Design and Experimental Evaluation of Multiple Adaptation Layers in Self-optimizing Particle Swarm Optimization

Thomas Ritscher, Sabine Helwig, and Rolf Wanka
Department of Computer Science, University of Erlangen-Nuremberg, Germany
{thomas.ritscher@googlemail.com, sabine.helwig, rwanka}@informatik.uni-erlangen.de

Abstract—Particle swarm optimization (PSO) is a nature-inspired technique for solving continuous optimization problems. For a fixed optimization problem, the quality of the found solution depends significantly on the choice of the algorithmic PSO parameters such as the inertia weight and the acceleration coefficients. It is a challenging task to choose appropriate values for these parameters by hand or mathematically. In this paper, a novel self-optimizing particle swarm optimizer with multiple adaptation layers is introduced. In the new algorithm, adaptation takes place on both particle and subswarm level. The new idea of using virtual parameter swarms which hold modifiable parameter configurations each is introduced. The algorithmic PSO parameters can be mutated by using, for instance, well-known techniques from the field of evolutionary algorithms, in order to allow fine-granular parameter adaptation to the problem at hand. The new algorithm is experimentally evaluated, and compared to a standard PSO and the TRIBES algorithm. The experimental study shows that our new algorithm is highly competitive to previously suggested approaches.

I. INTRODUCTION

1) The problem: In the area of meta-heuristic optimization, there is a variety of nature-inspired approaches: Evolutionary Algorithms (EA), Genetic Algorithms (GA), Ant Colony Optimization (ACO), and Particle Swarm optimization (PSO), to name some popular of them. PSO has been introduced by Kennedy and Eberhart [KE95], [EK95]. Here, a swarm of $N$ individual particles cooperatively tries to minimize a given continuous fitness function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, i.e., to determine an input $\mathbf{x}_{\text{opt}} \in \mathbb{R}^n$ to $f$ such that $f(\mathbf{x}_{\text{opt}})$ is minimum. The swarm explores the $n$-dimensional space as follows: It works in iterations. In iteration $t$, every particle $i$ has a position $\mathbf{x}_{i,t}$, a velocity $\mathbf{v}_{i,t}$, and a neighborhood of particles. Particle $i$ knows additionally its currently best own solution $\mathbf{p}_{i,t}$, the private guide, and the currently best solution $\mathbf{l}_{i,t}$, the local guide, of its neighbors. The positions of the particles are updated according to the PSO movement equations. The new position of $i$ is computed as follows:

$$
\mathbf{v}_{i,t+1} = \omega \cdot \mathbf{v}_{i,t} + c_1 \cdot \mathbf{r}_{1,i,t} \odot (\mathbf{p}_{i,t} - \mathbf{x}_{i,t}) + c_2 \cdot \mathbf{r}_{2,i,t} \odot (\mathbf{l}_{i,t} - \mathbf{x}_{i,t})
$$

$$
\mathbf{x}_{i,t+1} = \mathbf{x}_{i,t} + \mathbf{v}_{i,t+1}
$$

$\odot$ denotes element-by-element vector multiplication, $\omega$ (the inertia weight), $c_1$, and $c_2$ (the acceleration coefficients) denote constants chosen by the user. They govern the relative impact of the velocity and of $\mathbf{p}_{i,t}$ and $\mathbf{l}_{i,t}$, and $\mathbf{r}_{1,i,t}$ and $\mathbf{r}_{2,i,t}$ are random vectors with components drawn uniformly at random from $[0, 1]$.

With respect to $f$, the choice of values for $N$, $\omega$, $c_1$, and $c_2$ heavily influences the exploitation and exploration properties and, hence, the success of the algorithm. Depending on $f$, it can be often observed that the swarm agrees too early to a solution though it is still far away from optimum. On the other hand, it can be often observed that the swarm still explores the space, rather than investigating the vicinity of positions already recognized as good. Therefore, it is desirable that the swarm adapts to $f$ during its execution, i.e., it adjusts (some or all of) the PSO parameters, namely the number of particles, inertia weight, acceleration coefficients, and neighborhood topology, to the current problem at hand. Adaptation can take place by iteratively running one or more parameter configurations for some time, evaluating the success of particles, subswarms, and parameter values, and changing the configurations applying some optimization methods.

