EXTRACTION OF FUNCTION FEATURES FOR AN AUTOMATIC CONFIGURATION OF PARTICLE SWARM OPTIMIZATION

Tjorben Bogon[1,2], Georgios Poursanidis[1], Andreas D. Lattner[1] and Ingo J. Timm[2]
[1]*Information Systems and Simulation, Institute of Computer Science and Mathematics*
Goethe University Frankfurt, Frankfurt, Germany
[2]*Business Informatics I, University of Trier, Trier, Germany*

Keywords: Particle swarm optimization, Machine learning, Swarm intelligence, Parameter configuration, Objective function feature Computation.

Abstract: In this paper we introduce a new approach for automatic parameter configuration of Particle Swarm Optimization (PSO) by using features of objective function evaluations for classification. This classification utilizes a decision tree that is trained by using 32 function features. To classify different functions we compute features of the function from observed PSO behavior. These features are an adequate description to compare different objective functions. This approach leads to a trained classifier which gets as input a function and returns a parameter set. Using this parameter set leads to an equal or better optimization process compared to the standard parameter settings of Particle Swarm Optimization on selected test functions.

1 INTRODUCTION

Metaheuristics in stochastic local search are used in numerical optimization problems in high-dimensional spaces. For varying types of mathematical functions, different optimization techniques vary w.r.t. the optimization process (Wolpert and Macready, 1997). A characteristic of these metaheuristics is the configuration of the parameters (Hoos and Stützle, 2004). These parameters are essential for the efficient optimization behavior of the metaheuristic but depend on the objective function, too. An efficient set of parameters influences the optimization in speed and performance. If a good parameter set is selected, an adequate solution will be found faster compared to a bad configuration of the metaheuristic. The choice of the parameters is based on the experience of the user and his knowledge about the domain or on empirical research found in literature. This parameter settings, called standard configurations, perform a not optimal but an adequate optimization behavior for most objective functions. An example for metaheuristics is the Particle Swarm Optimization (PSO). PSO is introduced by (Eberhart and Kennedy, 1995) and is a population-based optimization technique which is used in continuous high dimensional search spaces.

PSO consists of a swarm of particles which "fly" through the search space and update their position by taking into account their own best position and depending on the topology, the best position found by other particles. PSO is an example for the parameter configuration problem. If the parameters are well chosen, the whole swarm will find an adequate minimum and focus on this solution. The swarm slows down the velocity trying to get better values in the continuous search space around this found solution. This exploitation can be on a local optimum especially if the wrong parameter set is chosen and the swarm cannot escape from this local minimum. On the other hand the particles can never find the global optimum if they are too fast and never focus. This swarm behavior depends mainly on the chosen parameter and leads to solutions of different quality.

One problem in choosing the right parameters without knowledge about the objective function is to identify relevant characteristics of the function which can be used for a comparison among functions. The underlying assumption is that, e.g., a function $f_1 = x^2$ and a function $f_2 = 3x^2 + 2$ exhibit similar optimization behavior if the same parameter set for a Particle Swarm Optimization is used. In order to choose a promising parameter set, functions must be compara-

Bogon T., Poursanidis G., D. Lattner A. and J. Timm I..
EXTRACTION OF FUNCTION FEATURES FOR AN AUTOMATIC CONFIGURATION OF PARTICLE SWARM OPTIMIZATION.
DOI: 10.5220/0003134500510060
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 51-60
ISBN: 978-989-8425-40-9

ble with respect to certain objective function characteristics.

We describe an approach to computing features of the objective function by observing the swarm behavior. For each function we seek for a parameter set that performs better than the standard configuration and provide this set as output class for supervised learning. These data allow us to train a C4.5 decision tree (Quinlan, 1993) as classifier that computes an adequate configuration for the Particle Swarm Optimization by using function features. Experimental trials show that our decision tree classifies functions into the correct classes in many cases. This classification can be used to select promising parameter sets for which the Particle Swarm Optimization is expected to perform better in comparison to the standard configuration.

This work is structured as follows: In section 2 we describe other approaches pointing out the problem of computing good parameter sets for a metaheuristic and explain the Particle Swarm Optimization. Section 3 describes how to compute the features of a function and thereby make the functions comparable. After computing the features we describe our experimental setup and the way to build up the decision tree. Section 4 contains our experimental results for building the parameter classes to select promising parameter sets in PSO. The last section discusses our results and describes issues for future work.

