x) = 0$, the situation 101 (customer still waiting, no server present, server on the next floor) occurs. A "take" decision leads to the situation (no customer present, server is still on the floor, and no server is on the next floor), that is 010. As the number of sites with passengers is reduced in the latter case, the reward is $+1$, whereas no reward is given in the former case.

Despite the model's simplicity, it is hard to find an optimal policy π^*, even for a small S–ring, since its difference from heuristic suboptimal policies is non–trivial. The most obvious heuristic policy is the greedy one, i.e. when given the choice, always serve the customer. However, this policy is not optimal, except in the case of heavy traffic ($p > 0.5$). This means that a good policy must bypass some customers occasionally to avoid the "bunching" effect: many elevator cars are positioned in close proximity to another. The performance of the system is poor when bunching occurs. The S–ring is a well suited model to simulate and analyze bunching problems [11].

Let $\theta : \mathbb{R} \rightarrow \mathbb{B}$, be the Heaviside function, $\theta(z) = 0, \text{if } z < 0$, and $\theta(z) = 1, \text{if } z \geqslant 0$, and let $x = x(t)$ be the state at time t, see Eq. (1). Let also $y \in \mathbb{R}^{2n}$ be a weight vector. A linear discriminator, or perceptron, can be used to present the policy in a compact manner. The basic optimal control problem is to find a weight vector y, that represents a policy π^* for the given S–ring configuration, such that, the expected number of sites with waiting passengers in the system is minimized [10, 12].

2.2 Particle Swarm Optimization

Particle swarm optimization (PSO) belongs to the class of stochastic, population based optimization algorithms. The ideas that underlie PSO are inspired by the social behavior of flocking organisms, such as swarms of birds and fish schools, whose behavior appears to adhere to some fundamental rules such as nearest–neighbor velocity matching and acceleration by distance [13].

PSO exploits a population of individuals to probe the search space. In this context, the population is called a *swarm* and the individuals are called *particles*. Each particle moves with an adaptable velocity within the search space, and it keeps in memory the best position it has ever attained. In the *global* variant of PSO, the best position ever achieved by the swarm is communicated to all the particles. In the *local* variant, each particle is assigned a neighborhood that consists of prespecified particles. In this case, the best position ever attained by the particles that comprise the neighborhood is communicated among them [13].

Let $S \subset \mathbb{R}^D$ be a D–dimensional search space, and P be the number of particles of the swarm. Then, the i–th particle is a D–dimensional vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})^\top \in S$. Its velocity is also a D–dimensional vector $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})^\top \in S$. The best position ever visited by the i–th particle is a point in S, denoted as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})^\top$. If g is the index of the particle with the best position ever detected by the swarm, and t is the iteration counter (number of fitness function evaluations), then the resulting equations for the manipulation of the swarm are [14]

$$V_i(t+1) = wV_i(t) + c_1 r_1 \big(P_i(t) - X_i(t)\big) + c_2 r_2 \big(P_g(t) - X_i(t)\big), \tag{2}$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \tag{3}$$

where $i = 1, 2, \ldots, N$; w is a parameter called the *inertia weight*; c_1 and c_2 are positive constants, called the *cognitive* and *social* parameter, respectively; and r_1, r_2 are random numbers, uniformly distributed in $[0, 1]$.

Alternatively, a different version, which incorporates a parameter called *constriction factor* has been proposed [15]. In this version the swarm is manipulated according to the equations:

$$V_i(t+1) = \chi\Big[V_i(t) + c_1 r_1 \big(P_i(t) - X_i(t)\big) + c_2 r_2 \big(P_g(t) - X_i(t)\big)\Big], \tag{4}$$

$$X_i(t+1) = X_i(t) + V_i(t+1), \tag{5}$$

where χ is the constriction factor.

Both the constriction factor and the inertia weight are used to control the magnitude of the velocities. However, their values are determined in different ways. The value of the constriction factor is determined through a formula [15]. On the other hand, the inertia weight, w, is empirically determined. Experimental results indicate that it is preferable to initialize the inertia weight to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions. Thus, an initial value around 1 and a gradual decline towards 0 is considered a proper choice for w.

