Theoretical Computer Science 411 (2010) 2084–2100

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Runtime analysis of a binary particle swarm optimizer

Dirk Sudholt [a,*], Carsten Witt [b]

[a] International Computer Science Institute, Berkeley, CA 94704, USA

[b] DTU Informatics, Technical University of Denmark, Kongens Lyngby, Denmark

ARTICLE INFO

Keywords:
Particle swarm optimization
Runtime analysis

ABSTRACT

We investigate the runtime of a binary Particle Swarm Optimizer (PSO) for optimizing pseudo-Boolean functions $f: \{0,1\}^n \to \mathbb{R}$. The binary PSO maintains a swarm of particles searching for good solutions. Each particle consists of a current position from $\{0,1\}^n$, its own best position and a velocity vector used in a probabilistic process to update its current position. The velocities for a particle are then updated in the direction of its own best position and the position of the best particle in the swarm.

We present a lower bound for the time needed to optimize any pseudo-Boolean function with a unique optimum. To prove upper bounds we transfer a fitness-level argument that is well-established for evolutionary algorithms (EAs) to PSO. This method is applied to estimate the expected runtime for the class of unimodal functions. A simple variant of the binary PSO is considered in more detail for the test function OneMax, showing that there the binary PSO is competitive to EAs. An additional experimental comparison reveals further insights.

1. Introduction

The runtime analysis of randomized search heuristics is a growing area with many interesting results found in the few last decades. The analysis of evolutionary algorithms started with the investigation of simple evolutionary algorithms applied to simple example functions (see, e.g., [6]). The theoretical results derived from such analyses then helped in developing methods for analyzing more complex evolutionary algorithms on more complex problems. The runtime analysis of evolutionary algorithms can be called a success story since nowadays analyses are possible for many problems from combinatorial optimization. This includes, for example, the single-source shortest path problem [24], maximum matchings [7], spanning tree problems [19,18], matroid optimization [23] as well as the NP-hard partition problem [27]. Further examples can be found in a recent survey by Oliveto, He, and Yao [22].

In recent years, the first runtime analyses on swarm intelligence algorithms have appeared, following an approach similar to that taken for the analysis of evolutionary algorithms. Such analyses are, in general, more difficult than for evolutionary algorithms as the probabilistic model underlying swarm algorithms may depend on a long history of past solutions. Droste [5] presented first runtime analyses for an estimation-of-distribution algorithm, the compact genetic algorithm, for linear functions. Regarding ant colony optimization (ACO), first runtime analyses have been presented independently by Gutjahr [8] and Neumann and Witt [21]. Neumann and Witt defined a simple ant algorithm called 1-ANT and analyzed its performance on the well-known test function OneMax. It turned out that the 1-ANT is very sensitive to the choice of the so-called evaporation factor that determines the amount of change in the probabilistic model. Similar results for other functions were presented by Doerr, Neumann, Sudholt and Witt [4]. On the other hand, variants of the Max–Min Ant System (MMAS)

* Corresponding author.
E-mail address: sudholt@icsi.berkeley.edu (D. Sudholt).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.03.002

proved to be quite effective for simple functions [9,17]. As in the history of evolutionary algorithms, it was found that the techniques developed in these early works could also be applied to problems from combinatorial optimization. Neumann and Witt [20] presented an analysis for ACO algorithms for the minimum spanning tree problem. Attiratanasunthron and Fakcharoenphol [1] as well as Horoba and Sudholt [11,29] analyzed ACO algorithms for shortest path problems. A conclusion drawn from these works is that ACO algorithms perform better than evolutionary algorithms in some settings.

Particle Swarm Optimizers (PSO) form another class of swarm algorithms mostly applied in continuous spaces. Originally developed by Kennedy and Eberhart [12], the bio-inspired optimization principle has become popular in recent years. A comprehensive treatment is given in the book by Kennedy, Eberhart, and Shi [14]. A typical PSO maintains a swarm of particles where each particle corresponds to a solution of the problem at hand. Each particle moves through the search space with a certain velocity. In every iteration the velocity of a particle is updated in the direction of its own best solution and the best particle in its neighborhood. This kind of behavior is motivated from social psychology theory as it combines cognitive and social effects to determine the behavior of each particle.

Kennedy and Eberhart [13] presented a first binary particle swarm optimizer, called *binary PSO*. As in classical PSO, velocities are used to determine the next position of a particle. However, as each bit may only obtain discrete values 0 and 1, velocities are used in a stochastic solution construction process. More precisely, the velocity value of a bit determines the probability of setting this bit to 1 in the next solution construction. This closely relates to some of the ACO algorithms cited above.

