A Memetic Framework for Solving Difficult Inverse Problems

Maciej Smołka(✉) and Robert Schaefer

AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{schaefer,smolka}@agh.edu.pl

Abstract. The paper introduces a multi-deme, memetic global optimization strategy *Hierarchic memetic Strategy* (HMS) especially well-suited to the solution of a class of parametric inverse problems. This strategy develops dynamically a tree of dependent populations (demes) searching with the various accuracy growing from the root to the leaves. The search accuracy is associated with the accuracy of solving direct problems by *hp*–adaptive Finite Element Method. Throughout the paper we describe details of exploited accuracy adaptation and computational cost reduction mechanisms, an agent-based architecture of the proposed system, a sample implementation and preliminary benchmark results.

Keywords: Inverse problems · Hybrid optimization methods · Memetic algorithms

1 Motivation

Inverse problems form an important area of the contemporary research related to fundamental problems in science and engineering (see *e.g.* [1]). Among its numerous applications one can find such activities as oil and gas explorations, material processing and others. A quite general definition of the inverse problem is to find a value of a parameter $\omega^* \in \mathcal{D}$ realizing

$$\min_{\omega \in \mathcal{D}} \{ f(u_o, u(\omega)) : \ A(u(\omega)) = 0 \} \tag{1}$$

where A is a *direct problem operator*, $u(\omega) \in U$ is the direct solution corresponding to ω, $u_o \in \mathcal{O}$ is an observation (typically a measured quantity related somehow to the direct solution) and $f(\mathcal{O}, U) \longrightarrow \mathbb{R}_+$ is *a misfit functional.* In a typical situation U is a Sobolev space and $A : U \longrightarrow U'$ is a differential operator between U and its conjugate. When solving such problems one usually faces some significant obstacles. One of them is the ill-conditioning, i.e. a small change

The work presented in this paper has been partially supported by Polish National Science Center grants no. DEC-2012/07/B/ST6/01229 and DEC-2011/03/B/ST6/01393.

in parameters sometimes results in a big change in results. Other noticeable difficulties are the multi-modality, i.e. the non-uniqueness of solutions, and possible low regularity of the misfit functional. Both of them significantly reduce the usefulness of computationally relatively inexpensive convex optimization methods (such as gradient-based ones), because in the lack of the misfit differentiability their use is problematic in general and, even worse, in the case of multiple local optima they do not deliver the guarantee of finding the global one.

There exist some methods to overcome those difficulties. One of the most popular is the misfit regularization (see *e.g.* [2]) providing a modified version of the misfit, which is regular and convex (hence unimodal). This can be a very effective technique, however it is not very useful when the considered inverse problem is inherently multimodal and we need to find all minima. On the other hand, a careless use of the regularization can lead to the replacement of the original problem solution with an artificial solution of the over-regularized misfit. A different way is to use a stochastic global optimization methods from simple Monte Carlo type to more sophisticated single- and multi-deme genetic searches (see *e.g.* [3–5]). Such methods may handle irregularity and multimodality, but the price is the high computational cost and the low accuracy. Another possibility is to perform multiple convex searches from a set of points generated randomly (multistart strategy). Such methods might be additionally improved by the sophisticated post-processing leading to the reduction of a random sample from which local methods are started or the early suspension of non-promising local searches (see *e.g.* [6,7]).

The authors intend to synthesize slightly diverse ideas of the inverse analysis arising from the following sources.

Hierarchic Genetic Strategy (HGS). This strategy develops dynamically a tree of dependent *demes* i.e. sub-populations of the total multiset of various type individuals created by the strategy. The root-deme performs the most global search with a low accuracy. The search performed by demes located deeper in the tree is more localized and more accurate. See [8] for details and [9] for HGS floating point encoding implementation. An important HGS extension going towards the effective solving of the inverse parametric problems is the *hp*–HGS strategy (see [10] and references therein) which combines HGS with the *hp*-adaptive Finite Element Method (*hp*–FEM) [11]. This strategy offers the advantageous computational cost resulting from the common scaling of the *hp*–FEM error according to the various accuracy of HGS inverse search in the root deme, branch demes and leaf demes. The *hp*–HGS asymptotic guarantee of finding all extremes and the computational cost reduction rate are discussed in [10].

