It's Fate: A Self-organising Evolutionary Algorithm

Jan Bim, Giorgos Karafotias, S.K. Smit, A.E. Eiben, and Evert Haasdijk

Vrije Universiteit Amsterdam

Abstract. We introduce a novel evolutionary algorithm where the centralized oracle –the selection-reproduction loop– is replaced by a distributed system of *Fate Agents* that autonomously perform the evolutionary operations. This results in a distributed, situated, and self-organizing EA, where candidate solutions and Fate Agents co-exist and co-evolve. Our motivation comes from evolutionary swarm robotics where candidate solutions evolve in real time and space. As a first proof-of-concept, however, here we test the algorithm with abstract function optimization problems. The results show that the Fate Agents EA is capable of evolving good solutions and it can cope with noise and changing fitness landscapes. Furthermore, an analysis of algorithm behavior also shows that this EA successfully regulates population sizes and adapts its parameters.

1 Introduction

Evolutionary algorithms (EAs) offer a natural approach to provide adaptive capabilities to systems that are by nature distributed in a real or virtual space. Examples of such systems are robot swarms that have to adapt to some dynamically changing environment, or a collection of adaptive software agents that provide services at different locations in a vast computer network. Such systems are becoming more and more important, and so is the need to make them evolvable on-the-fly. The problem is that traditional EAs with central control are not suited for these kinds of applications.

In traditional EAs we can distinguish two entities: the population of candidate solutions that undergo evolution and an omniscient oracle (the main EA loop) that decides about all individuals and performs the evolutionary operators. In situated evolution in general, and in evolutionary swarm robotics in particular, a single oracle has a number of drawbacks [11]. Firstly, it forms a single point of failure, secondly, it may limit scalability as it may not be able to process all the information about the individuals in a timely manner, and thirdly, it may not be reachable for certain individuals if the distance exceeds the feasible (or cost effective) range of communication. The natural solution would be a system with multiple, spatially distributed EA-oracles that provide sufficient coverage of the whole population. Furthermore, the inherently dynamic circumstances in such applications require that the EA-oracles can adjust their own settings on-the-fly [8].

The first objective of this paper is to introduce a system that combines two fundamental properties by design: 1) evolutionary operators are distributed and 2) the algorithmic settings are self-regulating. The key idea is to decompose the EA loop into three separate functional components, parent selection, reproduction/variation, and survivor selection, and create autonomous entities that implement these components. We name

these entities *Fate Agents* after the 'Moirai' of Greek mythology. For a maximally modular system we define three types of Fate Agents, one for each EA component, and add several Fate Agents of each type to the regular population of candidate solutions. These Fate Agents control the evolution of both regular candidate solutions and other Fate Agents in their direct surroundings. Because the Fate Agents also act on Fate Agents, the population of Fate Agents evolves itself and the EA configuration becomes adaptive.

Obviously, an elegant design does not by itself justify a new system. The second objective of this paper is an experimental assessment to answer three main questions:

1. Can this evolutionary system solve problems at all?
2. Can this evolutionary system cope with noise?
3. Can this evolutionary system cope with changing fitness landscapes?

Because the Fate Agents EA is new and has, to our knowledge, never been implemented before, we are also interested in system behavior. Therefore, we also inspect various run-time descriptors (e.g., the numbers of agents) that can help us understand what happens during a run.

2   Related Work

Existing work can be related to our research from the angles of the two main properties mentioned above: distributed, agent-based evolutionary operators and self-regulating algorithmic settings. The latter is a classic theme in EC, often labelled parameter control or on-line parameter setting [2,8]. In this context, our work is distinguished by the novel Fate Agent technique to modify EA parameters and the fact that it can handle *all* parameters regarding selection, reproduction and population size. This contrasts with the majority of related work where typically one or two EA parameters are handled. Furthermore, our system naturally handles population sizes, which is one of the toughest problems in EAs with autonomous selection [14].

