ACO Beats EA on a Dynamic Pseudo-Boolean Function

Timo Kötzing[1] and Hendrik Molter[2]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
[2] Saarland University, Saarbrücken, Germany

Abstract. In this paper, we contribute to the understanding of the behavior of bio-inspired algorithms when tracking the optimum of a dynamically changing fitness function over time. In particular, we are interested in the difference between a simple evolutionary algorithm (EA) and a simple ant colony optimization (ACO) system on deterministically changing fitness functions, which we call *dynamic fitness patterns*. Of course, the algorithms have no prior knowledge about the patterns.

We construct a bit string optimization problem where we can show that the ACO system is able to follow the optimum while the EA gets lost.

1 Introduction

Bio-inspired algorithms are an important class of randomized search heuristics, valued for their easy applicability also in challenging environments. In particular, environments with uncertainty are settings difficult for problem specific algorithms, while bio-inspired algorithms can more easily deal with the uncertainty.

Jin and Branke [9] discuss different sources of uncertainty and surveys the literature evolutionary algorithms in uncertain environments. Two important sources are a *noisy* fitness function and a *dynamic* fitness function. In the case of a noisy fitness function, the fitness of a search point follows a random distribution, which is the same distribution at each evaluation of the fitness. In the case of a dynamic fitness function, the fitness of a search point at any given time is a deterministic value, but this value changes over time.

Bio-inspired algorithms seem to be very robust against noise and dynamic changes which makes them very popular for these two optimization problem classes. Two prominent types of bio-inspired algorithm are *Evolutionary Algorithms* (EA) and *Ant Colony Optimization* (ACO) systems.

In this paper we are concerned with *dynamic* fitness functions; the goal for an algorithm will be to track the optimum of a dynamically changing fitness function over time. We consider the setting of *pseudo-Boolean fitness functions*, where the search space is given as all bit strings of a fixed length n and the fitness of any search point is a real value.

Some results about *evolutionary* algorithms tracking optima in pseudo-Boolean optimization are given in [1,5,6,15,8]. Some of these papers analyze

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 113–122, 2012.

settings where the optimum changes *randomly*; we are more interested in how bio-inspired algorithms can *track* an optimum that changes *deterministically* (but, of course, the algorithm has no prior knowledge of what changes will happen). We believe that an analysis in such settings on certain dynamic patterns gives a better view on the strengths of bio-inspired algorithms than its performance on an optimum which moves away in a random direction – those changes are hard to control, due to their inherent randomness and the many choices for the optimum to evade in our search space.

In this paper, we try to gain new insights in the differences between the behavior of EA and ACO on these optimization problems; more specifically, we analyze the *(1+1)-EA* and a version of the *Max-Min-ant system* (*MMAS*, introduced in [16]) given in Section 2. We compare the performance of these algorithms on a dynamic optimization problem where we can show that *MMAS* is able to follow the optimum while the *(1+1)-EA* gets lost (see Section 3.1).

The theoretical analysis of these algorithms is very challenging; therefore we use simple dynamic optimization problems, derived from the *OneMax* problem; this approach was also taken in some of the previous literature for the *(1+1)-EA*. There are many analyses of *MMAS* on *static* variants of OneMax (see, for example, [13,7,4,11,12,10,17]), but so far there is no theoretical work on the performance of ACO algorithms on dynamic problems.

We introduce the notion of a *dynamic fitness pattern*. Instead of considering random movements of the optimum, and analyzing an algorithms performance at tracking this random behavior, we are interested in how the *(1+1)-EA* and *MMAS* behave in settings with more structure.

We construct a "maze", in which the *(1+1)-EA* cannot track the optimum and will get lost with high probability. On the other hand, *MMAS can* track the optimum, also with high probability. Key to the success of *MMAS* is the use of intermediate pheromone values, which serve as a medium-term memory: pheromones build up according to solutions that have been good *recently*, extracting *trends* in the fitness pattern. The *(1+1)-EA* misses these trends and, instead, oscillates between different good solutions.

We use *drift analysis* as our key tool to analyze the behavior of the algorithms; in particular, we use many drift theorems from the literature, including a drift theorem with tail bounds from [2], which allows us to derive high probability results.

In general, high probability results are sparse in the analysis of bio-inspired algorithms; a notable recent exception is [17], where many tail bounds are given.

