Lazy Agents for Large Scale Global Optimization

Joerg Bremer¹ and Sebastian Lehnhoff²

¹Department of Computing Science, University of Oldenburg, Uhlhornsweg, Oldenburg, Germany
²R&D Division Energy, OFFIS – Institute for Information Technology, Escherweg, Oldenburg, Germany

Keywords: Global Optimization, Distributed Optimization, Multi-agent Systems, Lazy Agents, Coordinate Descent Optimization.

Abstract: Optimization problems with rugged, multi-modal Fitness landscapes, non-linear problems, and derivative-free optimization entails challenges to heuristics especially in the high-dimensional case. High-dimensionality also tightens the problem of premature convergence and leads to an exponential increase in search space size. Parallelization for acceleration often involves domain specific knowledge for data domain partition or functional or algorithmic decomposition. We extend a fully decentralized agent-based approach for a global optimization algorithm based on coordinate descent and gossiping that has no specific decomposition needs and can thus be applied to arbitrary optimization problems. Originally, the agent method suffers from likely getting stuck in high-dimensional problems. We extend a laziness mechanism that lets the agents randomly postpone actions of local optimization and achieve a better avoidance of stagnation in local optima. The extension is tested against the original method as well as against established methods. The lazy agent approach turns out to be competitive and often superior in many cases.

1 INTRODUCTION

Global optimization of non-convex, non-linear problems has long been subject to research (Bäck et al., 1997; Horst and Pardalos, 1995). Approaches can roughly be classified into deterministic and probabilistic methods. Deterministic approaches like interval methods (Hansen, 1980), Cutting Plane methods (Tuy et al., 1985), or Lipschitzian methods (Hansen et al., 1992) often suffer from intractability of the problem or getting stuck in local optima (Simon, 2013). In case of a rugged fitness landscape of multimodal, non-linear functions, probabilistic heuristics are indispensable. Often derivative free methods are needed, too.

Many optimization approaches have so far been proposed for solving these problems; among them are evolutionary methods or swarm-based methods (Bäck et al., 1997; Dorigo and Stützle, 2004; Simon, 2013; Hansen, 2006; Kennedy and Eberhart, 1995; Storn and Price, 1997). In (Bremer and Lehnhoff, 2017a), an agent-based methods has been proposed with the advanced of good scaling properties as with each new objective dimension an agent is added locally searching along the respective dimension (Bremer and Lehnhoff, 2017a). The approach uses the COHDA protocol (Hinrichs et al., 2013). In this approach, the agents perform a decentralized block coordinate descent (Wright, 2015) and self-organized aggregate locally found optima to an overall solution.

In (Hinrichs and Sonnenschein, 2014; Anders et al., 2012), the effect of communication delays in message sending and the degree of variation in such agent systems on the solution quality has been scrutinized. Increasing variation (agents with different knowledge interact) leads to better results. An increase in inter-agent variation can also be achieved by letting agents delay individual decisions. Hence, we combine the ideas from (Bremer and Lehnhoff, 2017a) and (Hinrichs and Sonnenschein, 2014) and extend the agent approach to global optimization by integrating a decision delay into the agents. In this way, the agents sort of behave lazy with regard to their decision duty.

Agents in the COHDA protocol act after the receive-decide-act metaphor (Hinrichs et al., 2013). When applied to local optimization, the decide process decides locally on the best parameter position with regard to just one respective dimension of the objective function. Thus, the agent performs a 1-dimensional optimization along an intersection of the objective function and takes the other dimensions (his
belief on the other agent's local optimizations) as fixed for the moment. We extend this approach by a mechanism that postpones the decision process. Thus the agent gathers more information from other agents (including transient ones with more communication hops) and may decide on a more solid basis.

The rest of the paper is organized as follows. After a brief recap of (large scale) global optimization, heuristics, and the agent approach for solving, the extension of laziness to the agents is explained. The effectiveness is demonstrated by comparing with the original approach and with standard algorithms.

