Adaptive Memetic Particle Swarm Optimization with Variable Local Search Pool Size

C. Voglis*
Department of Computer Science
University of Ioannina
GR-45110 Ioannina, Greece
voglis@cs.uoi.gr

P.E. Hadjidoukas
Computational Science
ETH Zurich
Zurich CH-8092, Switzerland
phadjido@mavt.ethz.ch

K.E. Parsopoulos
Department of Computer Science
University of Ioannina
GR-45110 Ioannina, Greece
kostasp@cs.uoi.gr

D.G. Papageorgiou
Department of Materials Science and Engineering
University of Ioannina
GR-45110 Ioannina, Greece
dpapageo@cc.uoi.gr

I.E. Lagaris
Department of Computer Science
University of Ioannina
GR-45110 Ioannina, Greece
lagaris@cs.uoi.gr

ABSTRACT

We propose an adaptive Memetic Particle Swarm Optimization algorithm where local search is selected from a pool of different algorithms. The choice of local search is based on a probabilistic strategy that uses a simple metric to score the efficiency of local search. Our study investigates whether the pool size affects the memetic algorithm's performance, as well as the possible benefit of using the adaptive strategy against a baseline static one. For this purpose, we employed the memetic algorithms framework provided in the recent MEMPSODE optimization software, and tested the proposed algorithms on the Benchmarking Black Box Optimization (BBOB 2012) test bed. The obtained results lead to a series of useful conclusions.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*global optimization*; G.4 [**Mathematical Software**]

Keywords

Global optimization, memetic algorithms, hybrid algorithms, black-box optimization, local search

1. INTRODUCTION

Over the past 20 years, Evolutionary Algorithms (EAs) and Swarm Intelligence (SI) approaches have been established as powerful optimization tools for solving global optimization problems [1, 2, 3]. Torn and Zilinskas [4] proposed that two major and competing goals govern the design of a global search strategy: *exploration* ensures that every part of the domain will be covered and *exploitation* concentrates the effort in a close neighborhood of the best detected positions.

Modern optimization algorithms achieve these two goals by combining a global and a local optimization component. Such hybrid schemes that are defined within the broad family of EAs, are called *memetic algorithms* (MAs), following [5] that defines *"a memetic algorithm as an evolutionary algorithm that includes one or more local search within its evolutionary cycle"*. Naturally, this definition can be extended to include SI methods such as Particle Swarm Optimization (PSO).

Preliminary MAs were defined by a single combination of a global and a local strategy. However, there is an increasing recent trend to consider more than one local search strategies during the search. For this purpose, a pool of local search methods is usually defined and the MA selects one among them to exploit a current best position. The selection can be performed in various ways by using scoring mechanisms and probability distributions [6, 5, 7, 8]. In relevant literature, these methods are also known as *hyperheuristic MAs*, *multi-memes*, and *meta-Lamarckian learning methods* and they were originally designed to accompany standard GA operators.

Using a pool of local search algorithms with a *clever* selection mechanism, an MA can be effective in various problem instances. Indeed, consider the case where the objective function is discontinuous or has large flat plateaus. A single local search algorithm that uses derivatives and line search, would probably fail to significantly improve a local best point. On the other hand, the same algorithm could achieve high convergence rate and accuracy when applied on a sufficiently smooth function.

MAs that adaptively select from a pool of local search algorithms, are usually called *Adaptive MAs* [6]. Algorithms of this category can be divided according to the adaptation type as *static*, *adaptive* and *self-adaptive*. An algorithm is called static when no form of feedback is considered during the search. On the other hand, it is called adaptive

*Corresponding author.

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

if feedback of the overall search governs the choice of local optimization algorithms. Self-adaptive approaches usually apply evolutionary operators to co-evolve the local searches.

Besides the aforementioned classification, the adaptation may be *qualitative* if each local optimization algorithm can be simply characterized as *good* or *bad*. On the other hand, it can be *quantitative* if the exact value of the feedback is important to score the local optimization algorithm. According to their adaptation level, MAs can be further characterized as *external* if a problem-specific knowledge from past experience of the practitioner is exploited, *local* when only a part of the historical trace of the algorithm is used to adapt the decision, and *global* when the complete historical knowledge of the run is used.

The present work extends the MAs proposed in [9] and implemented in the *MEMPSODE* optimization software [10]. These hybrid approaches are based on the Unified Particle Swarm Optimization (UPSO) [11, 12] algorithm, combined with local search algorithms provided by the *Merlin* optimization environment [13]. The purpose of our study was twofold. Firstly, we aimed at investigating the impact of the local search pool to the overall algorithm, even in the case of the simplest static scheme where all local search algorithms have equal probability of appearance. A similar study appears in [14] providing many interesting results, although for a limited number of test functions.

Secondly, we introduce a simple scoring and selection mechanism that adaptively selects an appropriate local search from the pool. Based on the categorization presented above, our variant belongs to the category of *quantitative* and *local* adaptive MAs. Extensive testing showed that the proposed scheme outperforms the simple static one regardless of the local search pool size, while increasing the pool size also increases the overall efficiency of the method.

The rest of the paper is organized as follows: in Section 2 we describe in detail our adaptive MA scheme. In Section 3, we present experimental results on a popular test set, and Section 4 exposes conclusive remarks and possible future research extensions.

2. PROPOSED APPROACH

The proposed algorithmic schemes are presented in the following section, along with the necessary background information. The new algorithm is based on [10, 9] with the extension of adding multiple local searches that are stochastically selected following either a uniform random distribution (*Static Random scheme*) or an adaptive one, based on local search scoring (*Adaptive Random scheme*).