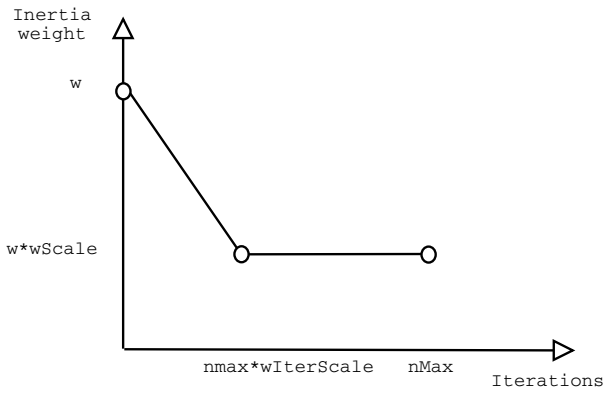

Fig. 2. Scaling of the inertia weight.

Proper fine–tuning of the parameters may result in faster convergence and alleviation of local minima [14]. The swarm and the velocities are usually initialized randomly and uniformly in the search space, although more sophisticated initialization techniques may enhance the performance of the algorithm [16]. Since variance reducing techniques were considered in our analysis, the positions of particles were initialized deterministically based on a scheme proposed in [17]:

$$X_i = (-1)^i/\sqrt{D}, \quad V_i \in [0,1], \quad i = 1,\ldots,D.$$

Regarding the inertia weight, two parameters, $w_{\text{scale}} \in [0,1]$, and $w_{\text{iterScale}} \in [0,1]$, were used. The inertia weight is linear decreasing from w to $w \cdot w_{\text{scale}}$, over $G_{\max} \cdot w_{\text{iterScale}}$, iterations. Then, for the last $G_{\max} \cdot (1 - w_{\text{iterScale}})$ iterations, it has a constant value $w \cdot w_{\text{scale}}$. Figure 2 depicts the scaling procedure.

2.3 Environment

Eiben distinguishes three main types of problems [18]:

1. Design problems that require algorithms that find one excellent solution at least once.
2. Repetitive problems, where good solutions must be found quickly and for different instances of the problem.
3. On–line control problems that are repetitive problems with tight time constraints.

Elevator group control, as considered here, belongs to the 2nd category. In the near future, when the elevator controllers will be capable of learning during their operation, the controllers should be able to adapt their policy on–line, and the problem will belong to the 3rd category.

PSO is a stochastic algorithm, thus it requires random number seeds. An optimization practitioner is interested in robust solutions, i.e. solutions that do not

Table 2. Fractional factorial 2^{6-2}_{IV} design for PSO. This design can be used to screen out important factors. It was used in the first phase of the experimental analysis.

	P	$w_{\max}$	c_1	c_2	w_{scale}	$w_{\text{iterScale}}$
1	5	0.5	0.1	0.1	0.0	0.8
2	40	0.5	0.1	0.1	0.2	0.8
3	5	1.5	0.1	0.1	0.2	1.0
4	40	1.5	0.1	0.1	0.0	1.0
5	5	0.5	2.5	0.1	0.2	1.0
6	40	0.5	2.5	0.1	0.0	1.0
7	5	1.5	2.5	0.1	0.0	0.8
8	40	1.5	2.5	0.1	0.2	0.8
9	5	0.5	0.1	2.5	0.0	1.0
10	40	0.5	0.1	2.5	0.2	1.0
11	5	1.5	0.1	2.5	0.2	0.8
12	40	1.5	0.1	2.5	0.0	0.8
13	5	0.5	2.5	2.5	0.2	0.8
14	40	0.5	2.5	2.5	0.0	0.8
15	5	1.5	2.5	2.5	0.0	1.0
16	40	1.5	2.5	2.5	0.2	1.0

depend on the random seeds that are used to generate the random streams. The statistical methodology presented here proposes guidelines to design robust PSO algorithms under restrictions, such as a limited number of function evaluations and just a few processors. These restrictions can be modeled if we consider the performance of an algorithm as the (expected) best fitness function value for a limited number of fitness function evaluations.

3 Design of Experiments and Regression Trees

Statistical methods, such as experimental design techniques and regression analysis, can be used to analyze the experimental setting introduced in the previous section. The classical regression analysis approach can be enhanced by using tree–based regression methods. Regression trees appear well suited to screen out important factor settings.

A 2^{6-2}_{IV} fractional factorial design was considered to perform the screening experiments with PSO [19]. The corresponding factor settings are reported in Table 2. We considered a sequential approach that combines existing and new results and enables a step–wise increase in the regression model complexity. Starting with a simple linear model, the final model can be analyzed with response surface methods (RSM). The main focus of this paper lies on the screening phase, while RSM and other approaches like spatial regression techniques will be discussed in a forthcoming paper.

The most popular modeling methods is linear regression. It fits a straight line to a set of data values. The form of the function fitted by linear regression

Table 3. Comparison of the first with the improved designs. The first setting refers to the values from Table 2. $G_{\max} = 5,000$ function evaluations were used. N denotes the number of performed experiments (repeats with different random seeds) for each scenario.