2 RELATED WORK

Global optimization comprises many problems in practice as well as in the scientific community. These problems are often hallmarked by presence of a rugged fitness landscape with many local optima and non-linearity. Thus optimization algorithms are likely to become stuck in local optima and guaranteeing the exact optimum is often intractable; leading to the use of heuristics.

Evolution Strategies (Rechenberg, 1965) for example have shown excellent performance in global optimization especially when it comes to complex multi-modal, high-dimensional, real valued problems (Kramer, 2010; Ulmer et al., 2003). Each of these strategies has its own characteristics, strengths and weaknesses. A common characteristic is the generation of an offspring solution set by exploring the characteristics of the objective function in the immediate neighborhood of an existing set of solutions. When the solution space is hard to explore or objective evaluations are costly, computational effort is a common drawback for all population-based schemes. Real world problems often face additional computational efforts for fitness evaluations; e.g. in Smart Grid load planning scenarios, fitness evaluation involves simulating a large number of energy resources and their behaviour (Bremer and Sonnenschein, 2014).

Especially in high-dimensional problems, premature convergence (Leung et al., 1997; Trelea, 2003; Rudolph, 2001) entails additional challenges onto the used optimization method. Heuristics often converge too early towards a sub-optimal solution and then get stuck in this local optimum. This might for instance happen if an adaption strategy decreases the mutation range and thus the range of the currently searched surrounding sub-region and possible ways out of a current trough are no longer scrutinized.

On the other hand, much effort has been spent to accelerate convergence of these methods. Example techniques are: improved population initialization (Rahnamayan et al., 2007), adaptive population sizes (Ahrari and Shariat-Panahi, 2015) or exploiting sub-populations (Rigling and Moore, 1999). Sometimes a surrogate model is used in case of computational expensive objective functions (Loscheliov et al., 2012) to substitute a share of objective function evaluations with cheap surrogate model evaluations. The surrogate model represents a learned model of the original objective function. Recent approaches use Radial Basis Functions, Polynomial Regression, Support Vector Regression, Artificial Neural Network or Kriging (Gano et al., 2006); each approach with individual advantages and drawbacks.

Recently, the number of large scale global optimizations problems grows as technology advances (Li et al., 2013). Large scale problems are difficult to solve for several reasons (Weise et al., 2012). The main reasons are the exponentially growing search space and a potential change of an objective function’s properties (Li et al., 2013; Weise et al., 2012; Shang and Qiu, 2006). Moreover, evaluating large scale objectives is expensive, especially in real world problems (Sobieszczanski-Sobieski and Haftka, 1997). Growing non-separability or variable interaction sometimes entail further challenges (Li et al., 2013).

For faster execution, different approaches for parallel problem solving have been scrutinized in the past; partly with a need for problem specific adaption for distribution. Four main questions define the design decisions for distributing a heuristic: which information to exchange, when to communicate, who communicates, and how to integrate received information (Nieße, 2015; Talbi, 2009). Examples for traditional meta-heuristics that are available as distributed version are: Particle swarm (Vanneschi et al., 2011), ant colony (Colorni et al., 1991), or parallel tempering (Li et al., 2009). Distribution for gaining higher solution accuracy is a rather rare use case. An example is given in (Bremer and Lehnhoff, 2016).

Another class of algorithms for global optimization that has been popular for many years by practitioners rather than scientists (Wright, 2015) is that of coordinate descent algorithms (Ortega and Rheinboldt, 1970). Coordinate descent algorithms iteratively search for the optimum in high dimensional problems by fixing most of the parameters (components of variable vector $\mathbf{x}$) and doing a line search along a single free coordinate axis. Usually, all components of $\mathbf{x}$ a cyclically chosen for approximating the objective with respect to the (fixed) other components (Wright, 2015). In each iteration, only a lower dimensional or even scalar sub-problem has to be solved.
The multi-variable objective $f(x)$ is solved by looking for the minimum in one direction at a time. There are several approaches for choosing the step size for the step towards the local minimum, but as long as the sequence $f(x^0), f(x^1), \ldots, f(x^n)$ is monotonically decreasing the method converges to an at least local optimum. Like any other gradient based method this approach gets easily stuck in case of a non-convex objective function.