We proceed as follows. In Section 2 we introduce all necessary mathematical definitions. In Section 3 we introduce and analyze a maze where *MMAS* manages to follow the optimum and *(1+1)-EA* loses track of it. We conclude in Section 4.

2 Mathematical Preliminaries

In this section we introduce all formal definitions that we will use in the analysis.

Our general problem setting is *pseudo-Boolean function* optimization. The solution space is the set of all bit strings of length n (for fixed n). We let $h(x, y)$

denote the *Hamming distance* between two bit strings x and y (the number of bits where x and y disagree).

One of the simplest functions for static optimization problems is the OneMax function. This function has proven to be very useful for theoretical analysis of evolutionary algorithms. For any bit string x of length n we let

$$\text{OneMax}(x) = n - h(x, 1^n).$$

The OneMax function is maximized by the all-ones bit string and, importantly, the fitness of a bit string x is better the closer x is to the all-ones string, i.e., the more 1s are in x.

Next we formally introduce the *(1+1)-EA* and *MMAS*.

The *(1+1)-EA* is depicted in Algorithm 1. It keeps a current-best solution x and, in each iteration, mutates it by flipping each bit independently with some mutation probability. This mutation probability is typically $1/n$, where n is the number of bits in the bit string.

If the mutant has a better current fitness than the current fitness of x (x is reevaluated), then x is replaced with its mutant, otherwise x is kept and the mutant discarded.

Algorithm 1. (1+1)-EA

```
1 initialize x;
2 repeat
3 |   x' ← mutate(x);
4 |   if fitness(x') ≥ fitness(x) then
5 |   |   x ← x';
6 until forever;
```

The second algorithm we analyze is an *Ant Colony Optimization* (ACO) system. Here ants drop pheromones on the bits and these serve as a probability distribution to construct new solutions. More specifically, we look at a Max-Min-ant system (*MMAS*, [16]). *MMAS* uses maximum and minimum values for the pheromone values (typically $1 - 1/n$ as maximal and $1/n$ as minimal pheromone values).

MMAS also maintains a current-best solution, initialized to a random bit string x. The pheromone value τ_i, for each bit i, is initialized to $1/2$.

In each iteration, we construct a new solution x' by drawing a new bit string where each bit x'_i is 1 with probability τ_i. Then we compare the fitness of this string to the reevaluated fitness of our current-best solution and replace the current-best solution if the new solution is better. Then we update the pheromone values using the current-best solution.

Note that, as long as no better solution is constructed, the pheromones are updated with always the same current-best solution over and over again until the pheromones hit the respective limits.

Algorithm 2. MMAS

```
initialize x, τ;
repeat
    x' ← constructSolution(τ);
    if fitness(x') ≥ fitness(x) then
        x ← x';
    τ ← update(τ, x);
until forever;
```

The update step works as follows. For each bit x_i, we update the respective pheromone value τ_i to a value τ_i' as follows.

$$\tau_i' = \begin{cases} \min\left\{(1-\rho)\tau_i + \rho, 1 - \frac{1}{n}\right\}, & \text{if } x_i = 1; \\ \max\left\{(1-\rho)\tau_i, \frac{1}{n}\right\}, & \text{if } x_i = 0. \end{cases}$$

When a pheromone value hits a threshold, we call it *saturated*.

3 The Maze

In this section we consider bit strings of any given length n; the mutation probability of the *(1+1)-EA* is set to $1/n$, and the pheromone bounds of *MMAS* are $1/n$ and $1 - 1/n$, respectively.

3.1 Definition of the Maze

We give the following definition of a dynamic fitness pattern where *MMAS* can track the optimum while *(1+1)-EA* gets lost. The idea behind the pattern is, that we start with OneMax, but then let each bit oscillate one after another with a 001-oscillation and then set it to zero. We set the fitness of the optimal string to $n+2$, the fitness of the string, where only the oscillating bit is flipped, to $n+1$, and for the rest we use OneMax. Intuitively, when in an oscillation phase there are strictly more 0s than 1s, *MMAS* will converge (in pheromone) to 0, while the *(1+1)-EA* will oscillate with its current best solution between the different optima. Note that any oscillation pattern that contains more 1s than 0 will not be trackable by *MMAS* (and neither by the *(1+1)-EA*).