The distributed perspective is traditionally treated in the context of spatially structured EAs [12], where the cellular EA variants are the closest to our system [1]. Nevertheless, there are important differences: spatially structured EAs are based on "oracles" outside the population that do not change over time, while our Fate Agents operate "from within" and –most importantly– undergo evolution themselves. The combination of spatial structure and parameter control has been studied in [5] and [4], where each location in the grid space has a different combination of parameter values. These, however, are all set by the user at initialization and do not change over time.

Finally, our system can be related to meta-evolution [3], in particular the so-called local meta-evolutionary approaches [10] (not the meta-GA lookalikes). Work in this sub-domain is scarce, we only know about a handful of papers. For instance, [10] provides a theoretical analysis, [6] demonstrates it in GP, while [9] eloquently discusses computational vs. biological perspectives and elaborates on algorithms with an artificial chemistry flavor.

3   The Fate Agents Evolutionary Algorithm

Our Fate Agents EA is situated in a (virtual) space where agents move and interact. The evolving population consists of two main types of agents: passive agents that represent

candidate solutions to the problem being solved and active Fate Agents that embody EA operators and parameters. Fate Agents form evolving populations themselves because they act not only upon candidate solutions but also upon each other. This makes the Fate Agents EA entirely self-regulated. By design, Fate Agents have a limited range of perception and action: they can only influence other agents within this range. Consequently, the evolutionary process is fully distributed as there is no central authority that orchestrates evolution but different parts of the environment are regulated by different agents. Below we describe the agent types and functionalities and subsequently the algorithm's main cycle.

Candidate Solution Agents are the simplest type of agent: they only carry a genome which represents a solution to the given problem. The fitness of a candidate solution agent is the fitness of its genome according to this problem. In a swarm robotic application, for example, we could have working robots and Fate robots; the principal problem would then be to evolve controllers for the working robots. The candidate solutions would be the controllers in the working robots encoded by some appropriate data structure and the corresponding fitness would be based on the task the working robots have to solve. To solve an abstract function optimization problem, the candidate solutions' genome would be no different from that in a regular EA, but the candidate solution would be situated in and move about a virtual space, along with the Fate Agents. In general, we assume that candidate solution agents are able to move. However, they are passive in the algorithmic sense, being manipulated by Fate Agents.

Fate Agents personify and embody evolutionary operators: parent selection, variation/reproduction and survivor selection. Fate Agents have a limited range of operation so that each one can act only within its local neighborhood. Fate Agents themselves form an evolving population, hence they require a measure of fitness. We experimented with various approaches, such as (combinations of) measures like diversity, average and median fitness; we found that the use of the best candidate solution fitness in the area yields the best results. Thus, the fitness of a Fate Agent is set to the fitness of the fittest candidate solution in its neighborhood. There are three types of Fate Agents, each responsible for different evolutionary operators: *cupids* select and pair up parents, *breeders* create offspring while *reapers* remove agents to make room for new ones. Note that they perform these operations not only on candidate solutions but on each other as well, e.g. cupids make matches between cupids, reapers kill breeders, etc.

Cupids realize parent selection by determining who mates with whom. The selection procedure is the same for all kinds of agents. A cupid creates lists of potential parents by running a series of tournaments in its neighborhood. The number of tournaments held for each type of agent depends on two values: the number of agents of that type in the cupid's neighborhood and a probability that this type of agent is selected. The latter probability is different for each distinct cupid and subject to evolution in the cupid strain. The tournament sizes also evolve. Thus, a cupid's genome consists of four real values representing the selection probabilities for each agent type and one integer for tournament size.