2) Previous work: One of the most popular adaptive particle swarm optimizers is the so-called TRIBES algorithm presented by Clerc [Cle03], [Cle06a]. The algorithm mainly adapts the number of particles as well as their underlying neighborhood graph to the given optimization problem. The particle swarm is divided into several, fully-connected subswarms, the so-called tribes. The connectivity of the neighborhood graph is guaranteed by inter-subswarm communication links. For the adaptation process, a classification of both particles and tribes, e.g., in good and bad tribes, takes place based on their previous performance. According to these classifications, certain adaptation rules are applied. For instance, a new particle is generated by each bad tribe. Instead of the standard PSO movement equations, Clerc uses so-called pivot methods [Cle03], [Cle06a] in the TRIBES algorithm.

Xie et al. [XZY02] also introduce a classification scheme for particles. In their case, particles are either active or inactive. While active particles perform a standard movement, inactive particles are re-initialized.

In the Hierarchical Particle Swarm Optimizer presented by Janson and Middendorff [JM05], the swarm’s social network is adapted to the optimization process. A tree is used as neighborhood graph. The tree is periodically updated such that good particles move towards the root of the tree. In this way, they gain more impact on the other particles. Furthermore, different values for $\omega$ can be used by the particles on different
levels of the tree. As a (still problem-independent) adaptation, the branching degree is gradually decreased during the course of optimization.

In the adaptive Multi-Swarm PSO with Migration (MPSO) of Jordan et al. [JHW08], the particle swarm is divided into several subswarms that do not share information with each other. Each subswarm has its own configuration of the algorithmic PSO parameters. The overall goal is that the particle swarm adopts a configuration that is well-suited for the given problem. Therefore, fitness values and/or penalty points are assigned to each subswarm. Based on these values, particles migrate from one subswarm to another either periodically or if the penalty record of a subswarm reaches a specified limit.

Miranda and Fonseca [MF02] and Toscano-Pulido et al. [TPCSQ07] combine concepts of evolution strategies and genetic algorithms, respectively, with particle swarm optimization, to obtain adaptive PSO variants.

3) New results: We present a new self-optimizing PSO that pursues three adaptation approaches simultaneously in a combined way: (1) particle adaptation, where bad particles that became immobile are reinitialized; (2) sub-swarm adaptation, where good subswarms remove their worst particles and subswarms that are not good create new particles forming a new subswarm; and (3), as the main new contribution of this paper, parameter adaptation, where virtual parameter swarms consisting of particles from subswarms can adapt the inertia weight(s) and acceleration coefficients to $f$. Whereas methods similar to (1) and (2) have previously been investigated ([XZY02], [Cle06b] and [Cle03], [Cle06a]), our parameter adaptation (3) (virtual parameter swarms) and the way of combining the three methods is a new approach to adaptive PSO. We introduce new classification schemes, a new mutation mechanism that allows fine-grained parameter adaptation, the new concept of virtual parameter swarms, and a new simplified inter-subswarm communication strategy.

Our algorithm has been carefully implemented and experimentally evaluated. It has been tested on 20 established benchmark functions. These experimental results have been compared to the results obtained by standard PSO and to TRIBES mentioned above. It turns out that the new algorithm produces significantly better results on many of the tested benchmark functions.

4) Organization of paper: In the next section, we present a full description of our new self-optimizing PSO with multiple adaptation layers, and in Sec. III, we present a detailed experimental evaluation by comparing our adaptive PSO to standard PSO [BK07] and to the TRIBES [Cle03], [Cle06a] algorithm. We also report on further experiments conducted with variants of our algorithm.