In (Hinrichs et al., 2013) an agent based approach has been proposed as an algorithmic level decomposition scheme for decentralized problem solving (Talbi, 2009; Hinrichs et al., 2011), making it especially suitable for large scale problems.

Each agent is responsible for one dimension of the objective function. The intermediate solutions for other dimensions (represented by decisions published by other agents) are regarded as temporarily fixed. Thus, each agent only searches along a 1-dimensional cross-section of the objective and thus has to solve merely a simplified sub-problem. Nevertheless, for evaluation of the solution, the full objective function is used. In this way, the approach achieves an asynchronous coordinate descent with the ability to escape local minima by parallel searching different regions of the search space. The approach uses as basis a protocol from Hinrichs et al., 2013).

In Hinrichs et al., 2013 a fully decentralized agent-based approach for combinatorial optimization problems has been introduced. Originally, the combinatorial optimization heuristics for distributed agents (COHDA) had been invented to solve the problem of predictive scheduling (Sonnenchein et al., 2014) in the Smart Grid.

The key concept of COHDA is an asynchronous iterative approximate best-response behavior, where each participating agent – originally representing a decentralized energy unit – reacts to updated information from other agents by adapting its own action (select an energy production scheme that enables group of energy generators to fulfil an energy product from the market as good as possible). All agents $a_i \in A$ initially only know their own respective search space $S_i$ of feasible energy schedules that can be operated by the own energy resource. From an algorithmic point of view, the difficulty of the problem is given by the distributed nature of the system in contrast to the task of finding a common allocation of schedules for a global target power profile.

Thus, the agents coordinate by updating and exchanging information about each other. For privacy and communication overhead reasons, the potential flexibility (alternative actions) is not communicated as a whole by an agent. Instead, the agents communicate single selected local solutions (energy production schedules in the Smart Grid case) within the approach as described in the following.

First of all, the agents are placed in an artificial communication topology based on the small-world scheme, (e.g. a small world topology (Watts and Strogatz, 1998), such that each agent is connected to a non-empty subset of other agents. This overlay topology might be a ring in the least connected variant.

Each agent collects two distinct sets of information: on the one hand the believed current configuration $y_i$ of the system (that is, the most up to date information $a_i$ has about currently selected schedules of all agents), and on the other hand the best known combination $y_i^\ast$ of schedules with respect to the global objective function it has encountered so far.

Beginning with an arbitrarily chosen agent by passing it a message containing only the global objective (i.e. the target power profile), each agent repeatedly executes the three steps perceive, decide, act (cf. (Nieße et al., 2014)):

Algorithm 1: Basic scheme of an agent’s decision on local optima in the extension of COHDA to global optimization.

1: // let $x \in \mathbb{R}^d$ an intermediate solution
2: \begin{align*}
    x_k & \leftarrow \begin{cases}
    x_k & \text{if } x_k \in k_{aj} & \forall k \neq j \\
    x \sim U(\text{range}) & \text{else}
    \end{cases} \\
    \frac{\text{if } f(x) < f(x_{old}) & \text{then}}{
3: \text{solve with Brent optimizer:}
4: \begin{align*}
    x_j & \leftarrow \arg\min f_j(x) = f(x,x) = f(x_1, x_{j-1}, \ldots, x_{j+1}, \ldots, x_d) \\
5: \text{update workspace } K_j
6: \end{align*}
7: \end{if}

1. **perceive:** When an agent $a_i$ receives a message $\gamma_p$ from one of its neighbors (say, $a_p$), it imports the contents of this message into its own memory.