2 PARAMETER SETTINGS IN METAHEURISTICS

The main difference between solving a problem with exact methods or with metaheuristics is the quality of the solution. Metaheuristics – for example, nature inspired metaheuristics (Bonabeau et al., 1999) – have no guarantee of finding the global optimum. They focus on a point in the multidimensional search space which results to the best fitness value depending on the experience of the past optimization performance. This can be a local optimum, too. But the advantage of the metaheuristic is to find an adequate solution of a multidimensional continuous optimization problem in reasonable time (Talbi, 2009). This performance depends on the configuration of the metaheuristics and is an important fact of using metaheuristics. One group of metaheuristics are the population based metaheuristics. (Talbi, 2009) defines population-based metaheuristics as nature inspired heuristics which handle more than one solution at a time. With every iteration all solutions are recomputed based on the experience of the whole population. Examples of population-based metaheuristics are Genetic Algorithms which are an instance of Evolutionary Algortihms, Ant Colony Optimization and Particle Swarm Optimization. Different kinds of metaheuristics exhibit varying performance on a specific kinds of problem types. They differ w.r.t. the optimization speed and the solution quality. A metaheuristic's performance is based on their configuration. Finding a good parameter set is a non-trivial task and often based on a priori knowledge about the objective function and the problem. Setting up a metaheuristic with standard parameter sets lets the optimization find a decent solution but using a parameter set which is adapted to the specific objective function might even lead to better results. In this paper we focus on PSO and try to find features characterizing the objective function in order to select an adequate parameter configuration for this metaheuristic. The optimization behavior of the particles is based on the objective function and we try identify relevant information about the function. In the following section we give a brief introduction to particle swarm optimization.

2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is inspired by the social behavior of flocks of birds and shoals of fish. A number of simple entities, the particles, are placed in the domain of definition of some function or problem. The fitness (the value of the objective function) of each particle is evaluated at its current location. The movement of each particle is determined by its own fitness and the fitness of particles in its neighborhood in the swarm. PSO was first introduced in (Kennedy and Eberhart, 1995). The results of one decade of research and improvements to the field of PSO were recently summarized in (Bratton and Kennedy, 2007), recommending standards for comparing different PSO methods. Our definition is based on (Bratton and Kennedy, 2007). We aim at continuous optimization problems in a search space $\mathcal{S}$ defined over the finite set of continuous decision variables $X_1, X_2, \ldots, X_n$. Given the set Ω of conditions to the decision variables and the objective function $f : \mathcal{S} \rightarrow \mathcal{R}$ (also called fitness function) the goal is to determine an element $s^* \in \mathcal{S}$ that satisfies Ω and for which $f(s^*) \leq f(s)$, $\forall s \in \mathcal{S}$ holds. $f(s^*)$ is called a global optimum.

Given a fitness function f and a search space $\mathcal{S}$ the standard PSO initializes a set of particles, the swarm. In a D-dimensional search space $\mathcal{S}$ each particle P_i consists of three D-dimensional vectors: its position $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$, the best position the particle

visited in the past $\vec{p}_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$ (particle best) and a velocity $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. Usually the position is initialized uniformly distributed over $\mathcal{S}$ and the velocity is also uniformly distributed depending on the size of $\mathcal{S}$. The movement of each particle takes place in discrete steps using an update function. In order to calculate the update of a particle we need a supplementary vector $\vec{g} = (g_1, g_2, \ldots, g_D)$ (global best), the best position of a particle in its neighborhood. The update function, called inertia weight, consists of two parts. The new velocity of a particle P_i is calculated for each dimension $d = 1, 2, \ldots, D$:

$$v_{id}^{new} = w \cdot v_{id} + c_1 \varepsilon_{1d} (p_{id} - x_{id}) + c_2 \varepsilon_{2d} (g_d - x_{id}) \quad (1)$$

then the position is updated: $x_{id}^{new} = x_{id} + v_{id}^{new}$. The new velocity depends on the global best (g_d), particle best (p_{id}) and the old velocity (v_{id}) which is weighted by the inertia weight w. The parameters c_1 and c_2 provide the possibility to determine how strong a particle is attracted by the global and the particle best. The random vectors $\vec{\varepsilon}_1$ and $\vec{\varepsilon}_2$ are uniformly distributed over $[0,1)^D$ and produce the random movements of the swarm.

2.2 Algorithm Configuration Problem

The general problem of configuring algorithms (algorithm configuration problem) is defined by Hutter et al. (Hutter et al., 2007) as finding the best tuple θ out of all possible configurations Θ ($\theta \in \Theta$). θ represents a tuple with a concrete assignment of values for the parameter of an algorithm. Applied to metaheurisitcs the configuration of the algorithm parameters for a specific problem influences the behavior of the optimization process. Different parameter settings exhibit different performances at solving a problem. The problem to configure metaheuristics is a super ordinate problem and is analyzed for different kinds of metaheuristics. In PSO the convergence of the optimization depending on different parameter settings and different functions are analyzed by (Trelea, 2003), (Shi and Eberhart, 1998) and (van den Bergh and Engelbrecht, 2002). But these approaches focus only on the convergence of the PSO but not on function characteristics and the relationship between the parameter configuration and the function landscape.