II. PARTICLE SWARM OPTIMIZATION WITH MULTIPLE ADAPTATION LAYERS

In this section, the new algorithm will be described that uses various adaptation techniques. The goal is to realize a self-adapting PSO algorithm that is able to solve optimization problems without the necessity of parameters optimized (usually by hand) prior to the execution. In order to accomplish this task, the algorithm is designed by applying state-of-the-art adaptation techniques and an additional new parameter self-adaptation method. First we give an overview on the used techniques, then we describe them in detail.

A. General Concept

The general idea of our algorithm is to use essentially three adaptation techniques for the purpose of self-optimization. First there is the adaptation on particle level which is called particle adaptation. It is mainly inspired by Xie et al. [XZY02] and Clerc [Cle06b]. The second technique is derived from the concept of TRIBES (see Sec. I-2) and which is called subswarm adaptation in our algorithm. The third adaptation technique is the main new contribution of this paper. This parameter adaptation is the main reason for reducing the need for preinitialized parameters. Additionally, the parameter adaptation is intended to cope with the problem of particle stagnation. Hence, it reduces the waste of time by function evaluations on particles which do not move any more. Furthermore, this adaptation assigns a rating to every particle. This rating is used in the subswarm adaptation as well.

All adaptations are carried out after $T$ iterations. Best results were achieved experimentally with $T = N$, $N$ being the current total number of particles. In a single iteration, all particles execute one step according to the PSO movement equations.

Our swarm is partitioned into fully connected subswarms. In every subswarm, there is one particle called leader. It is one of the particles in the subswarm that led to the currently best solution of its subswarm. The subswarms are interconnected via the leaders. Additionally, every particle belongs to one of some virtual parameter swarms. The membership of a particle to a subswarm is independent from the membership to a virtual parameter swarm, and vice versa. A sample constellation of a full swarm with marked parameter swarm and subswarm membership is depicted in Fig. 1.

As suggested in other papers ([BK07], [XZY02]), our algorithm implements a linear decrease of “high search” activity. High search activity is often also called exploration and aims on finding new good solutions in areas not yet visited, while the “precise search” in the end is called exploitation and is intended to improve the solutions found so far. The parameter which controls the search behavior in our algorithm is called search and represents the probability to favor high search activities. search starts with the value 1 and ends with 0. Note that for a particle, the “personal best” always describes its best solution found so far, and the “local best” describes the best solution found in its neighborhood/subswarm.

In the following, we describe the three applied adaptation techniques in detail.

B. Particle Adaptation

A problem arising in particle swarm optimization is that in most variants a stagnation effect can be observed which means
that some or all particles stop after some number of iterations (see [Cle06b] for a detailed explanation). If this happens, function evaluations – which are assumed to be very expensive – waste time, and the overall performance goes down because these particles do not contribute further improvements. As suggested in the paper of Xie et al. [XZY02], these dead particles are therefore re-initialized. Further classifications of particles and the corresponding actions are done in order to improve the performance by supporting precise search close to good particle positions and by repositioning bad particles to find new solutions.

In more detail, each particle is classified according to the following scheme. A particle is called (1) dead if it did not move since the last iteration\(^1\), (2) very bad if it did not improve its personal best in this iteration and the one before, (3) bad if it did not improve its personal best in this iteration, (4) good if it improved its personal best in this iteration, and (5) very good if it improved the local best of its subswarm in this iteration.

With probability search, dead particles are randomly repositioned in the search space and they get a new, random velocity. With probability \(1 - \text{search}\), their current values are slightly altered in order to reactivate these particles. This is done by a Gaussian distributed random value with the old position as mean value and standard deviation \(c_1/10\). For the change in velocity, the mean value is the old velocity, and the standard deviation is \(\omega/10\). Due to these two possible actions and the decrease of the parameter search, high search is favored in the beginning, while in the end, there is much more exploitation caused by little changes in position and velocity. Furthermore, by proceeding in this way, some randomness comes into play by using \(c_1\) and \(\omega\) without introducing new parameters.