2.1 Unified Particle Swarm Optimization

Let the n-dimensional continuous optimization problem:

$$\min_{x \in X \subset \mathbb{R}^n} f(x), \tag{1}$$

where the search space X is an orthogonal hyperbox defined as:

$$X \equiv [l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n] \ \subset \ \mathbb{R}^n.$$

Also, let the index set:

$$I = \{1, 2, \ldots, N\}.$$

A *swarm* of N *particles* is defined as a set of search points:

$$S = \{x_1, x_2, \ldots, x_N\},$$

where each particle is an n-dimensional vector:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in X, \qquad i \in I.$$

The particles move by assuming an adaptable position shift, called *velocity*, denoted as:

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top, \qquad i \in I,$$

and they store the best position they have found,

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in})^\top \in X, \qquad i \in I.$$

in memory. If t denotes the iteration counter, the particles' positions and velocities are updated at each iteration as follows [15]:

$$\begin{aligned} v_{ij}^{(t+1)} &= \chi \left[v_{ij}^{(t)} + c_1 \mathcal{R}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) \right. \\ &\quad \left. + c_2 \mathcal{R}_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right) \right], \qquad (2) \\ x_{ij}^{(t+1)} &= x_{ij}^{(t)} + v_{ij}^{(t+1)}, \qquad (3) \end{aligned}$$

where $i \in I$ and $j = 1, 2, \ldots, n$. This is the *constriction coefficient* variant of PSO, named after the parameter χ in Eq. (2), which is used to restrict the magnitude of the velocities and it was derived through the stability analysis of PSO [15]. The rest of the parameters are the positive constants c_1 and c_2, also called *cognitive* and *social* parameter, respectively; and $\mathcal{R}_1$, $\mathcal{R}_2$, which are random numbers that differ for each i and j, drawn from a uniform distribution in the range $[0, 1]$.

The parameter g_i controls the information-sharing between the i-th particle and the rest. A neighborhood of the i-th particle is defined in the form of a set of indices of other particles. This set is determined according to user-defined communication schemes, also called *neighborhood topologies*, among the particles. A very popular neighborhood topology is the *ring* where the particles assume a circular communication scheme with each one communicating only with the particles with neighboring indices. Thus, for the i-th particle, the ring topology defines neighborhoods of the form:

$$\mathcal{N}_i = \{i - m, \ldots, i - 1, i, i + 1, \ldots, i + m\}.$$

with the indices recycling at the end, i.e., index 1 follows exactly after N. The parameter m controls the neighborhood's size and it is often called *neighborhood radius*. Thus, the parameter g_i is defined as:

$$g_i = \arg \min_{j \in \mathcal{N}_i} f(p_j).$$

Obviously, the topologies influence the flow of information among the particles and, hence, may affect the algorithm's efficiency and effectiveness. The special case where $\mathcal{N}_i \equiv I$ for all $i \in I$, is called the *gbest* (global) PSO model, while all other cases with $\mathcal{N}_i \subset I$ define *lbest* (local) PSO models.

The best position of each particle is updated at each iteration as follows:

$$p_i^{(t+1)} = \begin{cases} x_i^{(t+1)}, & \text{if } f\left(x_i^{(t+1)}\right) < f\left(p_i^{(t)}\right), \\ p_i^{(t)}, & \text{otherwise.} \end{cases} \tag{4}$$

The stability analysis of Clerc and Kennedy [15] revealed a trade-off between PSO's parameters, in order to achieve satisfactory convergence properties. Based on this analysis,

Table 1: Memetic strategies of the MPSO algorithm.

Scheme	Point of local search application
Scheme 1	p_g (overall best position)
Scheme 2	Each p_i, $i \in I$, with fixed probability $\rho \in (0, 1]$
Scheme 3	p_g and some randomly selected p_i, $i \in I$

the following parameter values are frequently used as default choices:

$$\chi = 0.729, \quad c_1 = c_2 = 2.05,$$

although alternative successful setups have also been reported for common test problems [16].

The *Unified PSO* (UPSO) algorithm generalizes the original PSO model by combining lbest and gbest velocity updates. UPSO stem-med from the speculation (supported by experimental evidence) that harnessing search directions with different exploration / exploitation properties can produce more efficient schemes. Let:

$$\mathcal{G}_{ij}^{(t+1)} = \chi \left[v_{ij}^{(t)} + c_1 r_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left(p_{gj}^{(t)} - x_{ij}^{(t)} \right) \right],$$

$$\mathcal{L}_{ij}^{(t+1)} = \chi \left[v_{ij}^{(t)} + c_1 r_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right) \right],$$

denote the velocity update of x_i in the gbest and lbest PSO model, respectively, where g is the index of the overall best particle, i.e.:

$$g = \arg\min_{j \in I} f(p_j).$$

Then, UPSO's update equations are defined as [11, 12]:

$$\mathcal{U}_{ij}^{(t+1)} = u\, \mathcal{G}_{ij}^{(t+1)} + (1 - u)\, \mathcal{L}_{ij}^{(t+1)}, \tag{5}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + \mathcal{U}_{ij}^{(t+1)}, \tag{6}$$

where $i \in I$ and $j = 1, 2, \ldots, n$. The parameter $u \in [0, 1]$ is called *unification factor* and it balances the influence (tradeoff) of the gbest and lbest velocity update. Obviously, the lbest PSO model is retrieved from Eq. (5) for $u = 0$, while $u = 1$ stands for the gbest PSO model. All intermediate values in the range $(0, 1)$ produce combinations with diverse convergence properties.

2.2 Memetic Strategy

There is a number of parameters to determine when designing MAs. These parameters refer to essential features of the algorithm regarding the *point* and *frequency* of application of the local search. Although additional information on the objective function may offer some guidance, in the general case these parameters are empirically determined based on a trial-and-error procedure.

For example, the Memetic PSO (MPSO) proposed in [9] was based on the fundamental schemes, called *memetic strategies*, reported in Table 1. These schemes can be applied either at each iteration or whenever a specific number of consecutive iterations has been completed. Obviously, local search is applied on a best position only if it has not been previously selected for local search or it has changed from the last application of local search.

Alternative memetic strategies may be dictated from specific features of the objective function. In general, local search can provide crucial improvements of the detected solutions, yet increasing the computational cost of the algorithm. For this reason, its frequency of application shall be kept low. In practice, MPSO uses only a small fraction of the swarm as initial points for local search [17].

2.3 Pool of Local Search Algorithms

In MEMPSODE [10] the UPSO variant draws local optimization methods from the robust Merlin optimization environment [13]. Up to its current implementation, Merlin includes 11 local search algorithms and provides a rich variety of controlling options. In this work we chose a subset of 4 diverse local searches, and we also included a random search module via the plugin mechanism (see [13]).