Different approaches to solve this algorithm configuration problem on metaheurisitcs are introduced: One approach is to find sets of adequate parameters which performs a good optimization on most different types of objective functions. This "standard parameters" are evaluated on a preset of functions to find a parameter set which leads to global good behavior of the metaheuristic. In PSO standard parameter sets are presented by (Clerc and Kennedy, 2002) and (Shi and Eberhart, 1998). Some approaches do not present a preset of parameters but change the values of the parameters during the runtime to get a better performance (Pant et al., 2007).

Another approach is introduced by Leyton-Brown et al. They try to create features which describe the underlying problem (Leyton-Brown et al., 2002) and generate a model predicting the right parameters depending on the classification. They introduce several features which are grouped into nine groups. The features include, among others, problem size statistics, e.g. number of clauses and variables, and measures based on different graphical representations. This analysis is based on discrete search spaces because on continuous search spaces it is not possible to set adequate discrete values for the parameter configuration which is needed by their appraoch.

Our problem is to configure an algorithm working on continuous search spaces and offers infinite possibilities of parameter sets. To solve this challenge we try, similar to Leyton-Brown et al., to train a classifier with features of the fitness function landscape computed by observing swarm behavior. These features are computed and combined with the best found parameter set on the function to a training instance (see figure 1). With a trained classifier at hands we compute the features of the objective function prior to the start of the optimization process. The classifier – in our case a decision tree – classifies the function and selects the specific parameter set that is expected to perform better in the optimization process than using the standard parameters. In our first experiments, which we understand as proof of concept, we choose only a few functions which do not represent any specific types of function. We want to show that our technique is able to identify functions based on the swarm behavior provided features and thereby, select the specific parameter configuration. In order to learn the classifier which suggests the parameter configuration, different function features are computed. These features are the basis of our training instances.

3 COMPUTATION OF FUNCTION FEATURES

Our computed features can be divided into three groups. Each group implies a distinct way of collecting information about the fitness topology of the objective function from particles. The first group *Random Probing* describes features which are calculated based on a random selection of fitness values and provides a general overview of the fitness topology.

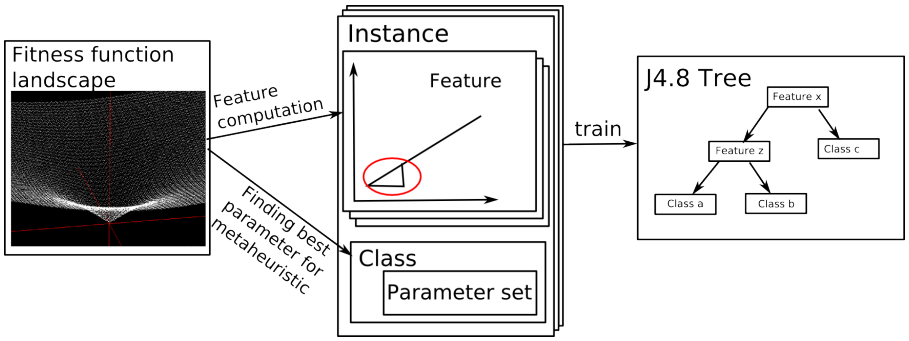

Figure 1: Process of building a classifier.

Distance-based features are calculated for the second group *Incremental Probing*. They reflect the distribution of surrounding fitness values of some pivot particles. The third group of features utilizes the dynamics of PSO to create features by using the changes of the global best fitness within a small PSO instance. The features are scale independent, i.e., that scaling the objective function by constants will not affect the feature values. By this we imply that a configuration for PSO leads to the same behavior on a function f as it shows for its scaled function $f' = \alpha f + \beta, \alpha > 0$. These three groups are based on each other which means that the pivot particle for the second group is taken from a particle of the first group to reduce the computing time. Important for all these features are the number of evaluations of the objective function. The feature computation should be only a small part of the whole optimization computation time.

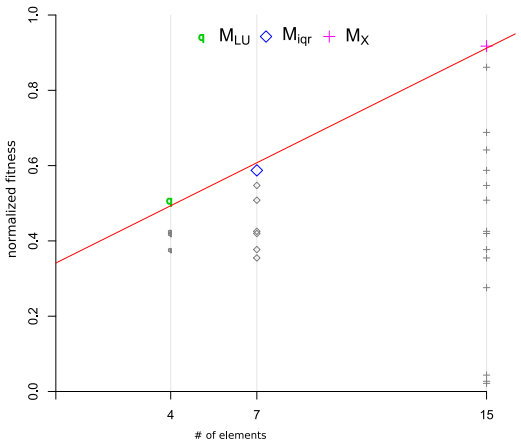

Figure 2: Example of a random probing with the comutation of $\mu_{RP.Max}$.