Function	Configuration	median	mean fitness	variance	N
Sphere	1st setting	0.0320	0.1818	0.0473	160
Sphere	improved	1.7040e-70	3.1153e-64	9.5433e-127	10
Rosenbrock	1st setting	0.6092	2.0006	8.2762	160
Rosenbrock	improved	0.0051	0.0297	0.0032	100
S-ring	(see Sec.4)	5.9331	5.9825	0.4204	192
S-ring	improved	5.5768	5.6285	0.2339	10

is $y = \beta X$, where β is a parameter vector whose values are determined so the function best fits the data. Linear regression is appropriate only if the data can be modeled by a straight line function, which is often not the case.

Breiman *et al.* introduced regression trees as a "flexible non–parametric tool to the data analyst's arsenal" [20]. They are used for screening variables and for checking the adequacy of regression models [21]. The construction of regression trees can be seen as a type of variable selection similar to the stepwise regression techniques in classical regression analysis [22]. Consider a set of predictor variables X and a quantitative response variable Y. A regression tree is a collection of rules such as "if $x_1 \leqslant 5$ and $x_4 \in \{A, C\}$, then the predicted value of Y is 14.2", that are arranged in a form of a binary tree. Recursively splitting the data in each node, builds up a binary tree. The partitioning algorithm stops when the node is homogeneous or the node contains too few observations. The endpoint for a tree is a partition of the space of possible observations. Compared to linear models, tree–based models are easier to interpret when qualitative and quantitative predictors are in the model.

4 Experimental Results

We first investigated the performance of the regression trees approach, on simple well–known test functions, to gain an intuition regarding the algorithm's workings. For this purpose, the test functions described below were used, where PSO's parameters were tuned through the regression tree approach. In all cases, a maximum value of $G_{\max} = 5000$ function evaluations appears to be appropriate.

Test Function 1 (Sphere). This is a unimodal function, $F(x) = \sum_{i=1}^{D} x_i^2$, with a single minimum at $X_{opt} = 0$, with $F(X_{opt}) = 0$. We considered $D = 36$ in our experiments. The values that were chosen for the first run parameterization, are reported in Table 2 . After two additional steps, the tuned setting reads: $P = 7$, $w_{\max} = 0.2$, $c_1 = 0.4$, $c_2 = 0.7$, $w_{\text{scale}} = 0.025$, and $w_{\text{iter}} = 0.5$. The mean of the first and the improved fitness value are 0.1818 and $3.1153e-64$, respectively.

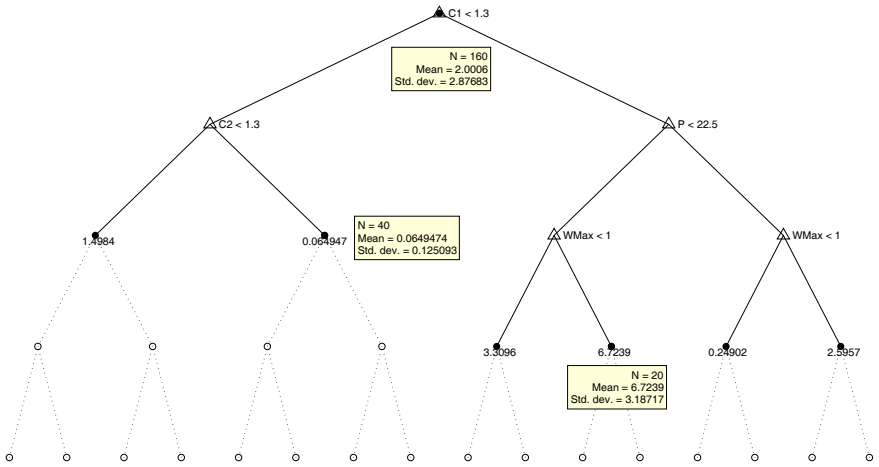

Fig. 3. Results for the Rosenbrock function. Regression tree for the first configuration from Table 2. The root node shows the overall mean $\mu = 2.0006$ and standard deviation 2.88 for $N_{exp} = 160$ experiments. Following the left branch to the next node (labeled $C2 < 1.3$), and taking the right branch, we can see that the mean fitness value reads 0.006. Therefore, a c_1 smaller than 1.3, and c_2 larger than 1.3, might lead to an improved performance. The performance is worse, if we chose $c_1 \geq 1.3$, $P < 22.5$, and $w_{\max} \geq 1.0$. The solid lines show the pruned tree.