Reapers realize survivor selection indirectly, by selecting who dies. The selection mechanism of reapers is identical to that of cupids (with the difference that reapers' tournaments select the worst of the candidates). Reapers' genomes also consist of four

selection probabilities for the different agent types and a tournament size. In earlier versions of the algorithm we tried different mechanisms for cupids and reapers. One approach was allowing a cupid/reaper to examine each and every agent in its neighborhood and make a separate decision whether to select it or not. That selection decision was facilitated by a simple perceptron using various measures of the agent and its surroundings as input. The weights and the threshold of the perceptron evolved. We found this representation to be overly complicated and results suggested mostly random selection. A variation of the selection scheme we currently use was to also evolve the probability that the winner of a tournament would actually be selected. Results suggested that this probability had no effect, possibly because the size of the tournament already provides sufficient control of selection pressure.

Breeders realize reproduction and variation by producing a child for a given couple of parent agents. For all kinds of agents the breeder performs both recombination and mutation. Breeders, as opposed to cupids and reapers, have different mechanisms for acting upon themselves and upon other agent types. In general, a breeder is given two parents by a cupid in the neighborhood and applies averaging crossover to produce one offspring and then Gaussian/creep mutation (in our experiments) on that offspring. A breeder's genome consists of three values: the mutation step sizes for candidate solutions, cupids and reapers. Thus, mutation of these agents evolves in the breeder population. Mutation step sizes for breeders are mutated according to the following rule taken from Evolution Strategies' self-adaptation:

$$\sigma_{t+1} = \sigma_t e^{\tau N(0,1)}$$

The reason for this distinction is that if breeders' mutation step sizes were also to evolve then these values would be used to mutate themselves. Trial experiments showed that this approach leads to a positive feedback loop that results in exploding values. Note, that the implementation of the breeder depends on the application: the crossover and mutation operators must suit the genomic representation in the candidate solutions. An earlier version of the breeder was designed with the intention to control as much of the reproduction process as possible. The breeders' genome consisted of mutation rates, mutation sizes and different parameters of crossover if applicable. It also included meta-mutation values that were used to mutate the previous values. There were three layers in a breeder's genome: the lower level consisted of values involved in the variation of candidate solutions and other Fate Agents while the upper levels were used to variate the lower layers (and thus the breeders themselves). Results showed that this approach was too complex and inappropriate for evolution, especially since upper level mutation step sizes had a rather minor short-term effect on the fitness of candidate solution agents.

The Main Cycle. In the experiments for this paper, we used the Fate Agent EA to solve abstract function optimization problems, so we had to devise a virtual space and operations for movement. Obviously, applications in a swarm robotics or ALife setting would come with predefined space and movement, and parts of the cycle presented here would be superfluous.

All the agents, both candidate solutions and Fate Agents, are situated in the same spatially structured environment, a torus shaped grid. Each cell can either be empty or occupied by exactly one agent. The algorithm makes discrete steps in two phases.

The first phase takes care of movement. For these experiments, we have simply implemented random movement by randomly swapping contents between two neighboring cells with a certain probability.

In the second phase evolutionary operators are executed. First, both cupids and reapers make their selections. Subsequently, offspring is produced in iterations as follows: in each iteration, a cupid with available parents and a free cell in its neighborhood is randomly chosen. A breeder is randomly selected from the neighborhood of the selected cupid and it is provided with the parents that are then removed from the cupid's list. The breeder produces a single child which is then placed in the empty cell. This procedure is repeated until there are no cupids with remaining selected agents and unoccupied cells in their neighborhood.

When offspring production has completed, reaping is performed: reapers are activated in random sequence until there are no reapers left with non-empty selection lists. Notice that during each reaping iteration, a reaper kills only one agent (of any type). Hence, a reaper can kill and be killed in the same reaping iteration. When reaping is complete, the evolutionary phase is concluded and the algorithm starts a new cycle. The overall algorithm cycle is presented in Algorithm 1.