3.1 Random Probing

Random Probing defines features that are calculated based on a set of $k = 100$ random particle positions which are within the initialization range of the objective function (100 particles to get a short but adequate description about the function window). Probing the objective function results in a distribution of fitness values which is used to extract three features. Trivial characteristics like mean and standard deviation cannot be used as features since they are not scale independent. That means, they will change their value if the function is scaled by constants. In order to create reliable features, the fitness values of all points are evaluated and three sets of particles (including their evaluation values) are created based upon these values. The first set is denoted M_X and contains all fitness values of the randomly selected points. The quartiles for the distribution of the fitness values are computed and the values between the upper or lower quartile are joined into the second set. This set is denoted M_{iqr}. The third set M_{LU} consists of the fitness values which are between a lower and upper boundary L and U. These boundaries are defined by $L = Q_1 + \frac{1}{2}(Q_M - Q_1)$ and $U = Q_M + \frac{1}{2}(Q_3 - Q_M)$ where Q_M denotes the median and Q_1, Q_3 denote the lower and upper quartile of M_X. For each set $M_X \supset M_{iqr} \supset M_{LU}$ the number of elements is determined.

The feature `Random Probing Min` $\mu_{RP.Min}$ is calculated based on the linear model that fits the relationship between the number of values and the minimum fitness values in each set. The straight line of the model is divided by the interquartile range of M_X. Similar to this the feature `Random Probing Max` is based on the slope of the straight line that describes the relationship of the number of elements and the maximum value of each set $M_X \supset M_{iqr} \supset M_{LU}$ (see figure 2). The slope divided by the interquartile range of M_X denoted by $\mu_{RP.Max}$ is the second feature of this group. Finally, for the feature `Random Probing`

`Range` denoted by $\mu_{RP.Range}$ the spread, that is the difference between the maximum and the minimum value, in each set is computed. As for the other features the slope is divided by the interquartile range of M_X. All features of this group are computed based on the fitness values of the randomly selected points. For each point the objective function is evaluated once, hence, $k = 100$ evaluations are necessary for *Random Probing*.

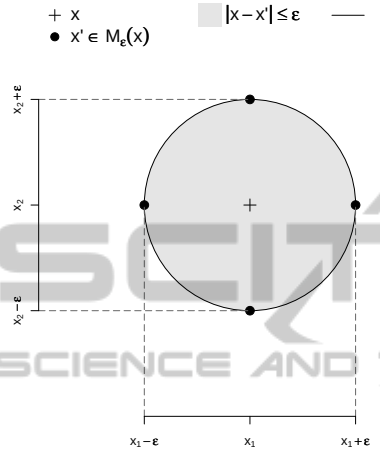

Figure 3: Example of an incremental group in a 2 dimensional space.

3.2 Incremental Probing

In contrast to the features of the previous group, *Incremental Probing* is computed by the fitness values of the particle positions which are located in a defined distance to a pivotal element which we choose from the feature group above. In order to calculate the relevant fitness values, the position of a randomly selected pivot element is consecutively shifted into one dimension. The distance is given by an increment $\varepsilon > 0$ which shifts the position of the pivot element in both directions of the dimension. In each dimension *i Incremental Probing* considers two points (see figure 3 for a 2 dimensonal example). For a given pivot element $x = (x_1, \ldots x_n)$ and a given increment $\varepsilon > 0$ these posistions are determined by

$$|x_i - x_i'| = \begin{cases} \varepsilon & |i = j \\ 0 & |i \neq j \end{cases} \tag{2}$$

where $j, i \leq n$. The increment ε is defined relatively to the domain. For instance, in a restricted n-dimensional domain $\mathbb{A} = I_1 \times \ldots \times I_n$, where the interval $I_i = [a_i, b_i]$ defines the valid subspace, the increment is applied as $\varepsilon \cdot \frac{b_i - a_i}{100}$. For each dimension the position of the pivot element is shifted into two directions. This leads to a set of $2n+1$ points including the pivot element. The fitness value of each valid point is calculated and these fitness values are used for the extraction of objective features.[1] In this group of features, nine features are created with the use of three increments of $\varepsilon_1 = 1$, $\varepsilon_2 = 2$ and $\varepsilon_5 = 5$. Let n be the dimension of the domain, then $2n+1$ evaluations are required to calculate the fitness values of the relevant points. Since three increments are used there are $(3 \times 2n) + 1$ evaluations required to calculate the features of *Incremental Probing*.