2. **decide:** The agent then searches $S_i$ for the best own local solution regarding the updated system state $\gamma_i$ and the global objective function. Local constraints are taken into account in advance if applicable. Details regarding this procedure have been presented in (Nieße et al., 2016). If a local solution can be found that satisfies the objective, a new solution selection is created. For the following comparison, only the global objective function must be taken into account: If the resulting modified system state $\gamma_i$ yields a better rating than the current solution candidate $y_i^\ast$, a new solution candidate is created based on $y_i$. Otherwise the old solution candidate still reflects the best combination regarding the global objective, so the agent reverts to its old selection stored in $y_i^\ast$. 
3. act: If $\gamma_i$ or $\gamma^*_i$ has been modified in one of the previous steps, the agent finally broadcasts these to its immediate neighbors in the communication topology.

During this process, for each agent $a_i$, its observed system configuration $\gamma_i$ as well as solution candidate $\gamma'_i$ are filled successively. After producing some intermediate solutions, the heuristic eventually terminates in a state where for all agents $\gamma_i$ as well as $\gamma'_i$ are identical, and no more messages are produced by the agents. At this point, $\gamma'_i$ is the final solution of the heuristic and contains exactly one schedule selection for each agent.

The COHDA protocol has meanwhile been applied to many different optimization problems (Bremer and Lehnhoff, 2017b; Bremer and Lehnhoff, 2017c). In (Bremer and Lehnhoff, 2017a) COHDA has also been applied to the continuous problem of global optimization.

3 LAZY COHDAGO

In (Bremer and Lehnhoff, 2017a) the COHDA protocol has been applied to global optimization (COHDAgo). Each agent is responsible for solving one dimension $x_i$ of a high-dimensional function $f(x)$ as global objective. Each time an agent receives a message from one of its neighbors, the own knowledge-base with assumptions about optimal coordinates of the optimum of $f$ (with $x^* = \arg\min f(x)$) is updated. Let $a_j$ be the agent that just has received a message from agent $a_i$. Then, the workspace $K_j$ of agent $a_j$ is merged with information from the received workspace $K_i$. Each workspace $K$ of an agent contains a set of coordinates $x_k$ such that $x_k$ reflects the $k$th coordinate of the current solution $x$ so far found from agent $a_k$. Additionally, information about other coordinates $x_{k_1}, \ldots, x_{k_n}$ reflecting local decisions of $a_{k_1}, \ldots, a_{k_n}$ that $a_i$ has received messages from is also integrated into $K_j$ if the information is newer or outdated the already known. Thus each agent gathers also transient information; finally about all local decisions.

In general, each coordinate $x_i$ that is not yet in $K_j$ is temporarily set to a random value $x_i \sim U(x_{\text{min}}, x_{\text{max}})$ for objective evaluation. W.l.o.g. all unknown values could also be set to zero. But, as many of the standard benchmark objective functions have their optimum at zero, this would result in an unfair comparison as such behavior would unintentionally induce some priori knowledge. Thus, we have chosen to initialize unknown values with a random value.

After the update procedure, agent $a_j$ takes all elements $x_k \in x$ with $k \neq j$ as temporarily fixed and starts solving a 1-dimensional sub-problem: $x_j = \arg\min f(x, x_j)$; where $f$ is the objective function with all values except element $x_j$ fixed. This problem with only $x$ as the single degree of freedom is solved using Brent’s method (Brent, 1971). Algorithm 1 summarizes this approach.

Brent’s method originally is a root finding procedure that combines the previously known bisection method and the secant method with an inverse quadratic interpolation. Whereas the latter are known
for fast convergence, bisection provides more reliability. By combining these methods – a first step was already undertaken by (Dekker, 1969) – convergence can be guaranteed with at most $O(n^2)$ iterations (with $n$ iterations for the bisection method). In case of a well-behaved function the method converges even superlinearly (Brent, 1971). We used an evaluated implementation from Apache Commons Math after a reference implementation from (Brent, 1973).