The random sequence and individual actions of cupids and reapers during offspring production and reaping approximate a distributed system with agents acting autonomously and concurrently. It might seem unorthodox that selection by the reapers is performed before offspring are produced, meaning that unfit offspring are allowed to survive and possibly reproduce. Our motivation for this order of operators is to give Fate Agents a 'free pass': before a Fate Agent is considered for removal it should have the chance to act upon its neighborhood at least once, so that its evaluation will closer reflect its true fitness. The designed order does give this free pass to Fate Agents (at least to cupids and breeders).

4   Experimental Setup

We conducted several experiments to validate our algorithm from a problem solving perspective and to observe its runtime behavior. To this end we used the test functions from the BBOB2012 test suite from the GECCO 2012 black box optimization contest, because they are well designed and proven by several other research groups [1]. Furthermore, we experimented on the Fletcher & Powell function, because it is very hard to solve but its landscape can be easily redefined for the tests on changing fitness functions. We performed three sets of experiments:

A As a proof-of-concept that the algorithm generally works and is capable of problem solving and self-regulation we used 6 functions: BBOB2012 f_3, f_{20}, f_{22}, f_{23}, f_{24}, and the Fletcher & Powell. We allowed the algorithm to run for 500 generations.

B To see if the algorithm can cope with noise we used 6 other BBOB2012 functions: f_{122}, f_{123}, f_{125}, f_{126}, f_{128}, f_{129} and let the algorithm run for 1000 generations.

C To examine how well the system can recover from and adapt to sudden cataclysmic changes we ran tests on the Fletcher & Powell function randomizing its matrices every 250 generations. Here we allowed the algorithm to run for 2000 generations.

---

[1] `http://coco.gforge.inria.fr/doku.php?id=bbob-2012`

Algorithm 1. The Fate Agent EA algorithm

```
generation ← 0;
while generation ≤ maxGeneration do
    doMovement;
    for all Cupid c do
        c.SelectParents;
        cupids.Add(c);
    end for
    for all Reaper r do
        r.SelectDeaths;
        reapers.Add(r);
    end for
    while cupids.NotEmpty do
        c ← cupids.GetRandomCupid;
        if c.HasNoFreeCell ∨ c.SelectedParents.Empty then
            cupids.Remove(c);
        else
            b ← c.GetRandomNeighborBreeder;
            cell ← c.GetRandomFreeCell;
            a ← b.Breed(c.GetParents);
            cell.placeAgent(a);
        end if
    end while
    while reapers.NotEmpty do
        r ← reapers.GetRandomReaper;
        if r.agentsToKill.Empty then
            reapers.remove(r);
        else
            r.killAgent;
        end if
    end while
    generation ← generation + 1;
end while
```

For the spatial embedding we used a 100×100 grid where each cell can either be empty or occupied by only one agent (thus there is a maximum of 10000 agents). The grid is initialized by filling all cells with random agents of random type with the following probabilities: 0.0625 for each Fate Agent type and 0.8125 for candidate solutions. Random movement is implemented by swapping the contents of two neighboring cells with probability 0.5 for each edge. Fate Agents have a neighborhood of radius 5 cells. All our experiments are repeatable, since we offer the code of the Fate Agent EA and the experiments through the webpage of the second author. [2]

We emphasize that the purpose of these experiments is not to advocate the Fate Agents EA as a competitive numeric optimizer but only to demonstrate its ability to

---

[2] See `http://www.few.vu.nl/~gks290/downloads/PPSN2012Fate.tar.gz` for the whole source code.

solve problems and investigate its dynamics and self-regulating behavior without time consuming robotics or ALife experiments. The numeric test suite was used merely as a convenient testbed for evolution, thus a comparison with benchmarks or BBOB champions would be out of context.

5 Does It Work?

The results in section A in Table 1 show that the Fate Agents EA is indeed able to solve problems, achieving good fitness on almost all BBOB test functions and a reasonably high fitness for the very difficult FP problem. Good success ratios are also achieved for two noiseless and three noisy functions.