Memetic algorithms (see *e.g.* [12]) allow to compose various techniques into a single population-based stochastic strategy in order to obtain more efficiency and flexibility. Candidate solutions are represented as software agents, other agents are responsible for governing populations, which leads to the idea of the computing Multi-Agent System (MAS) (see *e.g.* [13,14]). The first attempt to apply agents in profiling of HGS demes is described in [15]. An example of solving inverse parametric problem by an Evolutionary Multi-Agent System (EMAS) can be found in [16]. The paper [17] shows a different way of a memetic enhancement of genetic search by introducing 'gradient mutation' into the genetic solving of inverse problems coming from the computational mechanics.

Clustered Genetic Search (CGS) tries to extract the knowledge from the genetic sample (the population) or a sequence of samples in order to approximate central parts of local extreme's basins of attraction (see *e.g.* [7]). CGS follows the simple strategy introduced by Törn [6], which performs a density clustering of the uniformly sampled population undertaking the elitist selection.

The solution proposed in this paper, called *Hierarchic Memetic Strategy* (HMS) combines all mechanisms described above in a form of a loosely coupled tree of searching demes. The novelty of our proposition consists of the intensive profiling of searching process towards essential decreasing of computational cost and exploring multiple extremes. This profiling utilize intensively the knowledge about the solving problem extracted from the evolving demes and their current structure.

2 HMS Architecture

The main idea of the HMS is to provide a global optimization tool especially suited to solving difficult inverse problems. Their difficulty lies in their inherent multi-modality as well as the nontrivial computational cost of a direct problem solution, which is necessary for evaluating the misfit. Nevertheless, they also have some features we can take advantage of. First of all, their global minimum value is well-known (and equal to 0), which can be used in e.g. the construction of stopping conditions for stochastic evolution. Second, in some important cases the cost of the direct problem solution can be modulated by an assumed accuracy of the solution: it is the case of *hp*-FEM direct solvers [11].

As a global optimization tool the HMS tries to combine the high-level exploratory ability with the accuracy and efficiency of a local optimization method. Contrary to two-phase methods in which the global phase is followed by local searches, the HMS goes 'memetic way', i.e. intermixes local-optimization-oriented mechanisms into a global stochastic search machinery. The global part follows the multi-population evolutionary approach introduced by the HGS [8]. Namely the global search is performed by a collection of genetic populations. The populations can evolve in parallel, but they are not mutually independent. The structure of the dependency relation is hierarchical (i.e. tree-like, see Fig. 1) with a restricted number of levels. The HGS proved to have considerable exploratory capabilities together with a good search accuracy especially with floating-point phenotype encoding [9]. The HMS, naturally, tries to retain these abilities at the same time going beyond the HGS in some aspects. First of all, it adds local optimization to the set of operations applied to the genetic individuals. But this is done with care in order to avoid the premature population convergence on one hand and the high cost of running instances of a local method from inappropriate points on the other hand. Namely some genetic individuals (but not necessarily all of them) receive an identity and some intelligence hence becoming