Merlin also provides scripting capabilities that verify if a point is a local minimizer so that unnecessary local search applications can be avoided. In the remaining we briefly present the local optimization algorithms that constitute our pool. Henceforth, a local search approach will be denoted as LS, and we assume that it returns the approximation of the minimizer, its function value, and the required number of function evaluations.

2.3.1 BFGS

BFGS is a well-known quasi-Newton method with line search [18]. At the start of k-th iteration, a point $x^{(k)}$, the gradient (or a finite-difference approximation) $g^{(k)}$, and an approximation $B^{(k)}$ of the Hessian matrix, are available. The method continues by performing a line search [18] to a direction $s^{(k)}$ defined as $B^{(k)} s^{(k)} = -g^{(k)}$, and an update from $B^{(k)}$ to $B^{(k+1)}$, using the BFGS update formula [19]. A careful selection of line search, in addition to the powerful local properties of the Hessian approximation, results in one of the most robust and effective optimization algorithms. The line search algorithm uses a maximum of 30 function evaluations with parameters set at $c_1 = 0.0001$ and $c_2 = 0.9$.

2.3.2 SIMPLEX

This method belongs to the class of direct search methods for nonlinear optimization. It was designed by Nelder and Mead [20]. The algorithm is based on the concept of *simplex* (or polytope) in $\mathbb{R}^n$, which is a volume element defined by $(n + 1)$ vertices. The input of the algorithm is an initial simplex. The SIMPLEX algorithm then moves the initial simplex towards the minimum by adapting its geometry and finally shrinks it to a small volume element around the minimizer. The procedure is derivative-free and proceeds towards the minimum using a set of $n + 1$ points. Thus, it is expected to be tolerant to ill-conditioned cases. The transformation rules include *reflection*, *expansion*, and *contraction* of the initial simplex. The initial simplex was constructed so as to occupy 20% of the initial search space. The reflection factor was set to 1, the contraction factor was set to 0.5, the expansion factor to 2.0.

2.3.3 ROLL

This method belongs to the class of pattern search methods. It proceeds by taking proper steps along each coordinate direction, in turn. Then, the method performs an one-dimensional search on a properly formed direction, in order to tackle possible correlations among the variables. User inputs the initial point and a user-defined exploration factor which was set to 3.0 in our experiments. The one-dimensional search tolerance was set to 0.001 and uses a maximum of 30 function evaluations.

Algorithm 1: Pseudocode of MPSO.

Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; swarm size: N; unification factor: UF; probability for local search: ρ; Local search pool: $\mathcal{L} = \{LS^{(1)}, LS^{(2)}, \ldots, LS^{(k)}\}$; Phase period : $\mathcal{K}$; Adaptive flag: *adaptive*

Output: Best detected solution: x^*, $f(x^*)$.

```
// Initialization
for i = 1, 2, ..., N do
    Initialize x_i and u_i
    Set p_i ← x_i // Initialize best position
    f_i ← f(x_i) // Evaluate particle
    f_{p_i} ← f_i // Best position value
    act_i ← 0// By default all particles perform FEs
end
for i = 1, 2, ..., k do
    c^(i) ← 0;  S^(i) ← 0;  P^(i) ← 1/k// Initialize
end
// Main Iteration Loop
Set t ← 0    ls ← 0
while (termination criterion) do
    // Determine which particles will apply LS on their best position
    for i = 1, 2, ..., N do
        if rand() < ρ then
            act_i ← 1 // The i-th particles will perform LSs
        end
    end
    // Update Best Indices
    Calculate global best index g_1 and local best index g_2
    // Update Swarm/Population
    for i = 1, 2, ..., N do
        Calculate lbest velocity update, L_i, using g_2
        Calculate gbest velocity update, G_i, using g_1
        u_i ← UF L_i + (1 − UF) G_i // Unified PSO
        x_i = x_i + u_i // Update particle's position
    end
    // Update Best Positions/Individuals
    for i = 1, 2, ..., N do
        if f_i < f_{p_i} then
            p_i ← x_i
            f_{p_i} ← f_i
        end
    end
    // Evaluate Population or Apply Local search
    for i = 1, 2, ..., N do
        if act_i = 0 then
            f_i ← f(x_i) // Perform FE
        else
            j ← RouletteSelection(P)
            [p_i^+, f_{p_i}^+, f_evals] ← LS^(j)(p_i) // Perform LS
            if adaptive = 1 then
                score^(j) ← (|f_{p_i} − f_{p_i}^+| / |f_{p_i}^+|) / f_evals
                ls ← ls + 1;
                if train = 1 and mod(ls, K) = 0 then
                    // Entering adaptive phase
                    train ← 0
                else if train = 0 and mod(ls, 3 * K) = 0 then
                    // Entering training phase
                    train ← 1
                    // Average scores and probabilities are reset
                    for κ = 1, 2, ..., k do
                        c^(κ) ← 0;  S^(κ) ← 0;  P^(κ) ← 1/k
                    end
                end
                UpdateProb(j, score, S, P, train)
            end
        end
    end
end
```

2.3.4 AUTO

AUTO is a hyper-local search procedure that tries to automatically select the best LS algorithm. The methods BFGS, ROLL, SIMPLEX and TRUST are invoked one after the other. For each one, a rate is calculated by dividing the relative achieved reduction of the function's value by the number of function calls spent. The method with the highest rate is then invoked again and the procedure is repeated. If all rates assume vanishing values, then all the method tolerances are set to zero and the methods are applied in the following order: `ROLL, TRUST, BFGS, SIMPLEX`.

2.3.5 RAND

We added a random search component to our arsenal of local searches in order to tackle highly discontinuous problems with low or no structure. In every iteration a random step is taken from a uniform distribution inside a predefined box. If the step leads to a smaller function value, it is accepted. Subsequent rejections reduce the sampling box size, hence leading to smaller random steps. In our case, we start with random step inside a box of magnitude up to 5.0, reduced by 10% every 50 consecutive rejections. The method was integrated with the Merlin environment through the plugin mechanism.