After $x_j$ has been determined with Brent’s method, $x_j$ is communicated (along with all $x_i$ previously received from agent $a_i$) to all neighbors if $f(x^*)$ with $x_j$ gains a better result than the previous solution candidate. Figure 1 summarizes this procedure.

Into this agent process, we integrated laziness. Figure 2 shows the idea. As an additional stage in the receive-decide-act protocol, a random decision is made whether to postpone a decision on local optimality based on aggregated information. In contrast to the approach of (Anders et al., 2012), aggregation is nevertheless done with this additional stage. Only after information aggregation and thus after belief update it is randomly decided whether to continue with the decision process of the current belief (local optimization of the respective objective dimension) or with postponing this process. By doing so, additional information – either update information from the same agent, or additional information from other agents – may meanwhile arrive and aggregate. The delay is realized by putting the trigger message in a holding stack and resubmitting it later. Figure 3 shows the relative frequencies of delay (additional aggregation steps) that occur when a uniform distribution $U(0,1)$ is used for deciding on postponement. The likelihood of being postponed is denoted by $\lambda$. In this way, information may also take over newer information and thus may trigger a resumption at an older search branch that led to a dead-end. In general, the disturbance within the system increases, and thus premature convergence is better prevented. We denote this extension lazyCOHDAgo.

4 RESULTS

For evaluation, we used a set of well-known test functions that have been developed for benchmarking optimization methods: Elliptic, Ackley (Ulmer et al., 2003), Egg Holder (Jamil and Yang, 2013), Rastrigin (Aggarwal and Goswami, 2014), Griewank (Locatelli, 2003), Quadric (Jamil and Yang, 2013), and examples from the CEC '13 Workshop on Large Scale Optimization (Li et al., 2013).

In a first experiment, we tested the effect of lazy agents and solved a set of test functions with agents of different laziness $\lambda$. Table 1 shows the result for 50-dimensional versions of the test functions. In this rather low dimensional cases the effect is visible, but not that prominent. In most cases a slight improvement can be seen with growing laziness factor ($\lambda = 0$ denotes no laziness at all and thus responds to the original COHDAgo). The Elliptic function for example shows no improvement. In some cases, e.g. for the Quadric function the result quality deteriorates. But, also an overshoot can be observed with the Griewank function where the best result is obtained with a laziness of $\lambda = 0.3$. When applied to more complex and higher-dimensional objective functions the effect is way more prominent as can be seen in Table 2. The CEC $f_1$ function (Li et al., 2013) is a shifted elliptic function which is ill-conditioned with condition number $\approx 10^6$ in the 1000-dimensional case. Due to dimensionality these results have also been obtained with a laziness of $\lambda = 0.99$). From the wide range of solution qualities for $\lambda = 0.9$ – the achieved minimum result out of 20 runs was (200-dimensional case) $3.40 \times 10^{-19}$, which is almost as good as the result for $\lambda = 0.9$ – it can be concluded that the agent system is less susceptible to premature convergence and thus yields better mean results. The Rosenbrock function is a asymmetrically, non-linearly shifted version of (Rosenbrock, 1960) multiplied by the Alpine function.

Finally, we compared the results of the lazy agent approach with other established meta-heuristics for functions where the agent approach was successful. Please note that for some function (e.g. the result in table 1) were not that promising. For comparison we used the co-variance matrix adaption evolution strat-
Table 1: Performance of the lazy agent approach on different 50-dimensional test functions for different laziness factors $\lambda$.