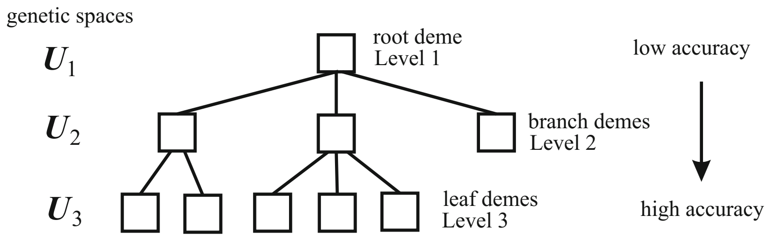

Fig. 1. HGS-like evolutionary population tree

independent agents in a multi-agent system (MAS), and the decision of performing the local search becomes their own responsibility. Moreover the demes are managed by special controller agents. Note that this is somewhat similar but at the same time significantly different from the Globally Balanced HGS (GB-HGS) [15] where only demes have corresponding agents. The idea of turning a passive genetic individual into an intelligent agent has some further consequences. We have to redefine the genetic operations in such a way that they can be applied to agents and while there is no big problem with the mutation and the crossover (but one has to note that in this case a new agent is activated), the selection cannot be performed in the simple genetic (or evolutionary) way. Namely we follow the lines of the EMAS [13,14], thus performing an operation analogous to the proportional selection but realized as a two-agent rendezvous.

In the sequel we shall present the structure of the HMS starting from a description of HMS agent types.

2.1 HMS Agent Types

Master Agent (MA). As a global system coordinator it is started as a first agent in the HMS MAS. Its responsibilities include performing the system initialization including the activation of other basic agents, i.e. the Objective Agent and a Local Agent of the deme-tree root. After the initialization, the Master Agent starts the global loop of deme coordination and checks if the global stopping condition is satisfied. It is shown in the following algorithm:

```
1: create OA
2: create root location node
3: repeat
4:     receive proposals from DAs and choose one
5: until global stop condition is satisfied.
```

Deme Agent (DA). It is a deme-tree node coordinator. Each deme has an associated level of computational accuracy stored as a property of the corresponding Local Agent. In fact Deme Agent is an abstract class with two different specializations: Evolutionary Agent and Local Agent.

Evolutionary Agent (EA). This is a simple (passive) evolutionary population owner. Periodically, after receiving the permission from the Master Agent it lets its population evolve for a fixed number of generations (this is called a *metaepoch*) and then sprouts a new deme from the current best individual unless the sprout condition is not satisfied (see the algorithm below). Note that similar agents form the structure of the GB-HGS [15]. The Evolutionary Agent algorithm may be presented in the following way.

```
 1: create initial deme population
 2: repeat
 3:     send a proposal to MA
 4:     if MA has accepted the proposal then
 5:         evolve owned population for a fixed step number
 6:         if the best individual satisfies the sprout condition then
 7:             create new child DA
 8:         end if
 9:     end if
10: until local stop condition is satisfied
```

Local Agent (LA). The Local Agent owns a population of Computational Agents and acts as their action local scheduler. Namely it receives action proposals from Computational Agents, selects one of them according to a probability distribution, send a proposal to the Master Agent and if the proposal is accepted, lets the selected Computational Agent perform its action (see the algorithm below). The Local Agent's responsibilities include also some action coordination, such as checking if a pending sprout action is allowed. The Local Agent algorithm is presented below.

```
 1: create initial deme population
 2: repeat
 3:     send CFP to all active CAs
 4:     receive action proposals from CAs and choose one
 5:     send a proposal to MA
 6:     if MA has accepted the proposal then
 7:         if CA action creates new individual then
 8:             create new CA
 9:         else if chosen action is SPROUT then
10:             if sprouting can be performed then
11:                 create new child DA
12:             end if
13:         end if
14:     end if
15: until local stop condition is satisfied
```

Computational Agent (CA). It is an *active* individual of the HMS genetic population. It owns an immutable genotype consisting of an encoded domain point (a chromosome) and a level of the computational precision. The precision level must be consistent with the owning Local Agent's level. The mutable part of a

Computational Agent's state includes a nonnegative memetic parameter called *life energy*. The life energy is exchanged during a Computational Agent action execution such that the total energy remains constant within each deme. Only agents with the positive life energy are considered active (alive) and take part in the system evolution. Namely there exists a pool of actions from which an active Computational Agent can choose one at a time to perform. The available action pool size depends primarily on the agent's life energy but can be affected by other parameters as well. The action selection is determined by a given probability distribution. Finally, the action is performed only if permitted by the owning Local Agent (see the algorithm below).

```
 1: request objective computing from OA
 2: while life energy $> 0$ do
 3:     receive CFP from owning LA
 4:     choose an available action
 5:     send the proposal to LA
 6:     if received permission from LA then
 7:         perform chosen action
 8:         update life energy
 9:     end if
10: end while
```

There is an energy quantum related to each action, which is spent (during GET it can sometimes be gained) by a Computational Agent during the action execution. Currently the following actions are considered (cf. [14]): GET, MUTATE, CROSSOVER, LOCOPT and SPROUT.