2.4 Adaptive selection strategy

As previously mentioned, we introduce a new adaptive selection strategy based on local search improvement scores. The core of our selection algorithm is *roulette selection* [6, 14] where the probabilities can either be static and equal (*Static Random scheme*) or dynamically change during the algorithm's execution (*Adaptive Random scheme*). The probabilities are used in global context, in the sense that they are identical for all particles. Alternatively, each particle could individually update the probabilities for each local search.

Roulette selection seems a natural choice since it ensures that the number of applications of a specific local search is stochastic and proportional to a probability. In this context, we assume a local search pool:

$$\mathcal{L} = \{LS^{(1)}, LS^{(2)}, \ldots, LS^{(k)}\},$$

where local search $LS^{(i)}$ is applied with probability $\mathcal{P}^{(i)}$, $i = 1, \ldots, k$. The probability is based on a scoring mechanism that is updated after the application of the local search. Let us assume that, at a specific point of the global search, each local search is applied $c^{(i)}$ time, $i = 1, \ldots, k$. In the $j-$th application of the local search $LS^{(i)}$, we define the $score_j^{(i)}$ as:

$$score_j^{(i)} = \frac{\frac{\left|\tilde{f}_j^{(i)} - \hat{f}_j^{(i)}\right|}{\left|\hat{f}_j^{(i)}\right|}}{\#fevals_j^{(i)}}, \quad j = 1, \ldots, c^{(i)}, \quad i = 1, \ldots, k,$$

where $\tilde{f}_j^{(i)}$ is the objective function value before the application of $LS^{(i)}$, $\hat{f}_j^{(i)}$ is the approximation of the local minimum and $\#fevals_j^{(i)}$ is the number of function evaluations spent.

Then, the *average score*, $\mathcal{S}^{(i)}$, is defined as:

$$\mathcal{S}^{(i)} = \frac{\sum_{j=1}^{c^{(i)}} score_j^{(i)}}{c^{(i)}}, \quad i = 1, \ldots, k,$$

Procedure UpdateProb(j, score, $\mathcal{S}, \mathcal{P}$, train)

Input: Selected index: i ; Score for the selected: $score^{(i)}$; Score array: $\mathcal{S}$, Probability array: $\mathcal{P}$, Phase flag: *train*

Output: New score: $\mathcal{S}$, New probability: $\mathcal{P}$; New count: c

$\mathcal{S}^{(i)} \leftarrow \frac{c^{(i)}\mathcal{S}^{(i)} + \text{score}}{c^{(i)}+1}$ `// Update mean` $\mathcal{S}^{(i)}$

$c^{(i)} \leftarrow c^{(i)} + 1$ `// Update count` $c^{(i)}$

`// If we are not in training phase update the probabilities`

if train $= 0$ **then**

  for $\kappa = 1, 2, \ldots, k$ **do**

    $\mathcal{P}^{(\kappa)} \leftarrow \frac{\mathcal{S}^{(\kappa)}}{\sum^k \mathcal{S}^{(\kappa)}}$

  end

end

and the probability $\mathcal{P}^{(i)}$ for $LS^{(i)}$ is calculated as:

$$\mathcal{P}^{(i)} = \frac{\mathcal{S}^{(i)}}{\sum_{i=1}^{k} \mathcal{S}^{(i)}}, \quad i = 1, \ldots, k,$$

The above probability is assigned during the adaptive random scheme.

In the case of the static random scheme we have:

$$\mathcal{P}^{(i)} = \frac{1}{k}, \quad i = 1, \ldots, k.$$

From the probability calculation in the adaptive case, we deduce the following:

(a) The average score for a local search $LS^{(i)}$ is formed using information from the first iteration up to the current. This may be restrictive since not all search areas of an objective function have the same morphology.

(b) If a local search does not perform well in early stages, then it may be assigned a low score and probability. In this case we may prematurely cutoff a local search from the pool without properly assessing its behaviour.

In order to address these deficiencies, we decided to split the adaptive random process into two phases, which are interchangeably repeated. During the first phase, called *training phase*, the algorithm collects information (average score) for the local searches and uses roulette selection with static and equal probabilities. After the training phase comes the *adaptive phase*, where the probabilities are adapted to the average score and also updated after every local search.

In the current scheme, the training phase takes place for $\mathcal{K}$ local searches and the adaptive phase for an integer multiple of $\mathcal{K}$. When the adaptive phase ends, all counters are reset, probabilities are equalized, and information is gathered from the start during a fresh new training phase. With this two-phase scheme, we manage to include a *short memory* in our adaptation, since very old information is now excluded. In addition, all local searches are given a fair participation share to collect their statistics.

The complete algorithmic scheme of our memetic algorithm is presented in Algorithm 1. The adaptation of the probabilities is presented in Procedure UpdateProb.

3. EXPERIMENTAL RESULTS

We considered both static and adaptive MAs, taking advantage of the Merlin's optimization software local minimizers. For every instance of our MA, we had to decide between static and adaptive, as well as regarding the local searches that comprised the pool. We henceforth denote the BFGS as **B**, Simplex as **S**, Roll as **R**, AUTO as **A**, and Random as **N**. A local search pool is then defined by its constituents. For example, **BRA** stands for a search pool containing the BFGS, Roll, and AUTO method. Since we limit our search to search pools of size 5, we end up with

$$\binom{5}{1} + \binom{5}{2} + \binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 31,$$

distinct MA instances for the static case and another 31 for the adaptive case.

Both strategies were applied on the Black Box Optimization Benchmark (BBOB) 2012 test set, which consists of 24 functions of six different dimensions, namely $n = 2$, 3, 5, 10, 20, 40. Each problem instance was tested 15 independent times until either its target value was reached with accuracy of 10^{-8} or a maximum number of $5 \times 10^5 \times n$ function evaluations was reached. If the solution is reached within the predefined accuracy, the experiment is marked as *success*, otherwise as *failure*. Results are then grouped by dimension.

3.1 Performance Measures

In order to rank the instances of our MAs, we used two measures. The first measure is the *Expected Running Time* (ERT) [21], which is defined as the expected number of function evaluations to reach a target function value for the first time. The second measure is the *success rate*, i.e., the percentage of problems solved with precision 10^{-8} out of the total number of problems. The results are first calculated per function (15 trials/function) and then summed per dimension ($15 \times 24 = 360$ trials/dimension). This grouping is necessary for presentation purposes, due to space limitations.