We will see in Section 3.2 that *MMAS* is able to track down all the zeros, because the pheromone values of the oscillating bit drop down to the lower border; however, as we will see in Section 3.3, the *(1+1)-EA* loses the zero with probability at least $1/4$ each time we move the oscillating bit one step. Then *(1+1)-EA* falls back to OneMax and is not able to find the optimum again, since it the fitness function leads to the all-ones string.

Formally, we define the dynamic fitness pattern as follows. Let $\mathrm{opt}_{001}(t)$ denote a function that equals 1 at all times t which have a remainder of 2 when divided by 3, and 0 otherwise.

Definition 1 (Maze). Let $\diamond$ denote bit string concatenation, $k > 0$ and let

$$f(i,t) = 0^i \diamond \mathrm{opt}_{001}(t) \diamond 1^{n-i-1},$$

$$f'(i,t) = 0^i \diamond (1 - \mathrm{opt}_{001}(t)) \diamond 1^{n-i-1}$$

define two bit strings of length n, where i determines the position of the oscillating bit and t the time point of the oscillation. These two bit strings will be the optimal and second best solution. Let $t_0 = kn^3 \log(n)$. For any bit string of length n and any t, we define

$$\mathrm{Maze}_k(x,t) = \begin{cases} n+2, & \text{if } x = f(\lfloor \frac{t-t_0}{t_0} \rfloor, t) \\ & \quad \text{and } t > t_0; \\ n+1, & \text{if } x = f'(\lfloor \frac{t-t_0}{t_0} \rfloor, t) \\ & \quad \text{and } t > t_0; \\ \mathrm{OneMax}(x), & \text{otherwise.} \end{cases}$$

We will choose a suitable constant k later. Note that we start the oscillation of the first bit after an initial phase of t_0 iterations, after which each oscillation takes t_0 iterations. The initial phase ensures that both *MMAS* and the *(1+1)-EA* will have found the all-ones bit string as their current-best solution before the oscillations start; the oscillation is, as we will see, long enough for *MMAS* to track the optimum.

3.2 MMAS on Maze

In this section we will show that *MMAS* can track the optimum of the maze with high probability. We start by showing that, during the oscillation phase of a bit between two 1s and 0, *MMAS* will drift towards the 1-bit

Lemma 2. Suppose $\rho \in \Theta(1/n)$. For all $c > 0$, during a single bit 110-oscillation, where in every iteration we construct a new solution with some probability p converging to $1/e$ and update pheromones with the current-best solution otherwise, the pheromone value is saturated as $1 - 1/n$ in $\mathcal{O}(n^3 \log(n))$ iterations with probability $1 - \mathcal{O}(n^{-c})$.

Proof. First we calculate the expected values for the pheromone value and current-best solution after one oscillation. We use the potential function $g(\tau, x) = \tau + q\rho x$, where q is a constant. We set this constant to $q = \frac{7}{2}$.

We begin with the $x = 1$ case, where we get

$$E[\tau'] = \tau\rho(p - 3 \pm \mathcal{O}(\rho)) + \rho(3 - p) + \tau \pm o(\rho),$$

$$E[x'] = \tau(p \pm \mathcal{O}(\rho)) + 1 - p \pm \mathcal{O}(\rho).$$

For the potential difference, we get

$$E[g(\tau', x')] - g(\tau, x) = \tau\rho(p - 3 + qp \pm \mathcal{O}(\rho)) + \rho(3 - p + qp) \pm o(\rho).$$

For $q = \frac{7}{2}$ and p close enough to $1/e$, this yields a drift in $\Omega((1-\tau)\rho)$.

For the $x = 0$ case, we get

$$E[\tau'] = \tau^3\rho(-p^3 \pm \mathcal{O}(\rho)) + \tau^2\rho(p^2 \pm \mathcal{O}(\rho)) + \tau\rho(5p - 2p^2 - 3 \pm \mathcal{O}(\rho)) + \tau,$$

$$E[x'] = \tau^3(-p^3 \pm \mathcal{O}(\rho)) + \tau^2(p^2 - p^3 \pm \mathcal{O}(\rho)) + \tau(2p - 2p^2 \pm \mathcal{O}(\rho)).$$

For the potential difference, we get

$$E[g(\tau', x')] - g(\tau, x) = \Omega(\tau\rho(5p + q2p - 2p^2 - q2p^2 - 3)).$$

For $q = \frac{7}{2}$ and p close enough to $1/e$, this yields a drift in $\Omega(\tau\rho)$.