The GET action is the above-mentioned kind of the distributed selection. It is a two-agent stochastic duel during which the proper quantum energy moves from the loser to the winner. A Computational Agent with a lower (i.e. better because closer to the global minimum) objective value has more chances to win. MUTATE and CROSSOVER are straightforward counterparts of corresponding genetic (or evolutionary) operations, like e.g. the normal mutation and the arithmetic crossover. The SPROUT action is inspired by the child branch sprouting operation, which is fundamental in the HGS [8]. In the HMS it produces a new deme together with its Local Agent and an initial population of Computational Agents. The probability of selecting SPROUT increases with the decreasing value of the objective. Obviously SPROUT makes no sense at the leaf level, where it can be optionally replaced with LOCOPT. The LOCOPT is a local optimization method execution started from the agent's decoded chromosome. In the current realization LOCOPT is allowed only at the leaves and, as in the case of SPROUT, the probability of its selection is high for Computational Agents with the low objective value.

Objective Agent (OA). In the real HMS use case (i.e. in solving inverse problems) the objective value is computed externally by a specialized direct solver. The responsibility of an Objective Agent (typically one in the whole system) is to provide a proper solver gateway, i.e. to execute the solver process (or several parallel processes) properly and to transfer the input data to the solver and the solver output back to the HMS. Additional Objective Agent activities may include: caching solver results, solver instance pooling (in the case of the parallel execution) and scheduling objective computations according to a sophisticated optimizing policy (e.g. a diffusion-based one [18]).

2.2 Population Structure

As it was stated before the HMS genetic population is decomposed into dependent demes forming a dynamically-changing tree of the fixed maximal depth m. Genetic individuals, i.e. Computing Agents, located at the tree levels close to the root perform the chaotic and inaccurate search, whereas going towards the leaves the search becomes more and more focused and the accuracy is increased (see Fig. 1). The variability of the search accuracy results from the diversity of the genotype encoding precision used at different tree levels. The latter of course depends on the encoding type. In the case of the binary encoding (as in the Simple Genetic Algorithm) it can be achieved by the binary genotype length variation, whereas in the case of the real number encoding (as in the Simple Evolutionary Algorithm) it can be realized by the appropriate phenotype scaling. The latter case is used in the prototype implementation of the HMS so we present here some details. The description follows the ones presented in papers [9,15].

In the real number encoding both phenotypes and genotypes are vectors from $\mathbb{R}^N$. We assume that the solution domain is a box $\mathcal{D} = [a_1, b_1] \times \cdots \times [a_N, b_N]$ and we take a sequence of scaling factors $\eta_i \in \mathbb{R}$ such that $\eta_1 > \eta_2 > \ldots \eta_{m-1} > \eta_m = 1$. Then the genetic universum at the tree level j is

$$U_j = \left[0, \frac{b_1 - a_1}{\eta_j}\right] \times \cdots \times \left[0, \frac{b_N - a_N}{\eta_j}\right] \tag{2}$$

and the encoding mapping at the level j is defined as

$$\mathcal{D} \ni x \longmapsto \left\{ \frac{x_k - a_k}{\eta_j} \right\}_{k=1}^{N} \in U_j. \tag{3}$$