The features `Incremental Min`, `Incremental Max` and `Incremental Range` are computed similar to the features of *Random Probing*. For each increment the minimum, maximum and the spread of the fitness values are computed. `Incremental Min` describes the relationship of the minimum and the corresponding increment. There are two subtypes for this feature. $\mu_{IP.Min}$ is divided by the slope of the model's straight line by the spread of the first increment whereas the second subtype $\mu_{IP.MinQ}$ divides the slope by the interquartile range of the first increment. The features `Incremental Max` and `Incremental Range` are handled accordingly. Three additional features are created by separately looking at the fitness values of the individual increments. The fitness values of each increment is sorted in ascending order and normalized into the interval $[0,1]$. This results in a sequence $\langle x_k \rangle = x_1, \ldots, x_k$ and we calculate a measure of linearity by

$$\mu_{IP.Fit} = \sum_{i=1}^{k} \left(x_i - \frac{i-1}{k-1} \right)^2 \tag{3}$$

where $\forall i < j : x_i < x_j$.

3.3 Incremental Swarming

The features of *Incremental Swarming* use the dynamic behavior of PSO to extract features of the objective function. Therefore, we construct a small swarm of two particles and initiate an optimization run. The particles are initialized with a defined distance to each other. We use a inertia PSO with parameter $\theta = (0.6221, 0.5902, 0.5902)$ and record the best solution found in the first $t = 20$ iterations. To get the parameter set θ we evaluated a few parameter sets empirically to find good values which lead to a fast convergence of the small swarm. The spread of the global best fitness is the difference between the first and the last fitness value. The development of the global best fitness depends on the initial positions of the particles. Consider a swarm which is initialized at a local optimum. Once a better fitness value is found, global best

[1] In case that the point is invalid, that is it lies outside the valid domain, the evaluation of the fitness value is skipped and the fitness value of the pivot element is used instead.

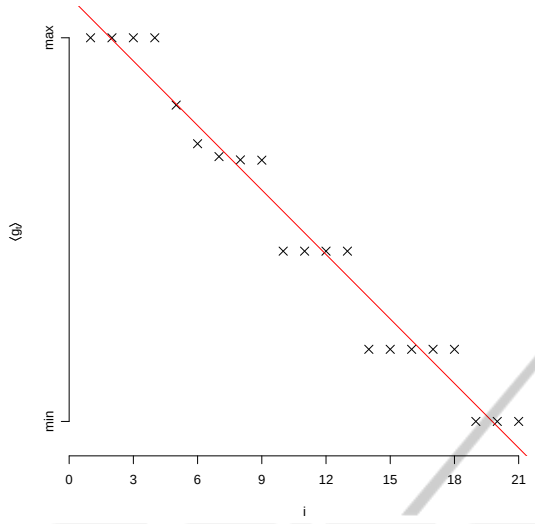

Figure 4: Example of an incremental swarming slope where g describes the best fitness of the actual evaluation step i.

fitness will change. But this may not happen in the few iterations that are observed. Therefore the swarm is initialized by a pivot element chosen from a set of evaluated points. *Incremental Swarming* considers a set of $k = 100$ evaluated solutions and the position which evaluates to the worst fitness value is chosen as pivot element. This is important because if we choose a pivot element randomly, it is possible to find a local optimum and the behavior of the swarm results in no movement. The other particle is initialized in a defined distance to the pivot element. Similarly to *Incremental Probing* we use increments to define the distance between the particles. The increment values $\varepsilon_1 = 1$, $\varepsilon_2 = 2$, $\varepsilon_5 = 5$ and $\varepsilon_{10} = 10$ are used to create 20 features. For each increment the feature `Swarming Slope` describes the development of the global best fitness as a linear model that fits the relationship between the iteration and the global best fitness value (see figure 4). For the feature $\mu_{IS.Slope}$ the slope of the straight line is divided by the spread of the global best fitness. `Swarming Max Slope` describes the greatest change of the global best fitness value between two successive iterations. For normalization the value of $\mu_{IS.Max}$ is divided by the spread of the global best fitness. The other three features, which are computed for each increment, are `Swarming Delta Lin` $\mu_{IS.Lin}$, `Swarming Delta Phi` $\mu_{IS.Phi}$, and `Swarming Delta Sgm` $\mu_{IS.Sgm}$. They describe to what degree the observed development of the global best fitness value differs from sequences that represent idealistic developments. `Swarming Delta Lin` implies a measure of linearity, thus quantifies how much the observed development differs from a linear decrease of the global best fitness. Let $\langle x_t \rangle = x_0, \ldots, x_t$ denote the observed sequence of the global best fitness value. We compute this feature with equation 4.