Thus we have an overall drift of $\Omega(\min\{\tau\rho, (1-\tau)\rho\})$. In particular, we have a uniform additive drift of $\Omega(\rho/n)$. Using the drift theorem with tail bounds from [2, Theorem 1], we obtain the result. Note that this wastes a lot, as the drift theorem with tail bounds is intended for use with multiplicative drift, while we use it on additive drift; also, the statement of the theorem requires a (1+1)-EA, but the proof, published in [3, Theorem 5], shows that this requirement is unnecessary. □

Note that, for the maze, will use Lemma 2 symmetrically for drifting *down* with the pheromone. We will choose k large enough so that during the oscillation of a bit, the pheromone value of the oscillating bit will at least once be $1/n$. Additionally, we have to make sure that the pheromone value stays low.

Lemma 3. Suppose $\rho \in \Theta(1/n)$. Let $k \geq 1$ and let i be the position of the oscillating bit of Maze_k. If the pheromone value τ_i is saturated at the lower threshold, for all $c > 0$, it will stay below $(\log(n))^2/n$ for at least $kn^3\log(n)$ iterations with probability $1 - \mathcal{O}(n^{-c})$.

Proof. This can be seen with a simple rescaling of the search space and an application of the drift theorem concerned with negative drift from Oliveto and Witt [14].

Note that the largest possible jump away from the lower threshold is 3ρ, since in every update the pheromone value is changes by at most ρ and we update 3 times per oscillation. After rescaling of the search space by $1/\rho$, we see that probability to make jumps larger than 3 is 0, so that the theorem from [14] easily applies. □

Another important ingredient for analyzing *MMAS* on the maze, we have to show that *MMAS* does not get lost when the oscillation switches from bit i to bit $i+1$. This can only happen, if in the last iteration of oscillating i, the ith bit of the current-best solution is 1. This happens with probability at most $1/n$, so we expect it to happen once in the run of the maze.

We show that, in this case, *MMAS* is able to regain track of the optimum again with high probability. W.l.o.g. we show this behavior for the case that all pheromones are saturated at the upper border, i.e. at $1 - 1/n$.

Lemma 4. Suppose $\rho \in \Theta(1/n)$ and the fitness function is $f(x) = n - \text{OneMax}(x)$ (this implies that the all-zeros string is the optimum of f). Furthermore, assume the current-best solution is $1^{n-1}0$ and the first $(n-1)$ pheromones are saturated at $1-1/n$, while the last pheromone value is at $1-O((\log(n))^2/n)$. Then we have, for all $c > 0$, *MMAS* samples 1^n within $\mathcal{O}(\log(n))$ iterations with probability $1 - \mathcal{O}(n^{-c})$.

Proof. Let $c > 0$, and let $t = 3c\log(n)$. We show that *MMAS* samples 1^n within t iterations with probability $1 - \mathcal{O}(n^{-c})$. Within these t iterations, *MMAS* will change more and more bits in its current-best solution to 0. For this to happen, *MMAS* needs to, in some iteration, sample one of the bits with pheromone $1 - 1/n$ as 0. For all $i < t$, let X_i be the number of bits that *MMAS* samples as 0 in iteration i, and that where not sampled as 0 in an earlier iteration. Clearly, for all $i < t$, $E(X_i) \leq 1$. Let $X = \sum_{i=1}^{t} X_i$ be the total number of different bits sampled as 0 within the first t iterations.

Using a Chernoff bound we see that

$$P(X > 3t) \leq n^{-c}.$$

Let $k = 3t = 9c\log(n)$. The following probabilities are conditional on $X \leq 3t = k$.