All 31 instances of various pool sizes, are then ranked according to the achieved ERT and success rate. An instance is ranked in place r with respect to ERT, if it is better (smaller) than r other instances. In a similar manner, an instance is ranked in place r with respect to its success rate, if it is better (larger) than r other instances.

Tables 2, 3, and 4, expose the ERTs and success rates (columns "*ERT*" and "*Perc.*", respectively). Also, the rankings (column "*Rnk*") per dimension and instance of the MA for the static random scheme, are reported. Similarly, Tables 5, 6, and 7, report the rankings for the adaptive random scheme. The anticipated result implies that higher ranks shall be associated with larger pool sizes.

3.2 Impact of Pool Size

One primary target of the present work, was to study the impact of the local search algorithms' pool size to the MAs performance. In order to achieve this goal, for each MA instance we summed its rankings for the different dimensions, and calculated its *overall ranking*, both for ERT and success rate.

In Figs. 1 and 2, we present bargraphs of the overall ranking for the ERT metric. The bars are also grouped and

	2-D				3-D			
	ERT	Rnk	Perc.	Rnk	ERT	Rnk	Perc.	Rnk
B	17234.6	5	0.99	0	58883.5	5	0.99	10
R	33672.7	0	1.00	9	118739	0	0.97	0
S	7886.67	27	0.99	0	38366.2	18	0.99	12
A	12901.2	14	0.99	0	63104.3	4	0.99	4
N	18392.1	4	0.99	0	47209.4	12	1.00	13
BR	21902.8	1	0.99	0	78300.5	1	0.98	2
BS	9418.76	21	0.99	0	29149	27	1.00	13
BA	15261.2	9	1.00	9	74676.3	2	0.98	1
BN	10591.6	17	0.99	0	34527.8	24	1.00	13
RS	15983.8	7	1.00	9	56445.8	6	0.99	4
RA	18437.4	3	1.00	9	55119	9	0.99	10
RN	19786	2	1.00	9	69912.7	3	1.00	13
SA	13556.5	12	1.00	9	51387.2	10	0.98	2
SN	9738.48	19	0.99	0	28783.4	28	1.00	13
AN	9135.57	24	1.00	9	35004	21	1.00	23
BRS	11896.3	16	1.00	9	40378.7	15	1.00	13
BRA	16562.2	6	1.00	9	56120.6	7	0.99	7
BRN	12782.2	15	1.00	9	55572.5	8	0.99	7
BSA	15323.3	8	1.00	9	46474.1	13	0.99	4
BSN	9342.82	23	0.99	0	29997.3	26	1.00	13
BAN	7713.18	28	1.00	9	34977.2	22	1.00	23
RSA	10392	18	1.00	9	39214.2	17	1.00	13
RNA	13005.4	13	1.00	9	36116.2	20	1.00	23
RNS	14234.1	11	1.00	9	43137.2	14	1.00	13
SAN	6339.37	30	1.00	9	24750.9	29	1.00	23
BRSA	14272.4	10	1.00	9	47384.3	11	0.99	7
BRSN	9369.96	22	1.00	9	34621.3	23	1.00	13
BRNA	9421.92	20	1.00	9	31857.7	25	1.00	23
BSAN	7609.32	29	1.00	9	19154.3	30	1.00	23
RSAN	7946.71	26	1.00	9	37558.5	19	1.00	23
BRSAN	8401.99	25	1.00	9	39635.6	16	1.00	23

Table 2: Static random (2D-3D)

	5-D				10-D			
	ERT	Rnk	Perc.	Rnk	ERT	Rnk	Perc.	Rnk
B	439302	1	0.89	1	2.19E+06	9	0.72	6
R	543979	0	0.87	0	2.37E+06	3	0.73	7
S	241755	25	0.96	20	2.20E+06	8	0.73	13
A	297670	7	0.94	5	2.21E+06	7	0.73	7
N	359369	4	0.94	4	2.90E+06	0	0.68	0
BR	424510	2	0.91	2	2.00E+06	29	0.74	25
BS	193982	29	0.97	29	2.00E+06	27	0.75	28
BA	267565	15	0.95	12	2.05E+06	25	0.74	20
BN	263960	16	0.96	27	2.32E+06	4	0.71	2
RS	268297	14	0.95	12	2.13E+06	17	0.74	17
RA	303987	6	0.95	10	2.08E+06	21	0.74	20
RN	414635	3	0.93	3	2.52E+06	1	0.71	2
SA	284650	10	0.94	7	2.00E+06	30	0.76	30
SN	236075	27	0.96	20	2.39E+06	2	0.71	1
AN	275795	12	0.95	10	2.10E+06	19	0.74	20
BRS	247268	24	0.96	25	2.05E+06	24	0.74	25
BRA	288568	9	0.95	16	2.04E+06	26	0.75	27
BRN	306707	5	0.94	7	2.16E+06	15	0.73	13
BSA	260685	19	0.95	16	2.07E+06	23	0.74	20
BSN	193238	30	0.97	30	2.11E+06	18	0.74	20
BAN	236523	26	0.96	25	2.24E+06	6	0.72	4
RSA	255395	21	0.96	20	2.08E+06	22	0.74	17
RNA	290491	8	0.94	7	2.14E+06	16	0.73	11
RNS	271492	13	0.95	16	2.28E+06	5	0.72	4
SAN	258621	20	0.95	16	2.19E+06	10	0.73	13
BRSA	251199	22	0.96	20	2.00E+06	28	0.75	28
BRSN	263464	17	0.95	12	2.17E+06	12	0.73	11
BRNA	281258	11	0.94	5	2.17E+06	13	0.74	16
BSAN	218502	28	0.96	27	2.17E+06	14	0.73	7
RSAN	261205	18	0.95	12	2.10E+06	20	0.74	17
BRSAN	248934	23	0.96	24	2.19E+06	11	0.73	10

Table 3: Static random (5D-10D)