Section B of Table 1 demonstrates that our system is able to cope with noise very well: it achieves high fitness for all problems and a good success ratio for three out of six noisy functions. As was explained in Section 4 the purpose of the experiments is not to propose the Fate Agents EA as a numeric optimizer, thus, we will not examine its performance on BBOB any further or determine how competitive it is.

Fig. 1. Example of system recovery, fitness vs. time, experiment C, run 4

Finally, based on the results of experiment set C, we can give a positive answer to the third question we posed in Sec. 1: Fig. 1 presents the best fitness over time for an example run. The sudden drops in fitness mark the points in time when the matrix of the FP problem is randomly changed, drastically changing the fitness landscape. As can be seen, the system recovers from this catastrophic change, although it does not always succeed. In general, 24 out of 30 runs exhibited at least one successful recovery while, in total, we observed an equal number of successful and unsuccessful recoveries. It should be noted that the FP problem is very hard and the time provided between changes is quite short (250 generations). Nevertheless, results show that the Fate Agent EA does possess the ability to cope with radical change even though the design has no specific provisions for that purpose.
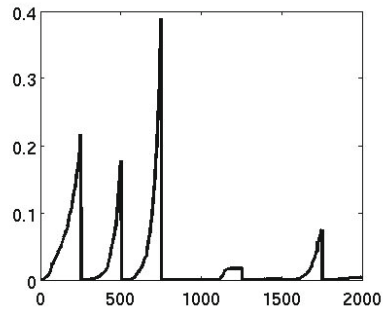
Table 1. Performance results for experiment sets A and B in terms of Average Best Fitness normalized between 0 (worst) and 1 (optimal), Average number of Evaluations to Best fitness achieved, Success Rate (success at 0.999 of normalized fitness) and Average number of Evaluations to Success (only successful runs taken into account)

Set A - Static Noiseless					Set B - Static Noisy			
	ABF	AEB	SR	AES		ABF	AEB	SR	AES
F&P	0.53	289682	0.0	-	f_{122}	0.99	207857	0.03	240684
f_3	0.87	223341	0.76	228440	f_{123}	0.99	130996	0.96	131425
f_{20}	0.73	210527	0.03	266631	f_{125}	0.99	131383	0.0	-
f_{22}	0.90	309865	0.8	328891	f_{126}	0.99	126342	1.0	126342
f_{23}	0.90	204382	0.0	-	f_{128}	0.97	172836	0.40	195561
f_{24}	0.05	247797	0.0	-	f_{129}	0.99	137674	0.76	135225

6   System Behavior

One of the most important aspects of the system's behavior is that the population sizes for the different types of agents are successfully regulated. They reach a balance quite different to the initialization ratios (see Section 4) very quickly while no agent type becomes extinct or dominates the population even though there are no external limitations imposed. This is a very interesting result on its own right, since regulating population sizes in EAs with autonomous selection is an open issue [14]. An example run is shown in Fig. 2. Agent numbers are initialized to default values but populations soon converge and maintain a balance throughout the run. All runs across experiment sets demonstrate similar population dynamics.

Considering self-adaptation in EAs, one of the basic expectations is the adaptation of the size of search steps (the mutation step size σ in terms of evolution strategies). In our system, the mutation of agents is controlled by the breeder agents. The breeders' genome includes mutation step sizes for every other agent type. Fig. 3 presents examples of three different behaviors observed in breeder populations. Each graph illustrates the best and mean fitness of the candidate solution population and the mutation step size applied to candidate solutions averaged over the whole

Fig. 2. Example of population breakdown over time, experiment A f_{22}, run 12

breeder population. Case (a) is an example of typical evolution with mutation size slowly converging to zero as the search converges to the optimum. Case (b) demonstrates a successful response of the breeders to premature convergence to a local optimum: after around 100 generations the search gets stuck and the breeder population reacts with a steep increase of the mutation size which helps escape and progress. Case (c) shows a failed attempt to escape a local optimum, even though breeders evolve high mutation sizes after the search is stuck.