For the case that we lower the pheromone value of a bit, we lower it by at most ρ. Hence, within t iterations, at most k new bits have a pheromone lower than $1 - 1/n$, by at most $t\rho = o(1)$; the last bit, after t iterations, has pheromone $1 - O((\log(n))^2/n + t\rho)$. Thus, those $k + 1$ bits have a pheromone of $1 - \text{polylog}(n)/n$.[1] All other bits still have maximal pheromone. Hence, the probability to sample 1^n in *any particular* of the t iteration is at least

$$(1 - 1/n)^{n-k-1} \cdot (1 - \text{polylog}(n)/n)^{k+1}.$$

Using Bernoulli's inequality, we can lower bound this probability with

$$\frac{1}{e} \cdot (1 - (2k)\text{polylog}(n)/n) = \frac{1}{e} \cdot (1 - \text{polylog}(n)/n).$$

Now we bound the probability that we do *not* sample 1^n in *any* of the t iterations from above with

$$(1 + \frac{1}{e}(-1 + \text{polylog}(n)/n))^t.$$

We can upper bound this probability by

$$\exp\left(\frac{t}{e}(-1 + \text{polylog}(n)/n)\right) \leq \mathcal{O}(n^{-c}).$$

We now have two chances for *MMAS* to fail to sample 1^n in the first t iterations: either by lowering pheromone on more than k bits, or by failing to sample 1^n

[1] With polylog(n) we denote the set of all functions bounded above by $c\log(n)^d$, for some $c, d > 0$.

conditional on not having decreased pheromones on more than k bits. Both failure probabilities are below n^{-c}; thus, *MMAS* samples 1^n in t iterations with probability $1 - \mathcal{O}(n^{-c})$. □

The following theorem is taken from [17, Corollary 5] and gives high probability bounds for *MMAS* optimizing OneMax.

Theorem 5 ([17]). For all $c > 0$, *MMAS* optimizes OneMax in time $\mathcal{O}(n \log(n)/\rho)$ with probability $1 - \mathcal{O}(n^{-c})$.

We are now ready to give the central theorem of this paper.

Theorem 6. Suppose $\rho \in \Theta(1/n)$. For all $c > 0$ there is a k such that, for n large enough, *MMAS* can follow the optimum[2] of the dynamic fitness pattern Maze_k with probability $1 - \mathcal{O}(n^{-c})$.

Proof. Let $c > 0$. Let k' be the largest implicit constant of the runtime bounds of Lemma 2 and 4, as well as Theorem 5, for obtaining a failure probability of $\mathcal{O}(n^{-c-1})$. Let $k = 3k'$.

Since we run OneMax for the first $kn^3 \log(n)$ iterations, we know by Theorem 5 that *MMAS* finds the optimum and all pheromone values are saturated before the oscillations starts with probability $1 - \mathcal{O}(n^{-c-1})$. During the oscillation of any bit, by (the symmetric version of) Lemma 2, we lower the pheromone of the oscillating bit to $1/n$ with probability $1-\mathcal{O}(n^{-c-1})$ after one third the oscillation of the bit.

Using Lemma 3, the pheromone of an oscillating bit during its oscillation will never be further away than $\log(n)^2/n$ with probability $1-\mathcal{O}(n^{-c-1})$. Thus, when the next bit starts oscillating, by Lemma 4, *MMAS* needs at most one third the oscillation to sample again a solution that has fitness $n + 1$ or $n + 2$ with probability $1 - \mathcal{O}(n^{-c-1})$. *MMAS* can now recover all pheromones to the proper borders within another third of the oscillation and after this, the process repeats.

MMAS loses the optimum or does not saturate the pheromone value of the oscillating bit by the end of the oscillation with probability n^{-c-1}.

By induction and the union bound for the failure probabilities, *MMAS* follows the optimum of Maze_k with probability $1 - \mathcal{O}(n^{-c})$. □

3.3 (1+1)-EA on Maze

Now we take a look at the *(1+1)-EA*. Because of space limitations, we omit the proofs of this section.

Consider the oscillation of bit i; we will show that the expected value of bit i of the current-best solution of the *(1+1)-EA* is at least $\frac{1}{4}$.

[2] The algorithm is said to "follow the optimum" if, at the end of all phases, the distance of the current-best solution to the current optimum is constant.

Lemma 7. Let $k \geq 1$; consider the *(1+1)-EA* optimizing Maze_k during the oscillation of bit i, and suppose the current-best solution of the *(1+1)-EA* has a 0 on all positions before and at the oscillating bit, and a 1 on all others. For all t, let X^t be the random variable denoting the value of bit i of the current-best solution t iterations later. Then, for all $t \geq n \log(n)$, $E[X^t] \geq 1/4$.