Test Function 2 (Rosenbrock). This function is defined as,

$$F(x) = \sum_{i=1}^{D-1} [100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2],$$

with its sole minimum at $X_{opt} = (1, \ldots, 1)$, with $F(x_{opt}) = 0$. We considered $D = 2$. The improved parameterization reads $P = 40$, $w = 1.5$, $c_1 = 0.1$, $c_2 = 2.5$, $w_{\text{scale}} = 0$, and $w_{\text{iterScale}} = 0.8$.

In the next step of our analysis, the S–ring model was considered, with $n = 18$, $m = 6$, $p = 0.1$. Surprisingly, the method that has been applied successfully to the classical test functions, failed. The regression tree collapsed: it consisted of one node only, and no conclusions could be drawn on how to improve the PSO parameterization. Further investigation revealed the cause of this failure: the levels chosen in Table 2 were too extreme for this problem. A swarm size, P, of 5 particles leads to a performance that is as worse as the performance of a swarm that has 40 particles. Therefore, a new design, which is more conservative than the former, was chosen: $P \in \{10, 20\}$, $w_{\max} \in \{0.8, 1.2\}$, $c_1 \in \{1.5, 2.0\}$, $c_2 \in \{1.5, 2.0\}$, $w_{\text{scale}} \in \{0, 0.1\}$, and $w_{\text{iterScale}} \in \{0.9, 1.0\}$. Based on this design, the tree based approach was able to find an improved parameter setting: $P = 25$, $w_{\max} = 0.7$, $c_1 = 1.2$, $c_2 = 2.5$, $w_{\text{scale}} = 0.15$, and $w_{\text{iterScale}} = 0.75$. A comparison between the initial and the improved values is reported in Table 3. Finally a t–test

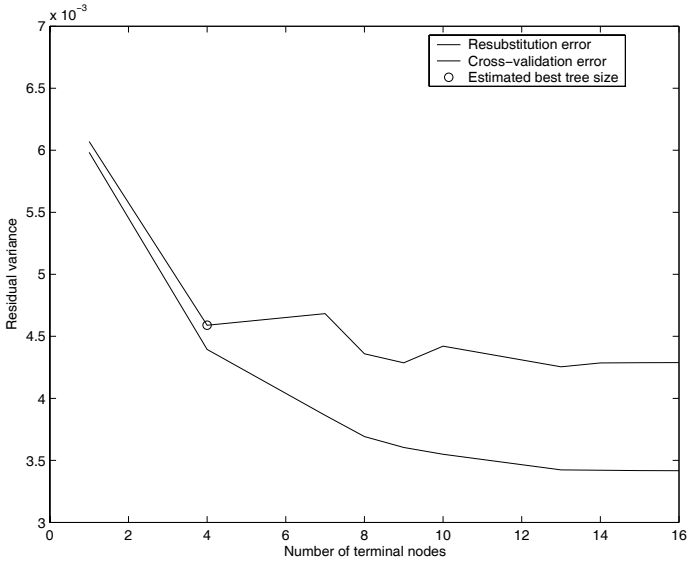

Fig. 4. S-ring. Complexity plot to determine the "best" tree size. The best tree is estimated by cross validation. It is the one that has a residual variance that is no more than one standard error above the minimum value along the cross validation line (upper line). The monotonically decreasing resubstitution error is shown in the lower line. The optimal tree has 4 terminal nodes.

showed, that the hypothesis, the means of the first and the improved parameter setting are equal, has to be rejected.

5 Discussion

Table 3 shows that regression trees provide effective and efficient means to improve the PSO performance significantly. Only a few tree growing phases (< 5) were needed to find better parameterizations. Modern statistic software packages like `R` provide powerful tools to perform this tuning [23]. Additionally to `R`, `MATLAB` was used. In addition to the regression tree based analysis, a classical regression analysis was performed, i.e. a stepwise model selection by exact AIC was applied to screen out important factors. Regression trees have shown their ability to model the dependencies between different PSO parameterizations in a very intuitive manner. They are more conservative than the classical regression techniques. Their results are reliable, as we obtained similar results from regression analysis. There were no conflicts of the two approaches during the optimization process. The corresponding regression model are much more complicated and require assumptions on the underlying distribution. Even hard real–world optimization problems can benefit from this approach. A drawback of the proposed approach, is the determination of a good starting design. This

problem is common to all statistical methods in this field: as a good parameterization depends on the problem and the environment (we might have obtained different results if our analysis has been based on 100000 fitness function evaluations or a completely different termination criterion), a good initial design cannot be known a priori for new optimization problems (e.g. the S–ring). Generally, we can recommend a sequential design that is extended during the tuning procedure and leads to more and more complex trees and corresponding regression models.