	20-D				40-D			
	ERT	Rnk	Perc.	Rnk	ERT	Rnk	Perc.	Rnk
B	1.15E+07	6	0.48	4	3.26E+07	9	0.39	9
R	1.18E+07	4	0.49	5	49459579	3	0.30	1
S	1.25E+07	3	0.48	3	5.01E+07	2	0.30	3
A	8.03E+06	21	0.60	26	3.35E+07	7	0.38	6
N	2.04E+07	0	0.35	0	6.36E+07	0	0.25	0
BR	9.69E+06	10	0.52	8	2.68E+07	20	0.45	20
BS	9.55E+06	11	0.55	12	2.87E+07	19	0.43	18
BA	7.58E+06	27	0.60	26	2.45E+07	26	0.47	26
BN	1.13E+07	7	0.49	5	3.18E+07	13	0.40	12
RS	1.07E+07	8	0.52	8	3.66E+07	5	0.37	5
RA	7.91E+06	24	0.59	20	3.31E+07	8	0.39	8
RN	1.31E+07	2	0.46	2	3.82E+07	4	0.36	4
SA	7.95E+06	23	0.60	23	3.01E+07	15	0.42	16
SN	1.43E+07	1	0.44	1	5.09E+07	1	0.30	1
AN	8.15E+06	19	0.59	18	2.92E+07	17	0.43	18
BRS	8.64E+06	14	0.57	14	2.92E+07	16	0.42	15
BRA	8.00E+06	22	0.58	16	2.62E+07	21	0.45	20
BRN	9.50E+06	12	0.53	11	2.60E+07	23	0.46	22
BSA	7.53E+06	29	0.60	23	2.30E+07	30	0.49	29
BSN	1.02E+07	9	0.52	8	3.23E+07	10	0.40	10
BAN	8.20E+06	18	0.58	16	2.37E+07	28	0.48	28
RSA	8.08E+06	20	0.60	23	2.89E+07	18	0.43	17
RNA	8.38E+06	15	0.59	18	3.08E+07	14	0.41	14
RNS	1.16E+07	5	0.50	7	3.52E+07	6	0.38	6
SAN	7.81E+06	26	0.61	28	3.21E+07	11	0.40	10
BRSA	7.34E+06	30	0.61	30	2.51E+07	25	0.46	25
BRSN	8.91E+06	13	0.56	13	2.61E+07	22	0.46	22
BRNA	8.29E+06	16	0.58	15	2.53E+07	24	0.46	24
BSAN	7.84E+06	25	0.60	21	2.33E+07	29	0.49	29
RSAN	8.21E+06	17	0.60	21	3.19E+07	12	0.40	12
BRSAN	7.57E+06	28	0.61	28	2.41E+07	27	0.48	27

Table 4: Static random (20D-40D)

	2-D				3-D			
	ERT	Rnk	Perc.	Rnk	ERT	Rnk	Perc.	Rnk
B	17234.6	2	0.99	0	58883.5	5	0.99	5
R	33672.7	0	1.00	10	118739	0	0.97	0
S	7886.67	26	0.99	0	38366.2	15	0.99	7
A	12901.2	9	0.99	0	63104.3	4	0.99	3
N	18392.1	1	0.99	0	47209.4	8	1.00	10
BR	15142.4	5	0.99	0	69040.1	3	0.98	1
BS	11360.3	13	0.99	0	30275.8	22	1.00	10
BA	13281	7	1.00	10	79851.1	1	0.98	1
BN	10174.8	15	0.99	0	30264.3	23	1.00	10
RS	8370.56	22	1.00	10	36949.2	18	0.99	7
RA	16891.8	3	0.99	0	53754	6	0.99	3
RN	16628.4	4	1.00	10	71724.6	2	1.00	10
SA	7920.96	25	1.00	10	33665.7	19	1.00	22
SN	8807.72	21	0.99	0	26551.1	27	1.00	10
AN	10044.8	16	1.00	10	43025.8	11	1.00	22
BRS	10039	17	1.00	10	21891.8	28	1.00	22
BRA	13774.7	6	1.00	10	51796.7	7	0.99	5
BRN	12041.3	10	1.00	10	41782.8	12	1.00	10
BSA	11480.8	12	1.00	10	39016.1	14	0.99	7
BSN	9269.34	19	0.99	0	21463	29	1.00	10
BAN	6860.97	30	1.00	10	32457.4	20	1.00	10
RSA	10650	14	1.00	10	40177.9	13	1.00	10
RNA	13218.6	8	1.00	10	43182.8	10	1.00	22
RNS	11726.1	11	1.00	10	37378.6	16	1.00	10
SAN	7417.73	28	1.00	10	19911.3	30	1.00	22
BRSA	8218.18	24	1.00	10	43856	9	1.00	10
BRSN	9488.68	18	1.00	10	31387.7	21	1.00	10
BRNA	8995.34	20	1.00	10	37078.9	17	1.00	22
BSAN	6886.24	29	1.00	10	29498.5	25	1.00	22
RSAN	7669.77	27	1.00	10	28681	26	1.00	22
BRSAN	8351.6	23	1.00	10	29737.7	24	1.00	22

Table 5: Adaptive random (2D-3D)

assume different color per pool size. A first reading of the graphs offers an apparent ascertainment: on average, increasing the local search pool size implies an increase of the MAs performance. This is confirmed also in the adaptive case, as depicted in Fig. 2 where the BRSAN instance (all available local searches) scores the best rank.

The same results are confirmed with respect to the success rate in Figs. 3 and 4. Again, the BRSAN instance scores the best rank. Preliminary experiments with larger pool sizes ($|\mathcal{L}| = 6$ and $|\mathcal{L}| = 9$) have shown even better performance, especially in higher dimensions.

3.3 Adaptive vs Static Scheme

The secondary target of our study, was the comparison of the adaptive scheme against the static one. Excluding the 5 instances with only one local search algorithm, there were 26 (instances) $\times$ 6 dimensions = 156 distinct comparisons for all the reported experiments.

Regarding ERT, the adaptive scheme outperforms the static one in 100 out of 156 cases (denoted as "wins"). For the success rate, the adaptive scheme outperforms the static one on 63 instances, and scores exactly the same with the static on another 52 instances. This can be easily inferred by comparing the average lines in Figs. 1, 2, and Figs. 3, 4. Especially for pool size equal to 4 and 5, the performance gain is obvious.