Note that mutation sizes are correlated to the average fitness, not to the best fitness. This is reasonable considering that Fate Agents have a limited range and are unaware of global values. In all cases, the mutation size converges to zero as soon as the whole population converges (mean fitness becomes equal to the best fitness). This implies that the system has the ability to respond and escape local optima as long as there is still diversity available but is unable to create new diversity after global convergence, as is the case in Fig. 3(c).

Finally, we made an interesting observation related to spatial dynamics: results show that, on average, cupids consistently have better fitness than reapers. Both agent types are evaluated according to the fittest candidate solution in their neighborhood and both agent types have the same range for these neighborhoods. We conclude that reapers are usually found in areas with less fit individuals while cupids frequent areas with fitter individuals. Since movement is random, this effect can only be the result of cupids' selection probabilities: cupids in 'bad' areas consistently evolve a preference for selecting reapers while cupids in 'good' areas develop a preference for selecting even more
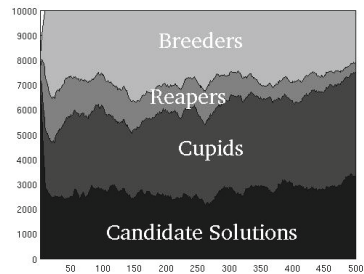
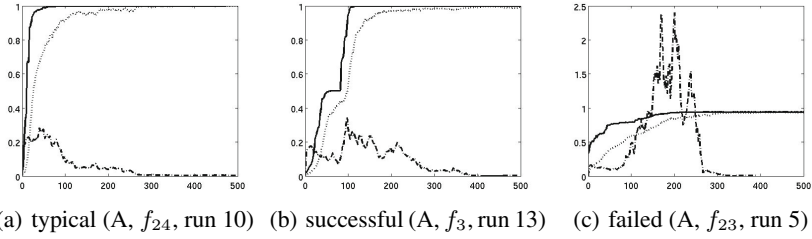(a) typical (A, f_{24}, run 10)   (b) successful (A, f_3, run 13)   (c) failed (A, f_{23}, run 5)

Fig. 3. Three examples of breeders' evolution and response to premature convergence. Lines represent best fitness (solid), mean fitness (dotted) and mutation step size for candidate solutions (dashed) over time.

cupids. Furthermore, reapers almost always have very high preference for killing other reapers and, consequently, low ages (they mostly survive only one generation).[3] This implies that cupids in bad areas create 'reaper explosions' that eradicate low-fitness candidate solutions and also clean up after themselves as reapers select each other.

7 Conclusions and Future Work

Motivated by specific challenges in evolutionary swarm robotics, we introduced the Fate Agents Evolutionary Algorithm. It forms a new type of distributed and self-regulating EA where algorithmic operators are implemented through autonomous agents placed in the same space where the candidate solutions live and die. This provides a natural solution to the problems of the limited range and scalability a single oracle based EA would suffer from. Compared to alternative solutions where evolutionary operators are embodied in the robots [13,7] Fate Agents offer increased controllability for experimenters and users. Furthermore, our Fate Agents are not only operating on candidate solutions, but also on themselves. Hence, a Fate Agents EA has an inherent capability to regulate its own configuration.

Because proof-of-concept experiments with (simulated) robots would have taken very much time, we performed the first assessment of this new EA with synthetic fitness landscapes. To this end, we conducted experiments to explore our system's problem solving ability and self-regulating behavior on challenging numerical optimization problems. Results showed that the Fate Agents EA is capable of solving these problems and of coping with noise and disruptive changes. Furthermore, it successfully regulates population sizes and adapts its parameters.

In conclusion, the Fate Agents EA is a new kind of evolutionary algorithm that deserves further research from a number of angles. These include 1) applications in collective adaptive systems, such as swarm and evolutionary robotics (comparisons with other on-line on-board evolutionary mechanisms); 2) as a new paradigm for self-adapting EAs . Furthermore, though it may seem contradictory to our initial motivation, our results indicate that the Fate Agents EA may also deserve further investigation as a numeric function optimizer.