Moreover we define the scaling mapping $scale_{i,j} : U_i \ni x \mapsto \frac{\eta_i}{\eta_j} x \in U_j$. In such a genetic universa the search at lower levels is more chaotic (because the mutation acts stronger) and less precise (the loss of precision is caused by limitations in the real number representation). One can use various genetic operators in such an encoding, but among the most important one can find the normal mutation $y_i = x_i + \mathcal{N}(0, \sigma_j^{mut})$ for $i = 1, \ldots, N$, where $\mathcal{N}(0, \sigma_j^{mut})$ is a normally-distributed random variable with the standard deviation σ_j^{mut} set separately for each level j, and the arithmetic crossover $y_i = x_i^1 + \mathcal{U}([0,1])(x_i^2 - x_i^1)$ for $i = 1, \ldots, N$, where $\mathcal{U}([0,1])$ is a random variable distributed uniformly over the interval $[0,1]$. Both operators are used in our sample implementation. Furthermore we exploit the classical fitness-proportional (roulette-wheel) selection in passive populations (on Evolutionary Agents) additionally preserving the best individual of each generation. A newly sprouted deme's population is sampled according to the N-dimensional Gaussian distribution centered at the properly encoded fittest individual of the parent process with the diagonal covariance matrix with values $(\sigma_j^{sprout})^2$ on the diagonal. The sprout cannot be performed in population P at level j if there exists a population P' at level $j+1$ such that $|\overline{y} - scale_{i,i+1}(y)| < c_j$, where y is the best individual in P, $\overline{y}$ is the average phenotype of P' and c_j is a branch comparison constant.

Finally, it should be mentioned that the further utilization of the knowledge gathered during the multi-level enhanced genetic evolution is possible by means of the clustering technique, in which better approximation of attraction basins of the local minima can be developed allowing yet more precise application of local optimization methods.

3 Sample Implementation

As our algorithmic framework is sophisticated, agent-based one, it also poses several challenges for the implementation task. Two main goals were especially considered during the design phase: flexibility and efficiency.

Flexibility. It was quite obvious from the beginning that HMS, being a framework, should be extensively configurable, which means that it has to embrace changes in such aspects as various particular sub-algorithms (*e.g.* the computation of CA action probabilities), local and global stopping conditions, local optimization methods, objective approximations etc. All such issues are addressed primarily by the extensive use of appropriate design patterns (such as Strategy or Proxy). Some aspects of configurability are obtained through the inclusion of scripting capabilities into the solid Java skeleton, namely some sub-algorithms can be defined in separate JavaScript scripts. There is also a higher level of flexibility reached by HMS. Through the foundation on the Java Agent Development Framework JADE [19] (in version 4.2) it obtained a potential ability of distributed deployment. The use of JADE is justified by its de facto standard position in the multi-agent middleware area and the relative easiness to write code controlling concurrent agents communicating through asynchronous message passing. JADE's FIPA standard compliance encouraged us to base HMS agent communication protocols on the FIPA solutions as well. Both the location selection performed by the Master Agent with the cooperation of Local Agents and the Computational Agent selection conducted by a Local Agent are a modifications of the FIPA Contract-Net protocol. Another example is the multiple use of the FIPA Request protocol (e.g. requesting the objective value from the Objective Agent by a Computational Agent).

Efficiency. A message-intensive multi-agent system may seem not very suitable for numerical computations. However, one should consider that in our real use case the cost of solving a direct problem dominates the other costs, including agent thread allocation and asynchronous message passing, by far. Hence our main effort is to reduce the number of the direct solver calls and decrease the cost of the particular direct solution as far as possible (and this is obtained through the presented analysis) instead of looking for a more time-effective implementation environment, which would lack other above-mentioned desired features.

4 Benchmark Tests

Some preliminary benchmark tests were performed. Their aim was basically to prove the HMS abilities to find the global minimum with the assumed accuracy in comparison with an already-tested effective tool: GB-HGS [15]. Namely we took the best accuracy obtained by GB-HGS in the optimization of two popular benchmark functions and treated this accuracy as the goal for HMS. The chosen type of tests (i.e. the tests with an assumed accuracy) influenced the setting of the HMS stopping conditions. Namely the global stopping condition was satisfied if a leaf approached the global minimum with the given accuracy, whereas a leaf stopping condition was satisfied if the leaf approached the global minimum or if a fixed number of its consecutive metaepochs were ineffective, i.e there was no significant change in the leaf's population average fitness. As the active populations do not use the basic notion of metaepoch, for stopping condition definition we use performing the number of steps equal to the current population size instead.