The other actions are as follows: Very bad particles are removed because they are not considered to be of any use for their subswarm any longer. Bad particles get a new, random velocity with probability search. With probability \(1 - \text{search}\), their position and velocity is set to the values of the currently best particle in the subswarm and then softly altered in the same way as the values of dead particles are changed. Good particles are not subject to any modifications because they seem to work fine, and very good particles are cloned because their close vicinities seem to be good and thus are worth to be exploited. A limit on the maximum number of changes in one adaptation round turned out to have no significant effect and was therefore omitted from our algorithm.

### C. Subswarm Adaptation

When our algorithms starts, all particles are evenly distributed among \(N/3\) subswarms, i.e., 3 particles per subswarm. This number is not chosen arbitrarily. It ensures that on the one hand there are real subswarms, not only single particles, and on the other hand there are in the beginning as many subswarms as possible in order to have a high search activity due to the loose connection between any particles of different subswarms in comparison to their direct inner-subswarm connections. Subswarms define a dynamic topology for particle communication. They are mainly inspired by Clerc’s TRIBES algorithm described in Sec. I-2. Depending on the current performance of a subswarm, different adaptations are processed which can generate new particles and subswarms as well as delete them. Clerc introduced the term information links for connections between particles, which we will use as well. All particles of a subswarm are fully connected by these information links. Information links work in both directions and define the topology used in the PSO movement equations. Every particle has an information link to any other particle in its neighborhood/subswarm. Additionally, a particle is also its own neighbor. Based on these links, there is a ranked Fully Informed Particle Swarm (FIPS) topology (proposed by Jordan et al. [JHW08]) in each subswarm. Additionally, each subswarm is connected to the other subswarms by so-called leaders ensuring that the graph is also always fully connected. A particle becomes the leader of a subswarm when it improves its subswarm’s local best and thus has the currently best personal best solution in the subswarm. All subswarm leaders form a neighborhood on their own in order to realize information exchange between subswarms. That means a leader is only influenced by all other leaders, not by its own subswarm. Thus an information link between a regular particle and a subswarm leader is not used in both directions, in contrast to links between two regular particles or between two leaders. A leader can only be removed if it is the last particle of its subswarm and if it is at the same time worse than all other leaders. The concept of leaders defines a small hierarchy. It is inspired by the Hierarchical PSO by Janson and Middendorf [JM05]. For the adaptation process, which again takes place every \(N\)-th iteration, every subswarm knows, from the particle adaptation, the number of very good \(N_{\text{very good}}\) and good \(N_{\text{good}}\) particles that belong to it.

\(^1\text{Due to floating point arithmetics this is measured by a threshold which defines the minimum movement.}\)
According to these numbers, the subswarm itself is ranked as **good** if \(N_{\text{very good}} + N_{\text{good}} > \frac{\text{subswarmsize}}{2} \cdot \text{search}\). Subswarm ranking is influenced by the \text{search} parameter in order to not have only good subswarms in the beginning and only **bad** swarms in the end. Typically, many particles will find improved positions in the beginning of the execution, and in the end only few particles will still find improved positions.

The adaptation rules are defined as follows: Like in TRIBES, a good subswarm removes its worst particle (the one with the worst personal best) as long as there are at least two particles in the subswarm. The last particle of the swarm (which then automatically is the leader) gets removed only, if all other leaders are better than this one. Otherwise, nothing happens. In this way, no particle that could still offer precious information is removed.

If a subswarm is not ranked **good**, it is called **bad** and generates a new particle. All of these in one adaptation step generated particles together form one new subswarm. Though this is also very similar to TRIBES, **no information links** are established between each leader of a generating subswarm and the generated particle explicitly, but implicitly through the communication links between subswarm leaders. This adaptation inserts new impulses through the randomly positioned new particles and in this way it can help to reactivate a stale constellation. Especially at the end of the execution, the number of particles can increase significantly, but not at a critical speed. On the one hand, this increase is problem dependent, on the other hand, no negative side effects have been noticed. In contrary, the experiments turned out that all effort to limit the number of particles by, for instance, always removing the worst subswarm and its particles in the end of each adaptation, decreased the performance.