$$\mu_{IS.Lin} = \sum_{i=0}^{k} \left(x_i - \frac{t-i+1}{t-1} \right)^2 \tag{4}$$

Similarly we create two additional ideal sequences and compute the features $\mu_{IS.Phi}$ and $\mu_{IS.Sgm}$ by the equations 5–6:

$$\mu_{IS.Phi} = \sum_{i=0}^{k} \left(x_i - \phi^{i-1} \right)^2 \tag{5}$$

$$\mu_{IS.Sgm} = \sum_{i=0}^{k} \left(x_i - \frac{1}{1+\exp^{(i-1)\phi}} \right)^2 \tag{6}$$

where $\phi = \frac{2}{1+\sqrt{5}}$. The factor ϕ was selected in order to mediate between a linear and an exponential developing. The development of the global best fitness is used to calculate the features of *Incremental Swarming*. The pivot element for the initialization of the swarm is chosen from a set of k solutions and since the swarm of $m = 2$ particles is applied for $t = 20$ iterations, overall there are $k + m + mt$ evaluations of the objective function. We choose the pivot element from the set M_X which was created for the features of *Random Probing*. By this we reduce the number of additional evaluation to $m + mt = 42$.

4 EVALUATION

In this section we evaluate our features and build a classifier which computes specific parameter sets for the Particle Swarm Optimization on a specific function. This optimization should have a better performance compared to the PSO on the same function with standard parameter set.

4.1 Experimental Setup

We choose 7 test functions out of the suggested test function pool from (Bratton and Kennedy, 2007) and stop computing the fitness function after 300000 times. With our swarm size of 30 the number of epochs is consequently set to 10000. We define a run as a parameter set which is tested 90 times with a finite set of different seed values in order to get meaningful results. As topology of the swarm *gbest* is used. The initialization of the particle is in a defined square of the search space (see table 1). Before we start to train our classifier with the features we have to create the classes that represent specific parameter sets with a high quality of the optimization performance.

Table 1: Overview of the function pool and the initialization areas.

Function	Optimum	Domain	Initialization
Ackley	$x_i = 0$	$[-32,32]^n$	$[16,32]^n$
Gen. Schwefel	$x_i = 420.9687$	$[-500,500]^n$	$[-500,-250]^n$
Griewank	$x_i = 0$	$[-600,600]^n$	$[300,600]^n$
Rastrigin	$x_i = 0$	$[-5.12,5.12]^n$	$[2.56,5.12]^n$
Rosenbrock	$x_i = 1$	$[-30,30]^n$	$[15,30]^n$
Schwefel	$x_i = 0$	$[-100,100]^n$	$[50,100]^n$
Sphere	$x_i = 0$	$[-100,100]^n$	$[50,100]^n$

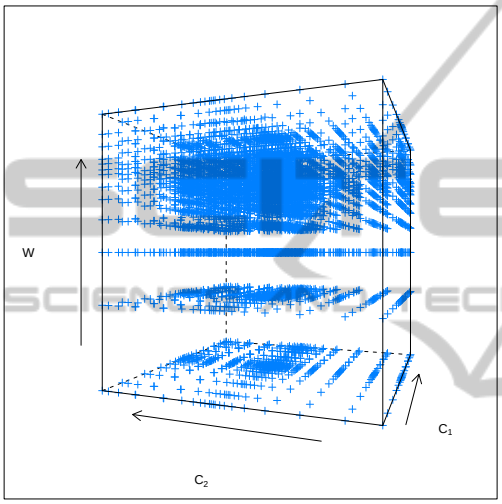

Figure 5: Parameter sets in the configuration space.

4.2 Finding the Best Parameter

In order to find the best parameter set for each function (see table 1), we start an extensive search with respect to the continuous values. We try to focus on real values with a precision of four decimal places. The standard parameter set for PSO is $\omega = (W, C_1, C_2)$ with $W = 0.72984$ and $C_1 = C_2 = 1.4962$. For the extensive examination of parameters we take into account the intervals $W \in [0,1]$ and $C_1, C_2 \in [0,2.5]$. We create a sequence between this interval values based on the standard value with a exponential factor of $(\frac{2}{1+\sqrt{5}})^x$ where x indicates the sequence number. We calculate 13 and 23 sequence values around the standard value and obtain a sequence of values between the intervals. Depending on the exponential factor the values close by the standard values have a lower distance to each other than the values closer to the borders of the interval. In figure 5 our configuration space of the extensive search is plotted. With all possible combinations of the single parameter values we examine $13 \times 23 \times 23 = 6877$ different parameter sets and test each of them for every function 90 times.