6 Summary and Outlook

A framework to design particle swarm algorithms with regression trees has been proposed. Regression trees have been shown as valuable tools to model the interactions between the algorithm, the problem, and the problem environment. The exogenous parameters of PSO was tuned and therefore the performance of the algorithm was improved significantly. Tuning did not only result in an improved fitness value, but it also improved the robustness of the parameterization: small parameter changes for tuned algorithms did not cause big differences in performance.

Regression trees are easily to interpret, they complement traditional regression techniques and do not make any assumptions on the underlying distribution. Furthermore, they are applicable for both quantitative and qualitative data. The latter are necessary to compare different PSO variants or PSO with different operators.

As the examples in this paper demonstrate, it might be a good practice to compare only tuned algorithms (and to report the extra costs for the tuning process).

References

1. H.-P. Schwefel, I. Wegener, and K. Weinert, editors. *Advances in Computational Intelligence – Theory and Practice*. Natural Computing Series. Springer, Berlin, 2003.
2. J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Piscataway, NJ, 1995. IEEE Service Center.
3. M. A. Abido. Optimal design of power system stabilizers using particle swarm optimization. *IEEE Trans. Energy Conversion*, 17(3):406–413, 2002.
4. C. O. Ourique, E. C. Biscaia, and J. Carlos Pinto. The use of particle swarm optimization for dynamical analysis in chemical processes. *Computers and Chemical Engineering*, 26:1783–1793, 2002.
5. K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1(2–3):235–306, 2002.
6. K. E. Parsopoulos, E. I. Papageorgiou, P. P. Groumpos, and M. N. Vrahatis. Evolutionary computation techniques for optimizing fuzzy cognitive maps in radiation therapy systems. *Lecture Notes in Computer Science (LNCS)*, 2004. in press.

7. N. G. Pavlidis, K. E. Parsopoulos, and M. N. Vrahatis. Computing nash equilibria through computational intelligence methods. *Journal of Computational and Applied Mathematics*, 2004. in press.
8. T. Beielstein, K. E. Parsopoulos, and M. N. Vrahatis. Tuning PSO parameters through sensitivity analysis. Technical Report CI 124/02, Department of Computer Science, University of Dortmund, Dortmund, Germany, 2002.
9. Thomas Beielstein, Sandor Markon, and Mike Preuß. Algorithm based validation of a simplified elevator group controller model. In T. Ibaraki, editor, *Proc. 5th Metaheuristics Int'l Conf. (MIC'03)*, pages 06/1–06/13 (CD–ROM), Kyoto, Japan, 2003.
10. S. Markon, D.V. Arnold, T. Bäck, T. Beielstein, and H.-G. Beyer. Thresholding – a selection operator for noisy ES. In J.-H. Kim, B.-T. Zhang, G. Fogel, and I. Kuscu, editors, *Proc. 2001 Congress on Evolutionary Computation (CEC'01)*, pages 465–472, Seoul, Korea, May 27–30, 2001. IEEE Press, Piscataway NJ.
11. G. Barney. *Elevator Traffic Analysis, Design and Control.* Cambridge U.P., 1986.
12. T. Beielstein and S. Markon. Threshold selection, hypothesis tests, and DOE methods. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 777–782. IEEE Press, 2002.
13. J. Kennedy and R. C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.
14. R. C. Eberhart and Y. Shi. Comparison between genetic algorithms and particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming*, volume VII, pages 611–616. Springer, 1998.
15. M. Clerc and J. Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.*, 6(1):58–73, 2002.
16. K. E. Parsopoulos and M. N. Vrahatis. Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method. In A. Grmela and N. E. Mastorakis, editors, *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 216–221. WSEAS Press, 2002.
17. H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology. Wiley Interscience, New York, 1995.
18. A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing.* Springer, 2003.
19. A.M. Law and W.D. Kelton. *Simulation Modelling and Analysis.* McGraw-Hill Series in Industrial Egineering and Management Science. McGraw-Hill, New York, 3rd edition, 2000.
20. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.
21. T. M. Therneau and E. J. Atkinson. An introduction to recursive partitioning using the rpart routines. Technical Report 61, Department of Health Science Research, Mayo Clinic, Rochester, 1997.
22. J. M. Chambers and T. H. Hastie, editors. *Statistical Models in S.* Wadsworth & Brooks/Cole, Pacific Grove, California, 1992.
23. R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.