---

[3] These observations are true for almost every run we conducted. Due to lack of space we cannot present relevant graphs.

References

1. Alba, E., Dorronsoro, B.: Cellular Genetic Algorithms. Springer (2008)
2. Eiben, A., Michalewicz, Z., Schoenauer, M., Smith, J.: Parameter control in evolutionary algorithms. In: Lobo, et al. [8], pp. 19–46
3. Freisleben, B.: Meta-evolutionary approaches. In: Bäck, T., Fogel, D., Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, pp. 214–223. Institute of Physics Publishing, Oxford University Press, Bristol, New York (1997)
4. Gong, Y., Fukunaga, A.: Distributed island-model genetic algorithms using heterogeneous parameter settings. In: IEEE Congress on Evolutionary Computation, pp. 820–827 (2011)
5. Gordon, V.S., Pirie, R., Wachter, A., Sharp, S.: Terrain-based genetic algorithm (TBGA): Modeling parameter space as terrain. In: Banzhaf, W., Daida, J., Eiben, A., Garzon, M., Honavar, V., Jakiela, M., Smith, R. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 1999, pp. 229–235. Morgan Kaufmann, San Francisco (1999)
6. Kantschik, W., Dittrich, P., Brameier, M., Banzhaf, W.: Meta-Evolution in Graph GP. In: Langdon, W.B., Fogarty, T.C., Nordin, P., Poli, R. (eds.) EuroGP 1999. LNCS, vol. 1598, pp. 15–28. Springer, Heidelberg (1999)
7. Karafotias, G., Haasdijk, E., Eiben, A.: An algorithm for distributed on-line, on-board evolutionary robotics. In: Krasnogor, N., Lanzi, P.L., Engelbrecht, A., Pelta, D., Gershenson, C., Squillero, G., Freitas, A., Ritchie, M., Preuss, M., Gagne, C., Ong, Y.S., Raidl, G., Gallager, M., Lozano, J., Coello-Coello, C., Silva, D.L., Hansen, N., Meyer-Nieberg, S., Smith, J., Eiben, G., Bernado-Mansilla, E., Browne, W., Spector, L., Yu, T., Clune, J., Hornby, G., Wong, M.-L., Collet, P., Gustafson, S., Watson, J.-P., Sipper, M., Poulding, S., Ochoa, G., Schoenauer, M., Witt, C., Auger, A. (eds.) Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011, Dublin, Ireland, July 12-16, pp. 171–178. ACM (2011)
8. Lobo, F., Lima, C., Michalewicz, Z. (eds.): Parameter Setting in Evolutionary Algorithms. Springer (2007)
9. Nellis, A.: Meta evolution. Qualifying Dissertation (2009)
10. Samsonovich, A.V., De Jong, K.A.: Pricing the 'free lunch' of meta-evolution. In: Beyer, H.-G., O'Reilly, U.-M. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005, pp. 1355–1362. ACM (2005)
11. Schut, M., Haasdijk, E., Eiben, A.E.: What is situated evolution? In: Proceedings of the 2009 IEEE Congress on Evolutionary Computation, Trondheim, May 18-21, pp. 3277–3284. IEEE Press (2009)
12. Tomassini, M.: Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Natural Computing Series. Springer-Verlag New York, Inc., Secaucus (2005)
13. Watson, R.A., Ficici, S.G., Pollack, J.B.: Embodied evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems 39(1), 1–18 (2002)
14. Wickramasinghe, W., van Steen, M., Eiben, A.E.: Peer-to-peer evolutionary algorithms with adaptive autonomous selection. In: D.T., et al. (eds.) Proc of the 9th conference on Genetic and Evolutionary Computation, GECCO 2007, pp. 1460–1467. ACM Press (2007)