As benchmarks we chose the 20-dimensional Rastrigin function over the box $[-512, 512]^{20}$ and the 10-dimensional Ackley path function over the box $[-30, 30]^{10}$. Both test were repeated 10 times. The tree had 2 levels. At the root level an Evolutionary Agent (i.e. a passive population) was run, whereas at the leaf level we executed Local Agents together with populations of Computational Agents (i.e. active individuals capable of performing the local optimization). The normal mutation and the arithmetic crossover were used as the genetic operations. To make the comparison more clear in both benchmarks most of HMS execution parameters was set exactly (or almost exactly) as in GB-HGS.

In 10D Ackley function minimization we assumed the accuracy of 0.01 (in this case the obtained accuracy was much better). The execution parameters for 10D Ackley function are summarized in Tab. 1. Note that the metaepoch length parameter is not directly applicable to Local Agents (see above). Similarly, the population size in this case is not constant, in our simulations it varied between 10 and 30. The objective call statistics are shown in Tab. 2. The cost of a local method application is included in the leaf level cost. Note that the average fitness call number in the case of GB-HGS shows only the order of the actual quantity but nothing more is available in [15]. In [15], however, one can also find results of minimizing 10D Ackley function by means of the Simple Evolutionary Algorithm (SEA). It turns out that SEA after 10^7 fitness calls approaches the minimum with the accuracy about 5, which is obviously far from the HMS's achievement.

In Rastrigin 20D we assumed the accuracy of 1000 (note that this time the number of local minima is really huge). The execution parameters are summarized in Tab. 3 (the meaning of the parameters is the same as in the Ackley

Table 1. HMS execution parameters (Ackley 10D)

	Root level	Leaf level
Population/initial population	50	10
Metaepoch length	50	-
Encoding scale η_j	4.0	1.0
Mutation rate	0.1	0.03
Crossover rate	0.5	0.5
Mutation standard deviation σ_j^{mut}	4.0	0.8
Sprout standard deviation σ_j^{sprout}	10.0	2.0
Sprout minimal distances c_j	12.0	2.4

Table 2. Average number of objective evaluations (Ackley 10D)

	Root level	Leaf level	Total
GB-HGS			10000000
HMS	147093	4340	151433

Table 3. HMS execution parameters (Rastrigin 20D)

	Root level	Leaf level
Population/initial population	50	10
Metaepoch length	50	-
Encoding scale η_j	5.0	1.0
Mutation rate	0.1	0.03
Crossover rate	0.5	0.5
Mutation standard deviation σ_j^{mut}	68.27	13.65
Sprout standard deviation σ_j^{sprout}	170.675	34.125
Sprout minimal distances c_j	204.81	40.95

Table 4. Average number of objective evaluations (Rastrigin 20D)

	Root level	Leaf level	Total
GB-HGS			10000000
HMS	194899	3570.1	198469.1

case). The fitness call statistics are gathered in Tab. 4. Note that the number of fitness calls is much higher at the root level, which is very advantageous from the point of view of inverse problem solving, because the cost of direct solution is much less then in case of leaves, because of much lower required accuracy.

Finally let us note that more thorough HMS testing should tackle real inverse problems (instead of simple benchmark functions). Such tests, involving oil exploration problems, are planned in the near future.

5 Conclusions

In the paper we have presented a memetic global optimization framework HMS. It can be used in general optimization but its main design goal is to solve inverse problems. The main benefit of the presented framework is a significant reduction of the computational cost together with the ability of the exploration of multiple extreme obtained on the several separate, but perfectly focusing ways, namely (see Sec. 2):

– self-adaptation through construction of a sophisticated deme topology;
– simultaneous error scaling;
– knowledge mining and online search profiling;
– parallel processing.

To develop these features HMS summarizes and improves ideas taken from HGS, *hp*-HGS and CGS (see Sec. 1).