<table>
<thead>
<tr>
<th>Function</th>
<th>$\lambda = 0.0$</th>
<th>$\lambda = 0.3$</th>
<th>$\lambda = 0.6$</th>
<th>$\lambda = 0.9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic</td>
<td>$1.527 \times 10^{-12} \pm 2.876 \times 10^{-13}$</td>
<td>$1.527 \times 10^{-12} \pm 1.976 \times 10^{-13}$</td>
<td>$1.527 \times 10^{-12} \pm 2.594 \times 10^{-13}$</td>
<td>$1.527 \times 10^{-12} \pm 7.48 \times 10^{-14}$</td>
</tr>
<tr>
<td>Ackley</td>
<td>$1.306 \times 10^{-1} \pm 2.988 \times 10^{-1}$</td>
<td>$1.217 \times 10^{-1} \pm 1.665 \times 10^{-1}$</td>
<td>$1.205 \times 10^{-1} \pm 1.86 \times 10^{-1}$</td>
<td>$1.124 \times 10^{-1} \pm 2.088 \times 10^{-1}$</td>
</tr>
<tr>
<td>EggHolder</td>
<td>$1.453 \times 10^{10} \pm 8.639 \times 10^{9}$</td>
<td>$1.423 \times 10^{10} \pm 8.811 \times 10^{9}$</td>
<td>$1.384 \times 10^{10} \pm 9.119 \times 10^{9}$</td>
<td>$1.345 \times 10^{10} \pm 9.441 \times 10^{9}$</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>$2.868 \times 10^{2} \pm 2.493 \times 10^{2}$</td>
<td>$2.870 \times 10^{2} \pm 1.569 \times 10^{2}$</td>
<td>$2.868 \times 10^{2} \pm 2.427 \times 10^{2}$</td>
<td>$2.858 \times 10^{2} \pm 3.088 \times 10^{2}$</td>
</tr>
<tr>
<td>Griewank</td>
<td>$2.95 \times 10^{-3} \pm 9.328 \times 10^{-3}$</td>
<td>$1.478 \times 10^{-3} \pm 4.674 \times 10^{-3}$</td>
<td>$1.59 \times 10^{-3} \pm 3.219 \times 10^{-3}$</td>
<td>$3.07 \times 10^{-2} \pm 4.132 \times 10^{-2}$</td>
</tr>
<tr>
<td>Quadric</td>
<td>$6.51 \times 10^{-26} \pm 6.525 \times 10^{-26}$</td>
<td>$1.196 \times 10^{-23} \pm 8.128 \times 10^{-26}$</td>
<td>$3.65 \times 10^{-5} \pm 5.141 \times 10^{-21}$</td>
<td>$4.43 \times 10^{-5} \pm 1.40 \times 10^{-14}$</td>
</tr>
</tbody>
</table>

Table 2: Performance of the lazy agent approach on different high-dimensional, ill-conditioned test functions for different laziness factors $\lambda$.

<table>
<thead>
<tr>
<th>Function</th>
<th>$\lambda = 0.0$</th>
<th>$\lambda = 0.9$</th>
<th>$\lambda = 0.99$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEC $f_1$, $d = 200$</td>
<td>$1.81 \times 10^{10} \pm 5.78 \times 10^{9}$</td>
<td>$2.20 \times 10^{8} \pm 4.11 \times 10^{9}$</td>
<td>$3.40 \times 10^{-19} \pm 1.54 \times 10^{-23}$</td>
</tr>
<tr>
<td>CEC $f_1$, $d = 500$</td>
<td>$4.28 \times 10^{6} \pm 8.28 \times 10^{9}$</td>
<td>$6.55 \times 10^{4} \pm 1.85 \times 10^{5}$</td>
<td>$7.60 \times 10^{-19} \pm 1.31 \times 10^{-21}$</td>
</tr>
<tr>
<td>Rosenbrock, $d = 250$</td>
<td>$1.01 \times 10^{-5} \pm 1.71 \times 10^{-5}$</td>
<td>$2.41 \times 10^{-7} \pm 5.37 \times 10^{-7}$</td>
<td>$5.68 \times 10^{-8} \pm 1.60 \times 10^{-7}$</td>
</tr>
</tbody>
</table>

Table 3: Comparison of the lazy agent approach with different established meta-heuristics.