4. CONCLUSIONS

In the present paper, we considered an adaptive PSO-based MA, and studied its behavior under various combinations of local search algorithms. As a first outcome of our study, the results support the claim that the efficiency of the MA is increasing with the pool size. This is apparent in the reported results but also verified in preliminary results of larger pool sizes (6 or 9).

This outcome is aligned with and supports the general feeling that *many local searches increase the search diver-*

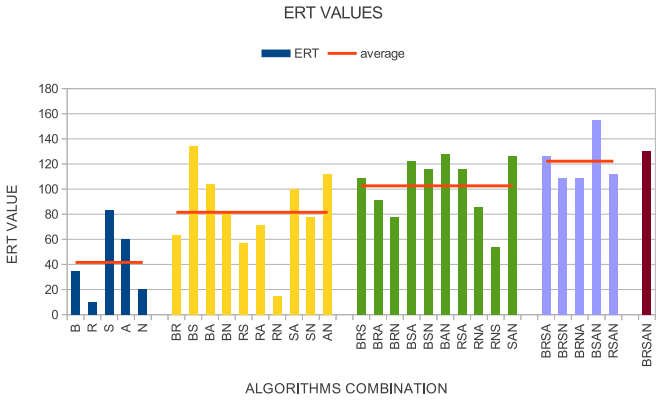

Figure 1: Static Random ERT score.

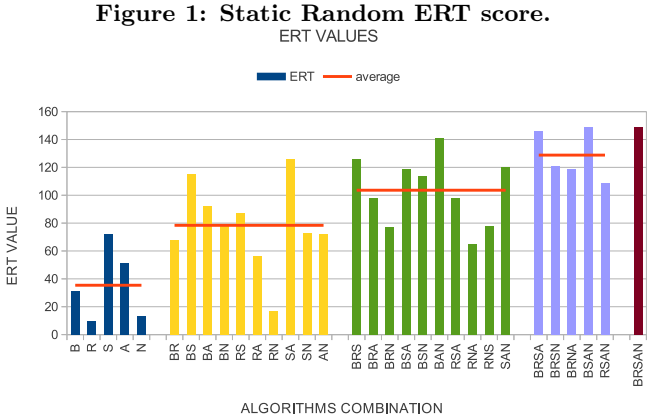

Figure 2: Adaptive Random ERT score.

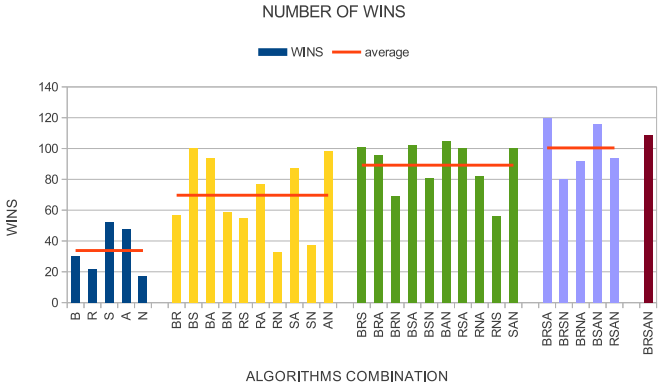

Figure 3: Static Random success percentage score.

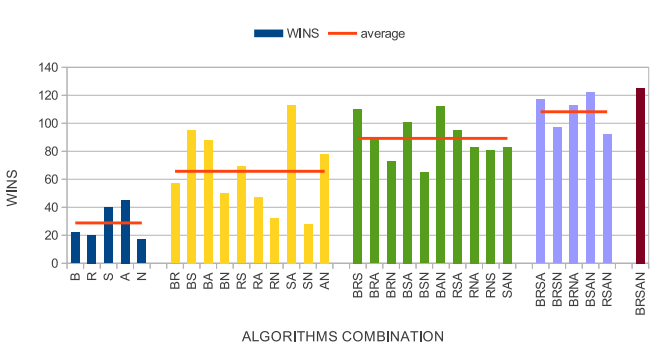

Figure 4: Adaptive Random success rate score.

	5-D				10-D			
	ERT	Rnk	Perc.	Rnk	ERT	Rnk	Perc.	Rnk
B	439302	1	0.89	1	2.19E+06	9	0.72	3
R	543979	0	0.87	0	2.37E+06	3	0.73	4
S	241755	20	0.96	17	2.20E+06	7	0.73	11
A	297670	6	0.94	8	2.21E+06	6	0.73	4
N	359369	4	0.94	7	2.90E+06	0	0.68	0
BR	391152	3	0.91	2	1.99E+06	24	0.75	21
BS	158731	30	0.97	30	2.00E+06	22	0.76	27
BA	252067	15	0.96	17	2.06E+06	17	0.74	16
BN	258361	14	0.96	17	2.18E+06	10	0.73	8
RS	231336	23	0.96	24	2.18E+06	11	0.74	14
RA	288191	10	0.94	9	2.17E+06	12	0.73	10
RN	406288	2	0.92	3	2.44E+06	1	0.72	2
SA	214927	28	0.97	29	2.04E+06	19	0.75	20
SN	242007	19	0.95	11	2.28E+06	4	0.73	4
AN	280515	11	0.95	11	2.38E+06	2	0.70	1
BRS	216680	27	0.97	28	2.00E+06	21	0.75	21
BRA	316240	5	0.93	4	1.94E+06	29	0.76	27
BRN	290660	9	0.95	11	2.11E+06	15	0.74	13
BSA	245343	17	0.96	17	2.00E+06	23	0.74	18
BSN	227881	25	0.95	15	2.06E+06	18	0.74	18
BAN	245487	16	0.96	17	1.99E+06	26	0.76	25
RSA	295044	8	0.94	5	1.98E+06	27	0.76	30
RNA	296843	7	0.94	5	2.20E+06	8	0.74	14
RNS	236878	22	0.96	24	2.09E+06	16	0.75	23
SAN	245265	18	0.94	9	2.23E+06	5	0.73	4
BRSA	230483	24	0.96	17	1.92E+06	30	0.76	25
BRSN	206475	29	0.96	27	2.02E+06	20	0.74	16
BRNA	260272	13	0.95	15	1.99E+06	25	0.75	23
BSAN	227108	26	0.96	24	2.11E+06	14	0.73	11
RSAN	264623	12	0.96	17	2.16E+06	13	0.73	8
BRSAN	237495	21	0.95	11	1.95E+06	28	0.76	29