### D. Parameter Adaptation

The third adaptation mechanism, the so-called **parameter adaptation**, is the main novelty in this paper. It is intended to provide an \text{PSO} algorithm where the parameters have not to be configured by hand. We employ a dynamic clustering approach. At the start of the algorithm, the particles are again evenly – but independently from the subswarm membership – distributed among a certain number of parameter swarms (we chose \(N/3\) parameter swarms, i.e., initially 3 particles per parameter swarm). A parameter swarm does not influence the neighborhood, it just is a virtual group of particles sharing the same set of movement parameters \(\omega, c_1, c_2\). Therefore, every parameter swarm has a different behavior regarding the inertia weight, the influence of its own personal best and their neighbors personal bests. A similar but static scheme was proposed by Jordan et al. [JHW08]. The variation of parameters adds robustness and, as an improvement over the multi-swarms by Jordan et al. [JHW08], the adaptation ensures that there is no need to provide the swarm with a special set of parameters. So we avoid that we have to adapt \text{PSO} to a specific type of problems by hand.

The parameters can be initialized completely at random or they can be set to pre-configured values, which leads to quicker found solutions. Even with pre-configured values, adaptation ensures more robustness and also optimizes the parameters with respect to the specific problem. As a basis for the adaptation, the parameter swarms are rated every \(N\)-th iteration (one round) as (1) **very good** if more than one particle in the parameter swarm improved its local best in this round or one particle did this more than once, (2) **good** if one particle improved its local best once this round, (3) **normal** if at least one particle improved its personal best in this round, and (4) **bad** if no particle improved its personal best in this round. Note that the local best is still the best solution found so far in a particle’s subswarm and the personal best is its own best solution so far.

The full detailed parameter adaptation algorithm is presented in Algorithm 1. The following functions are applied:

- **mutate()**: Change one of the three parameters \(c_1, c_2, \omega\) by selecting a random value from a Gaussian distribution with standard deviation 0.1 and the old value of the parameter as mean value. The parameter to be changed is chosen by a rotation of these three parameters.
- **transfer(to)**: Moves one randomly picked particle from one parameter swarm to the parameter swarm referenced by \(to\).
- **adaptParameters(from)**: Set the parameters of the current parameter swarm to the values of the parameter swarm from \(from\).
- **mix(with)**: Draw a random number \(R \in \{0, \min(N_1, N_2)\}\) with \(N_1\) and \(N_2\) being the number of particles of the two parameter swarms. Then execute alternately \(R\) transfer operations from the calling parameter swarm to the parameter swarm with and the same number of transfer operations in the other direction.
- **spread(to)**: \(to\) references a set of parameter swarms. All particles of the calling parameter swarm are transferred to the ones in this set in a way that every parameter swarm in the set gets the same number of particles. If this is not possible, the remaining particles are randomly distributed among the set.
- **reinitRandom()**: By calling this function, the parameter swarm discards its parameter values and randomly chooses a set of new ones. In order to do so, the parameters \(c_1\) and \(c_2\) are chosen by a Gaussian distribution with standard deviation 0.5 and a randomly picked value from the interval \([0, 2]\) as mean value. \(\omega\) is calculated in the same way except that the interval the mean value is taken from is \([0, 1]\).

After finishing the adaptations, every parameter swarm with more than \(\lceil N/2 \rceil\) particles (\(N\) being the number of total particles in the swarm) is divided into two parameter swarms of equal size until this criteria is met for all parameter swarms. This prevents all particles from being member of the same parameter swarm, which would actually disable the parameter adaptation. If a parameter swarm has no particles left (because it spread itself to other parameter swarms) it is deleted. The indices \(\text{rand}_{\{v, g, n, b\}}\) are values randomly drawn each time they occur, except for each pass of a \text{while}-statement they are constant. Every pass in a \text{while}-statement is also ended by the removal of the involved parameter swarms from the according queue meaning these parameter swarms will not be considered again in this adaptation. By removing parameter swarms,
Algorithm 1 Parameter swarm adaptation