As described above, we analyze the data of the extensive search by comparing the results of each configuration's optimization process on a function. We choose the best parameter sets for every function with respect to the best performance. The best performance is determined by the best fitness value, the mean fitness value of all 90 optimization processes and the distance to the nine other best performances within this 90 processes. This is essential because a good solution and a high distance to the other run results let this one run be an outlier. Figure 6 shows an example of a sorted sequence of the mean value of all parameter sets we tested on the "Ackley"-function. We compare the results of the specific parameter sets with the standard parameter set of (Bratton and Kennedy, 2007) using our PSO implementation, and get a significantly better result (or the same if we found the global optimum) on gbest for all functions with the selected parameter sets. Table 2 shows our results for the specific parameters for the different functions (300000 evaluations + 30000 evaluations for feature computation; denoted as "specific"), the same parameter set subtracting one percent evaluations for the feature computation (to demonstrate if we used this one percent of computation time to extract the features, i.e., a total of 300000 evaluations; "specific*") and the comparison to the standard parameter set included in our code. Additionally, the comparison to the results of the original paper of Bratton and Kennedy is included in the table.

The extensive search shows that the best specific parameter sets for the functions Gen. Schwefel and Rastrigin is comparable. The same effect is also supported by the features of both objective functions. This denotes that both functions are assigned the same class in our classifier. All the specific parameter sets are the base of our classes for each function. With the identified classes and the computed set of features for each function we can train the classifier.

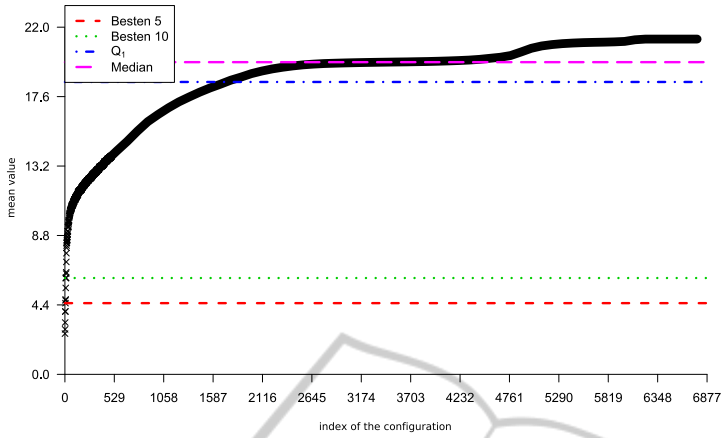

Figure 6: Sorted Set the Mean Value of all Parameterset Results.

Table 2: Comparison of the standard parameter set against the specific best parameter set; *denotes the optimization with 9900 iterations, i.e., 297000 function evaluations.

	Tests			Reference in (Bratton and Kennedy, 2007)		Best parameter set
Function	specific	specific*	standard	*gbest*	*lbest*	(W, C_1, C_2)
Ackley	**2.58**	2.62	18.34	17.6628	17.5891	(0.7893,0.3647, 2.3541)
Gen. Schwefel	**2154**	2155	3794	3508	3360	(0.7893,2.4098, 0.3647)
Griewank	**0.0135**	0.0135	0.0395	0.0308	***0.0009***	(0.6778, 2.1142, 1.3503)
Rastrigin	**6.12**	6.12	169.9	140.4876	144.8155	(0.7893,2.4098, 0.3647)
Rosenbrock	**0.851**	0.86	4.298	8.1579	12.6648	(0.7123,1.8782, 0.5902)
Schwefel	**0**	0	0	0	0.1259	more than one set
Sphere	**0**	0	0	0	0	more than one set

4.3 Learning and Classification

As classifier we use a C4.5 decision tree. In our implementation we use WEKA's J4.8 implementation (Witten and Frank, 2005). As learning input we compute 300 independent instances for each function. Each instance consists of 32 function features. The decision tree is created based upon the training data and evaluated by stratified 10-fold cross-validation (repeated 100 times). Based on the results of the extensive search we merge the classes for the objectives Gen. Schwefel and Rastrigin into one class. These functions share the same specific parameter set, i.e., the same parameter configuration performs best for both functions. Upon these six distinct classes we evaluate the model with cross validation. The mean accuracy of the 100 repetitions is 84.32% with a standard deviation of 0.29. Table 3 shows the confusion matrix of a sample classification. As it can be seen, there are 1769 of 2100 instances classified correctly (this means 15.76 percent of the instances are misclassified). The instances of the functions Ackley and Schwefel are classified correctly with an accuracy of 99.7 percent, that is only one instance of these classes is misclassified. The class for Gen. Schwefel and Rastrigin has an accuracy of 97.2 percent. The class Rosenbrock has a slightly lower accuracy, but still only 5.7 percent of its members are misclassified. The high number of incorrect instances is essentially due to the inability of the model to separate the functions Sphere and Griewank. The majority of the misclassified instances, 306 of 331, are instances of the Griewank or Sphere class that are classified as the other class.