Table 6: Adaptive random (5D-10D)

	20-D				40-D			
	ERT	Rnk	Perc.	Rnk	ERT	Rnk	Perc.	Rnk
B	1.15E+07	5	0.48	4	3.26E+07	9	0.39	9
R	1.18E+07	4	0.49	5	49459579	3	0.30	1
S	1.25E+07	2	0.48	3	5.01E+07	2	0.30	2
A	8.03E+06	19	0.60	24	3.35E+07	7	0.38	6
N	2.04E+07	0	0.35	0	6.36E+07	0	0.25	0
BR	9.23E+06	11	0.54	11	2.68E+07	22	0.44	22
BS	9.15E+06	12	0.55	12	2.96E+07	16	0.43	16
BA	7.68E+06	25	0.60	19	2.44E+07	27	0.47	25
BN	1.13E+07	6	0.49	5	3.19E+07	10	0.40	10
RS	1.03E+07	9	0.53	10	3.84E+07	4	0.36	4
RA	8.06E+06	17	0.59	17	3.33E+07	8	0.39	8
RN	1.24E+07	3	0.47	2	3.60E+07	5	0.38	5
SA	7.78E+06	22	0.60	19	3.07E+07	13	0.42	13
SN	1.46E+07	1	0.44	1	5.03E+07	1	0.30	2
AN	8.27E+06	15	0.59	16	2.89E+07	17	0.43	18
BRS	8.66E+06	14	0.57	13	2.84E+07	19	0.43	16
BRA	7.59E+06	27	0.60	19	2.53E+07	24	0.46	24
BRN	9.57E+06	10	0.53	9	2.72E+07	21	0.44	20
BSA	7.71E+06	23	0.60	19	2.30E+07	30	0.49	30
BSN	1.04E+07	8	0.51	8	3.01E+07	15	0.42	14
BAN	7.61E+06	26	0.61	27	2.63E+07	23	0.45	23
RSA	7.69E+06	24	0.61	28	3.09E+07	12	0.41	12
RNA	8.05E+06	18	0.59	18	3.06E+07	14	0.42	14
RNS	1.13E+07	7	0.50	7	3.42E+07	6	0.39	7
SAN	7.96E+06	21	0.60	19	2.88E+07	18	0.43	19
BRSA	7.42E+06	30	0.61	26	2.38E+07	29	0.48	29
BRSN	8.71E+06	13	0.57	13	2.73E+07	20	0.44	21
BRNA	8.10E+06	16	0.58	15	2.41E+07	28	0.47	28
BSAN	7.45E+06	29	0.62	30	2.47E+07	26	0.47	25
RSAN	8.00E+06	20	0.60	24	3.10E+07	11	0.41	11
BRSAN	7.47E+06	28	0.61	28	2.48E+07	25	0.47	25

Table 7: Adaptive random (20D-40D)

sity of the population, which has been stated and experimentally verified in previous research works. Future work will include experimentation to investigate the diversity of the ending points of local searches when they start from the same initial point, and how it is possibly connected to the MAs performance. We have also put some effort in analyzing which local searches are giving the best improvement in an attempt to analyze the composition and not only the cardinality of the local search pool.

Another interesting conclusion was the superiority of the adaptive against the static scheme. Our results suggest that even larger pool sizes will further benefit the adaptive scheme. Further research on alternative adaptation and scoring schemes, will provide a better picture of the MAs performance and potential.

5. REFERENCES

[1] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi–Objective Problems*. Kluwer, New York, 2002.

[2] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley, 2006.

[3] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[4] A. Törn and A. Žilinskas. *Global Optimization*. Springer–Verlag, Berlin, 1989.

[5] N. Krasnogor. *Studies on the Theory and Design Space of Memetic Algorithms*. PhD thesis, University of the West of England, 2002. Supervisor: Dr. J.E. Smith.

[6] Y-S Ong, M-H Lim, N. Zhu, and K-K Wong. Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on systems, man, and cybernetics*, 36(1):141–152, 2006.

[7] W. Jakob. A general cost-benefit-based adaptation framework for multimeme algorithms. *Memetic Computing*, 2(3):201–218, 2010.

[8] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. *Practice and Theory of Automated Timetabling III*, pages 176–190, 2001.

[9] Y. G. Petalas, K. E. Parsopoulos, and M. N. Vrahatis. Memetic particle swarm optimization. *Annals of Operations Research*, 156(1):99–127, 2007.

[10] C. Voglis, KE Parsopoulos, DG Papageorgiou, IE Lagaris, and MN Vrahatis. Mempsode: A global optimization software based on hybridization of population-based algorithms and local searches. *Computer Physics Communications*, 183(5):1139–1154, 2012.

[11] K. E. Parsopoulos and M. N. Vrahatis. UPSO: A unified particle swarm optimization scheme. In *Lecture Series on Computer and Computational Sciences, Vol. 1, Proceedings of the International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2004)*, pages 868–873. VSP International Science Publishers, Zeist, The Netherlands, 2004.

[12] K. E. Parsopoulos and M. N. Vrahatis. Parameter selection and adaptation in unified particle swarm optimization. *Mathematical and Computer Modelling*, 46(1–2):198–213, 2007.

[13] D.G. Papageorgiou, I.N. Demetropoulos, and I.E. Lagaris. MERLIN-3.1. 1. A new version of the Merlin optimization environment. *Computer Physics Communications*, 159(1):70–71, 2004.

[14] Y-S Ong and A. J. Keane. Meta-lamarkian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, 2004.

[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] I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85:317–325, 2003.

[17] W. E. Hart. *Adaptive Global Optimization with Local Search*. PhD thesis, University of California, San Diego,USA, 1994.

[18] J. Nocedal and S.J. Wright. *Numerical optimization*. Springer verlag, 1999.

[19] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[20] J.A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[21] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 1689–1696. ACM, 2010.