Require: \(N_v, N_g, N_n, N_b\) = number of very good, good, normal, bad parameter swarms in queue

Require: \(ps_v[N_v], ps_g[N_g], ps_n[N_n], ps_b[N_b]\) = vector (queue) with all very good, good, normal, bad parameter swarms in queue

for \(i = 0\) to \(N_n - 1\) do
    \(ps_n[i]\).mutate()
end for

if \(N_v > 0\) then
    if \(N_b == 0\) then
        while \(N_n > 0\) and \(N_v > 0\) do
            \(ps_n[rand_n]\).transfer(\(ps_v[rand_v]\))
        end while
        while \(N_n > 0\) and \(N_g > 0\) do
            \(ps_n[rand_n]\).transfer(\(ps_g[rand_g]\))
        end while
        \{no further actions for the rest\}
    else
        while \(N_g > 0\) and \(N_v > 0\) do
            \(ps_v[rand_v]\).adaptParameters(\(ps_v[rand_v]\))
            \(ps_v[rand_v]\).mix(\(ps_v[rand_v]\))
        end while
        while \(N_g > 0\) and \(N_g > 0\) do
            \(ps_g[rand_g]\).transfer(\(ps_g[rand_g]\))
            \(ps_g[rand_g]\).mutate()
        end while
        while \(N_n > 0\) and \(N_n > 0\) do
            \(ps_n[rand_n]\).spread(\(ps_n\))
        end while
        while \(N_b > 0\) do
            \(ps_v[rand_b]\).reinitRandom()
        end if
    else
        if \(N_b == 0\) then
            while \(N_n > 0\) and \(N_g > 0\) do
                \(ps_g[rand_g]\).transfer(\(ps_g[rand_g]\))
            end while
            \{no further actions for the rest\}
        else
            while \(N_b > 0\) and \(N_g > 0\) do
                \(ps_g[rand_g]\).adaptParameters(\(ps_g[rand_g]\))
                \(ps_g[rand_g]\).mix(\(ps_g[rand_g]\))
                \(ps_g[rand_g]\).mutate()
            end while
            while \(N_b > 0\) and \(N_g > 0\) do
                \(ps_g[rand_g]\).transfer(\(ps_g[rand_g]\))
                \(ps_g[rand_g]\).mutate()
            end while
            while \(N_b > 0\) do
                \(ps_v[rand_v]\).reinitRandom()
            end while
        end if
    end if

end if

III. EXPERIMENTAL EVALUATION

The new algorithm was experimentally evaluated by using well-known benchmark problems. The experimental study had two goals: First, a comparison of the new adaptive algorithm to a standard particle swarm optimizer and the TRIBES algorithm is provided. Second, the impact of the newly introduced parameters was tested. The main results are presented at the end of this section.

The algorithm was implemented, tested and compared by using the following set of test functions\(^2\), well established in the area of PSO: Ackley, Griewank, Michalewicz, Rastrigin, Rosenbrock, Schwefel, and Sphere. Furthermore, the CEC 2005 Benchmark functions f1–f14 published by Suganathan et al. [SHL+05] were used. As this is quite a broad spectrum of test functions the results are barely specific to just a certain type of function. This is especially important because the adaptations are intended to provide a general solution and not just a problem-specific enhancement. All test runs were performed with 50 program runs over each test function and, if not stated otherwise, there were initially \(\text{round}(10 + 2 \cdot \sqrt{\text{dim}})\) particles (adopted from [Cle08]). After 100,000 function evaluations, the algorithm was stopped and the results were analyzed. The statistical relevance of the obtained results was assessed by performing one-sided Wilcoxon rank sum tests [Wil45] with significance level \(\alpha = 0.01\). If pre-initialization of standard values for \(c_1, c_2, \omega\) is used, these are set to \(1.49617, 1.49617, 0.72984\), resp., like in SPSO07 by Bratton and Kennedy [BK07]. Infeasible particles are set to the nearest search space boundary while their velocities are set to zero in the respective dimensions [Cle06a]. Additionally, velocity clamping is used, i.e., each particle’s velocity is limited to half the search space range in each dimension. Velocities are initialized to half the distance of a particle’s initial position and a random position [C+07].