4.4 Computing Effort

Computing the features is based on the evaluated fitness value of specific positions in the search space. We restrict this calculation to 3000 which means one percent of the whole optimization process in our setting. To be comparable to the benchmark of Bratton

Table 3: Confusion matrix of the cross validation.

	classified as						Accuracy	Precision
class	Ack.	Grie.	G.S./R.	Rosen.	Schwe.	Sphe.	percent	percent
Ackley	**299**	1					99.7	99.7
Griewank		**116**	1			183	38.7	48.1
Gen.Schwe./Rast.	1	1	**583**	13	2		97.2	97.2
Rosenbrock			15	**283**	1	1	94.3	95.3
Schwefel				1	**299**		99.7	99.0
Sphere		123	1			**176**	58.7	48.9

and Kennedy we run the optimization for the specific parameter configuration for 9900 iterations leading to only 297000 fitness computations. We compare our results of the optimization with 10000 iterations to the optimization with 9900 iterations and get quite the same results as shown in table 2 (specific vs. specific*). The comparison shows minor differences in the magnitude of one percent.

5 DISCUSSION AND FUTURE WORK

In this paper we describe an approach to training a classifier which uses function features in order to select a better parameter configuration for Particle Swarm Optimization. We show how we compute the features for specific functions and describe how we get the classes of parameter sets. We include the trained classifier and evaluate the parameter configuration against a Particle Swarm Optimization with standard configuration. Our experiments demonstrate that we are able to classify different functions on basis of a few fitness evaluations and get a parameter set which leads the PSO to a significantly better optimization performance in comparison to a standard parameter set. Statistical tests (t-Tests with $\alpha = 0.05$) indicate better results for the functions where the global optimum has not been found in both settings.

The next steps are to involve all possible configurations of the PSO for example the swarm size or the neighborhood topology. These parameters are not involved in our approach because we based this work on the benchmark approach of (Bratton and Kennedy, 2007). The behavior of the swarm changes significantly if another neighborhood is chosen. To increase the size of the swarm is another task we will focus in future. Depending on the swarm size different parameter sets leads to the best optimization process. An idea is to create an abstract class of parameter sets which include different sets of predefined swarm sizes.

In order to get more information about the performance of our approach it would be interesting to allocate a fixed percentage of the whole evaluations for feature computation (e.g., 1%). In this case it would be interesting to examine the quality of the result if not all feature or features of minor quality were computed.

Another extension is to define typical mathematical function types to integrate not only one function as class but a few functions combined under a similar type of functions to get a general set of parameters. This might lead to a better generalization for the learned classifier. The problem of this task is to find a general problem class which defines typical kinds of mathematical functions.

REFERENCES

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1 edition.

Bratton, D. and Kennedy, J. (2007). Defining a standard for particle swarm optimization. *Swarm Intelligence Symposium*, pages 120–127.

Clerc, M. and Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using part swarm theory. *Proceedings of the Sixth International Symposium on Micro Maschine and Human Science*, pages 39–43.

Hoos, H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Hutter, F., Hoos, H. H., and Stutzle, T. (2007). Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence, (AAAI '07)*, pages 1152–1157.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Proceedings of the 1995 IEEE International Conference on Neural Network (Perth, Australia)*, pages 1942–1948.

Leyton-Brown, K., Nudelman, E., and Shoham, Y. (2002). Learning the empirical hardness of optimization problems: The case of combinatorial auctions. *Principles and Practice of Constraint Programming (CP '02)*, pages 91–100.

Pant, M., Thangaraj, R., and Singh, V. P. (2007). Particle swarm optimization using gaussian inertia weight. In *International Conference on Conference on Computational Intelligence and Multimedia Applications*, volume 1, pages 97–102.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning. San Mateo, CA*. Morgan Kaufmann.

Shi, Y. and Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In *Proceedings of the 7th International Conference on Evolutionary Programming VII, (EP '98)*, pages 591–600, London, UK. Springer-Verlag.

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*. Wiley, Hoboken, NJ.

Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Inf. Process. Lett.*, 85(6):317–325.

van den Bergh, F. and Engelbrecht, A. (2002). A new locally convergent particle swarm optimiser. *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, 3:6 pp.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.