The preliminary tests show the advantage of HMS over the refined hierarchic genetic strategy GB-HGS dedicated to multimodal problems (number of fitness calls decreases by two orders) as well as over the single deme evolutionary algorithm (here number of fitness call decreases even more). An additional cost decrement can be obtained by the common error scaling and the deme clustering, which were not included in the presented series of tests.

References

1. Tarantola, A.: Inverse Problem Theory. Mathematics and its Applications. Society for Industrial and Applied Mathematics (2005)
2. Engl, H., Hanke, M., Neubauer, A.: Regularization of Inverse Problems. Mathematics and its Applications, vol. 375. Springer, Heidelberg (1996)
3. Pardalos, P., Romeijn, H.: Handbook of Global Optimization (Nonconvex Optimization and its Applications), vol. 2. Kluwer (1995)
4. Chakraborty, U.K. (ed.): Advances in Differential Evolution, vol. 143. Studies in Computational Intelligence. Springer (2008)
5. Cantú Paz, E.: Efficient and accurate parallel genetic algorithms, vol. 2. Kluwer (2000)
6. Törn, A.A.: A search clustering approach to global optimization. In: Dixon, L.C.W., Szegö, G.P. (eds.) Towards Global Optimisation 2, pp. 49–62. North-Holland, Amsterdam (1978)
7. Schaefer, R., Adamska, K., Telega, H.: Genetic clustering in continuous landscape exploration. Engineering Applications of Artificial Intelligence **17**, 407–416 (2004)
8. Schaefer, R., Kołodziej, J.: Genetic search reinforced by the population hierarchy. In: Foundations of Genetic Algorithms 7, pp. 383–399, Morgan Kaufman (2003)
9. Wierzba, B., Semczuk, A., Kołodziej, J., Schaefer, R.: Hierarchical Genetic Strategy with real number encoding. In: Proceedings of the 6th Conference on Evolutionary Algorithms and Global Optimization, pp. 231–237 (2003)
10. Barabasz, B., Migórski, S., Schaefer, R., Paszyński, M.: Multi-deme, twin adaptive strategy hp-HGS. Inverse Problems in Science and Engineering **19**(1), 3–16 (2011)

11. Demkowicz, L., Kurtz, J., Pardo, D., Paszyński, M., Rachowicz, W., Zdunek, A.: Computing with hp Finite Elements II. Frontiers: Three-Dimensional Elliptic and Maxwell Problems with Applications. Chapman & Hall/CRC (2007)
12. Neri, F., Cotta, C., Moscato, P. (eds.): Handbook of Memetic Algorithms. Studies in Computational Intelligence, vol. 379. Springer, Heidelberg (2012)
13. Cetnarowicz, K., Kisiel-Dorohinicki, M., Nawarecki, E.: The application of evolution process in multi-agent world (MAW) to the prediction system. In: Tokoro, M. (ed.) Proceedings of the 2nd International Conference on Multiagent Systems (ICMAS 1996). AAAI Press (1996)
14. Byrski, A., Schaefer, R., Smołka, M., Cotta, C.: Asymptotic guarantee of success for multi-agent memetic systems. Bulletin of the Polish Academy of Sciences: Technical Sciences **61**(1), 257–278 (2013)
15. Jojczyk, P., Schaefer, R.: Global impact balancing in the hierarchic genetic search. Computing and Informatics **28**(2), 181–193 (2009)
16. Wróbel, K., Torba, P., Paszyński, M., Byrski, A.: Evolutionary multi-agent computing in inverse problems. Computer Science **14**(3), 367–383 (2013)
17. Burczyński, T., Orantek, P.: The hybrid genetic-gradient algorithm. In: Proceedings of 3rd KAEGiOG Conference, Potok Złoty, Poland (1999)
18. Grochowski, M., Smołka, M., Schaefer, R.: Architectural principles and scheduling strategies for computing agent systems. Fundamenta Informaticae **71**(1), 15–26 (2006)
19. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley (2007)