A. SPSO and TRIBES

In order to assess the new algorithm’s performance, it was compared to the Standard PSO (SPSO07 by Bratton and Kennedy [BK07]) and to Clerc’s TRIBES algorithm. The SPSO was published to get a more up-to-date basis for comparison purposes than the original PSO algorithm by Kennedy and Eberhart [KE95]. It uses a static ring topology and a constant number of \(N = 50\) particles. Tab. 1 shows the comparison results for 5-dimensional and for 100-dimensional benchmark problems.

---

\(^2\)Function descriptions and initialization ranges can be found, for instance, in [BK07]. The initialization space for Schwefel was set to \([-250, 250]^n\), where \(n\) is the problem dimensionality.
problems. Tab. II shows the computed objective values on the actual benchmarks.

Both SPSO and our new algorithm show similar performance on the low-dimensional problems. However, in higher dimensions, significant performance improvements can be observed on most tested benchmarks. Besides the adaptation strategies, the different bound handling strategies (absorb vs. infinity – for further information regarding bound handling, see [HW07]) might play an important role. The main goal of this comparison is to show that the performance of the new algorithm is at least competitive to the performance of a state-of-the-art algorithm on widely-used benchmarks, and to give a basic relation of the resulting values.

The comparison with TRIBES (implemented according to [Cle03]) shows significant performance differences, especially when solving higher dimensional problems. Although utilizing the same bound handling strategy, TRIBES has many differences to the new algorithm, for instance, a modified moving strategy. That makes it difficult to explain the exact reason for the observed performance differences. Instead, these experiments should be seen as a comparison of ready-for-use algorithms. The test functions f1 and f7 could only be solved in 100 dimensions by the new algorithm. But also in low dimensions the performance of TRIBES is surpassed, which shows the benefit of the new adaptations compared with a different adaptive and parameterless algorithm. Nevertheless, it has to be mentioned that TRIBES is the only algorithm in this comparative study that did not use any empirically found parameters, as the new algorithm was used with pre-initialized values (same as SPSO uses) in this experiment.

In 100 dimensions, the new algorithm clearly outperformed the other two on Rosenbrock, Sphere, f2, f6, f7, f9, f10, f12 and f13. In lower dimensions there were still differences, but since most problems were solved quite good after 100,000 function evaluations in lower dimensions the interesting part are the high dimensions.

B. Pre-initialization

In this experiment, the performance of running parameter swarms with randomly chosen start parameters (rand) is compared with the proposed variant with pre-initialized values (init) and with a version without parameter adaptation at all (nopara again uses the static standard parameters like SPOS07 [BK07]). rand chooses random values in [0, 1] for \( \omega \) and in [0, 2] for \( c_1, c_2 \) which are then taken as mean value of Gaussian distributions with standard deviation 0.5 that provides the final value each. nopara is realized by generating only one parameter swarm with all particles in it and disabling parameter adaptation completely. The comparative results are presented in Tab. III.

There is an improvement from nopara to init meaning that the desired robustness is actually achieved through this adaptation. The rather bad performance of rand is explainable through the quite long time the algorithm has to run before real good parameters are found. This leads to the conclusion that in the current form the algorithm should only be used without parameter initialization if it runs for a long time.

C. Further Experiments

Here we discuss some of the further experiments we have conducted.

First the decreasing search parameter is discussed. Inspired by prior work [BK07], [XZY02], [JM05], a decreasing activity of “high search” – already introduced in Sec. II – was tested by changing adaptation rules. With the parameter search decreasing linearly from 1 to 0, the probability of randomly re-initializing a dead particle, the probability of a bad particle to get a new random velocity and the classification of subswarms are influenced in a way that prefers high search activity at the beginning and a high exploitation at the end of the algorithm’s execution. Though the improvement was not that big, there were clear improvements on the test functions Sphere and f3.