Now we show that *(1+1)-EA* loses track of the optimum with high probability.

Theorem 8. For all $c > 0$ and $k \geq 1$, the *(1+1)-EA* loses track of the optimum of Maze_k with probability $1 - \mathcal{O}(n^{-c})$.

4 Conclusions and Future Work

We have given an example of a dynamic fitness pattern where *MMAS* is able to track the optimum with high probability, while the *(1+1)-EA* loses the optimum with high probability. This shows that there are instances where *MMAS* performs strictly better than the *(1+1)-EA*.

The intuition behind this difference is as follows. Consider a singe bit position, where the optimizing algorithm is supposed to find out whether a 0 or a 1 is better at that position. Suppose none of the other positions influence the fitness of the bit string. For an oscillating optimum at that bit position, the *(1+1)-EA*, in each iteration, will give a definite statement about what bit it thinks to be better (whatever bit the best-so-far bit string has at that position), while the *MMAS* uses intermediate pheromone values as a medium-term memory. This enables *MMAS* to "average" over the phases of oscillation, smoothing out the dynamic fitnesses of the search points, while the *(1+1)-EA* reacts strongly, too strongly, in each iteration.

As one line of future work it would be interesting to see how evolutionary algorithms with a large population size handle similar dynamic problems.

More importantly, though, future research goals should include finding results for more general classes of dynamic fitness patterns. In particular, dynamic combinatorial optimization might offer interesting settings for the analysis of the performance of bio-inspired algorithms.

References

1. Chen, T., Chen, Y., Tang, K., Chen, G., Yao, X.: The impact of mutation rate on the computation time of evolutionary dynamic optimization (2011), `http://arxiv.org/abs/1106.0566`
2. Doerr, B., Goldberg, L.A.: Drift Analysis with Tail Bounds. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 174–183. Springer, Heidelberg (2010)
3. Doerr, B., Goldberg, L.A.: Adaptive drift analysis. CoRR, abs/1108.0295 (2011)
4. Doerr, B., Neumann, F., Sudholt, D., Witt, C.: On the runtime analysis of the 1-ANT ACO algorithm. In: Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 33–40. ACM (2007)

5. Droste, S.: Analysis of the (1+1) EA for a dynamically changing OneMax-variant. In: IEEE Congress on Evolutionary Computation (CEC 2002), pp. 55–60. IEEE Press (2002)
6. Droste, S.: Analysis of the (1+1) EA for a Dynamically Bitwise Changing OneMax. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 909–921. Springer, Heidelberg (2003)
7. Gutjahr, W.J., Sebastiani, G.: Runtime analysis of ant colony optimization with best-so-far reinforcement. Methodology and Computing in Applied Probability 10, 409–433 (2008)
8. Jansen, T., Schellbach, U.: Theoretical analysis of a mutation-based evolutionary algorithm for a tracking problem in the lattice. In: Genetic and Evolutionary Computation Conference (GECCO 2005), pp. 841–848 (2005)
9. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments—a survey. IEEE Transactions on Evolutionary Computation 9, 303–317 (2005)
10. Kötzing, T., Neumann, F., Sudholt, D., Wagner, M.: Simple max-min ant systems and the optimization of linear pseudo-boolean functions. In: Foundations of Genetic Algorithms (FOGA 2011), pp. 209–218 (2011)
11. Neumann, F., Sudholt, D., Witt, C.: Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. Swarm Intelligence 3, 35–68 (2009)
12. Neumann, F., Sudholt, D., Witt, C.: A few ants are enough: ACO with iteration-best update. In: Genetic and Evolutionary Computation Conference (GECCO 2010), pp. 63–70. ACM (2010)
13. Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. Algorithmica 54, 243–255 (2009)
14. Oliveto, P.S., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica 59, 369–386 (2011)
15. Rohlfshagen, P., Lehre, P.K., Yao, X.: Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In: Genetic and Evolutionary Computation Conference (GECCO 2009), pp. 1713–1720 (2009)
16. Stützle, T., Hoos, H.H.: MAX-MIN ant system. Journal of Future Generations Computer Systems 16, 889–914 (2000)
17. Zhou, D., Luo, D., Lu, R., Han, Z.: The use of tail inequalities on the probable computational time of randomized search heuristics. Theoretical Computer Science (to appear, 2012)