<table>
<thead>
<tr>
<th>f</th>
<th>CMA-ES</th>
<th>DE</th>
<th>PSO</th>
<th>lazy COHDAgo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic</td>
<td>$3.41 \times 10^{-7} \pm 7.43 \times 10^{-7}$</td>
<td>$4.48 \times 10^{-7} \pm 2.24 \times 10^{-7}$</td>
<td>$2.65 \times 10^{-8} \pm 8.39 \times 10^{-9}$</td>
<td>$1.14 \times 10^{-41} \pm 2.64 \times 10^{-47}$</td>
</tr>
<tr>
<td>Ackley</td>
<td>$1.02 \times 10^{0} \pm 7.07 \times 10^{-7}$</td>
<td>$4.73 \times 10^{-5} \pm 7.06 \times 10^{-5}$</td>
<td>$2.0 \times 10^{0} \pm 0.0 \times 10^{0}$</td>
<td>$1.54 \times 10^{1} \pm 1.01 \times 10^{-1}$</td>
</tr>
<tr>
<td>Alpine</td>
<td>$4.2 \times 10^{0} \pm 3.96 \times 10^{0}$</td>
<td>$2.82 \times 10^{-3} \pm 9.18 \times 10^{-5}$</td>
<td>$6.61 \times 10^{-9} \pm 1.29 \times 10^{-8}$</td>
<td>$4.51 \times 10^{-12} \pm 1.32 \times 10^{-13}$</td>
</tr>
<tr>
<td>Griewank</td>
<td>$9.99 \times 10^{-4} \pm 3.11 \times 10^{-3}$</td>
<td>$4.41 \times 10^{-4} \pm 1.51 \times 10^{-5}$</td>
<td>$8.92 \times 10^{-3} \pm 4.54 \times 10^{-3}$</td>
<td>$5.11 \times 10^{-16} \pm 9.2 \times 10^{-16}$</td>
</tr>
</tbody>
</table>

Table 4: Respective best results (residual error) out of 20 runs each for the comparison from Table 3.

<table>
<thead>
<tr>
<th>f</th>
<th>CMA-ES</th>
<th>DE</th>
<th>PSO</th>
<th>lazy COHDAgo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptic</td>
<td>$9.28 \times 10^{-7}$</td>
<td>$1.89 \times 10^{-9}$</td>
<td>$1.04 \times 10^{-4}$</td>
<td>$1.14 \times 10^{-41}$</td>
</tr>
<tr>
<td>Ackley</td>
<td>$6.02 \times 10^{-6}$</td>
<td>$4.73 \times 10^{-2}$</td>
<td>$2.0 \times 10^{1}$</td>
<td>$1.52 \times 10^{1}$</td>
</tr>
<tr>
<td>Alpine</td>
<td>$1.28 \times 10^{-1}$</td>
<td>$2.7 \times 10^{-3}$</td>
<td>$1.2 \times 10^{-15}$</td>
<td>$4.3 \times 10^{-12}$</td>
</tr>
<tr>
<td>Griewank</td>
<td>$3.57 \times 10^{-6}$</td>
<td>$4.23 \times 10^{-4}$</td>
<td>$1.51 \times 10^{-5}$</td>
<td>$0 \times 10^{0}$</td>
</tr>
</tbody>
</table>

Large scale global optimization is a crucial task for many real-world applications in industry and engineering. Most meta-heuristics deteriorate rapidly with growing problem dimensionality. We proposed a laziness extension to an agent-based algorithm for global optimization and achieved a way better performance when applied to large scale problems. By randomly postponing the agent's decision on local optimization leads to less vulnerability to premature convergence, obviously due to an increasing inter-agent variation (Anders et al., 2012) and thus to the incorporation of past (outdated) information. This may stimulate search in already abandoned paths. Delaying the reaction of the agents in COHDA is known to increase the diversity in the population and thus leading to at least equally good results but with a larger number of steps (Hinrichs and Sonnenschein, 2014), but for some use cases—like large scale global optimization—also to better results.

The lazy COHDAgo approach has shown good and sometimes superior performance especially re-
garding solution quality. In future work, it may also be promising to further scrutinize the impact of the communication topology as design parameter.

REFERENCES