The next test was to determine the influence of the order of mutations and some variants. A mechanism called soft mutation in which every parameter is altered by a very small amount in each adaptation step as well as different mutation schemes with \( c_1, c_2 \) coupled and uncoupled and with standard deviation 0.1 and 0.02 each, did not produce any significant difference.

Another important issue is the length of the adaptation interval. It should not be chosen too short because then the effects of adaptations may not be visible yet. Otherwise, if they are too long, then adaptation is too slow. As a good compromise, the adaptation length was set to \( N \) (number of particles currently present in the whole swarm). \( N/2 \) and

---

**TABLE I**

Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B. Top: results on 5-dimensional benchmark problems. Bottom: results on 100-dimensional benchmark problems. The names canonically denote the algorithms.

<table>
<thead>
<tr>
<th></th>
<th>para5</th>
<th>spso5</th>
<th>tribes5</th>
</tr>
</thead>
<tbody>
<tr>
<td>para5</td>
<td>0</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>spso5</td>
<td>4</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>tribes5</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>para100</th>
<th>spso100</th>
<th>tribes100</th>
</tr>
</thead>
<tbody>
<tr>
<td>para100</td>
<td>0</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>spso100</td>
<td>3</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>tribes100</td>
<td>3</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

**TABLE III**

Summary of one-sided Wilcoxon rank sum test with significance level 0.01. For each algorithmic combination (A, B), this matrix shows how often A performed significantly better than B.

<table>
<thead>
<tr>
<th></th>
<th>nopara5</th>
<th>init5</th>
<th>rand5</th>
</tr>
</thead>
<tbody>
<tr>
<td>nopara5</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>init5</td>
<td>6</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>rand5</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>nopara100</th>
<th>init100</th>
<th>rand100</th>
</tr>
</thead>
<tbody>
<tr>
<td>nopara100</td>
<td>0</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>init100</td>
<td>5</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>rand100</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Parameter adaptation to the problem at hand.

The new algorithm was experimentally compared to a standard particle swarm optimizer and to the TRIBES algorithm of Clerc, by using 20 well-known benchmark problems from the literature. The experimental study showed that our new algorithm is highly competitive to previously suggested approaches on the tested benchmark set, producing significantly better results than the standard PSO and TRIBES on many of the tested benchmark functions.

Future work includes the adaptation of the non-continuous PSO parameters such as the particles’ bound handling strategy and their neighborhood topology.

TABLE II

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Average Objective Values and Standard Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ackley</td>
<td>3.970/c&lt;sub&gt;e&lt;/sub&gt;/0±7.1276e-07</td>
</tr>
<tr>
<td>Griewank</td>
<td>2.4248e-07±6.777e-09</td>
</tr>
<tr>
<td>Michalewicz</td>
<td>1.144e-08±4.6067e-09</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>3.0958±0.063799</td>
</tr>
<tr>
<td>para100</td>
<td>10.329±0.6326</td>
</tr>
<tr>
<td>tribes100</td>
<td>2.6405±6.04725</td>
</tr>
</tbody>
</table>

IV. CONCLUSION AND FUTURE WORK

Particle swarm optimization is a nature-inspired technique for solving continuous optimization problems. Its application usually includes the preliminary adjustment of algorithmic parameters such as the inertia weight ω and the acceleration coefficients c<sub>1</sub> and c<sub>2</sub> to the current problem at hand. The selection of the algorithmic parameters is a challenging task. In order to reduce the need of manual parameter adjustment, a novel particle swarm optimizer with multiple adaptation layers was introduced. In the new algorithm, adaptation takes place on both particle and subswarm level. Additionally, and as the main contribution of this paper, virtual parameter swarms, which hold modifiable parameter configurations each, were introduced. The algorithmic PSO parameters can be mutated by using, for instance, well-known techniques from the field of evolutionary algorithms, in order to allow fine-granular parameter adaptation to the problem at hand.

REFERENCES