Our aim is to develop a theoretical understanding of the binary PSO, in particular from the perspective of computational complexity. We start from the original formulation by Kennedy and Eberhart [13], where all velocities are clamped to a fixed interval $[-v_{\max}, v_{\max}]$ to prevent divergence of the system. We prove in Section 2 that the effect on the performance is disastrous if the velocity bound $v_{\max}$ is fixed while the problem size grows. Instead, we present a formulation of the binary PSO with $v_{\max}$ adjusted towards growing problem dimensions. The new choice of $v_{\max}$ leads to provably efficient optimization times without alternative approaches for velocity control as described, e.g., in [2].

In Section 3 we present lower bounds on the runtime of the binary PSO. Section 4 shows how fitness-level arguments, a powerful tool for the analysis of evolutionary algorithms, can be used for the analysis of the binary PSO. As an example, we apply this technique to the class of unimodal functions. In Section 5, we consider a specific variant of the binary PSO in more detail. The 1-PSO works with a swarm consisting of only one particle. Despite its simplicity, the 1-PSO turns out to be surprisingly efficient. A thorough analysis on the function OneMax in Section 5 shows that the 1-PSO is competitive to evolutionary algorithms. This theoretical result is supplemented in Section 6 with an experimental comparison of the 1-PSO with a simple evolutionary algorithm on OneMax. We conclude in Section 7.

2. The binary PSO

We consider the binary PSO of Kennedy and Eberhart [13] for the optimization of a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. The binary PSO algorithm maintains μ triples $(x^{(i)}, x^{(i)*}, v^{(i)})$, $1 \le i \le \mu$, denoted as particles. Each particle i consists of its current position $x^{(i)} \in \{0,1\}^n$, its own best position $x^{(i)*} \in \{0,1\}^n$ and its velocity $v^{(i)} \in \mathbb{R}^n$. Note that the velocity is from a continuous domain. In PSO terminology, the three components of a particle are often called vectors. Using the language of optimization, we will refer to particle positions synonymously as solutions.

The movement for each particle is influenced by the best particle in its neighborhood. Hence, depending on the neighborhood structure, different particles may be guided by different good solutions. In this work, however, we only consider a global neighborhood consisting of the whole swarm. This model is known as the *gbest* model and it implies that all particles are influenced by a single global best position, denoted as x^{**}. For simplicity, we hereinafter use the term *global best* and the symbol x^{**} both for the global best position and the particle whose own best position determined the global best position.

The velocities are updated as follows. The velocity vector is changed towards the particle's own best and towards the global best x^{**}. Using the language of social psychology, the first component is often called the cognitive component and the latter is often called the social component. The impact of these two components is determined by so-called *learning factors* $c_1, c_2 \in \mathbb{R}_0^+$ representing parameters of the system. The factor c_1 is the learning factor for the cognitive component and c_2 is the one for the social component. A common choice for the learning factors is to set $c_1 = c_2 = 2$.

We give a precise definition for the binary PSO with a swarm size of μ and learning factors c_1, c_2. Using lower indices we indicate the n components of the three parts of the particle.

The algorithm starts with an initialization step (Step 1), where all velocities are set to all-zero vectors and all solutions, including own best and global best positions, are undefined, represented by the symbol $\perp$. The subsequent loop (Steps 2–5) chooses anew in each iteration random vectors $r_1 \in (U[0, c_1])^n$ and $r_2 \in (U[0, c_2])^n$ with each component drawn independently and uniformly from the given interval. These values are then used as weights for the cognitive and the social component, respectively. Note that for the default choices $c_1 = c_2 = 2$ the expected weight for each component is 1. Using the language of evolutionary algorithms, we refer to iterations synonymously as generations.

In Step 3, the velocity is probabilistically translated into a new position for the particle, i.e., a new solution. As proposed in the original formulation, we use the sigmoid function

$$s(v) := \frac{1}{1 + e^{-v}}$$

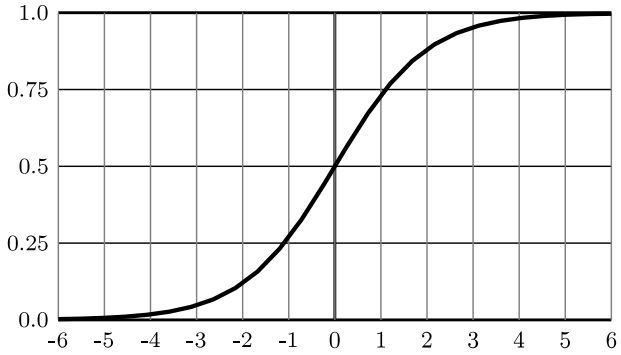

Fig. 1. The sigmoid function $s(v) = \frac{1}{1+e^{-v}}$.

sketched in Fig. 1. Positive velocity components bias the corresponding bit towards 1-values while negative velocities favor 0-values. At velocity 0^n, each bit is completely random; hence the first created solution is uniformly distributed over $\{0, 1\}^n$.

Afterwards, the particle's own best and the global best are exchanged if the newly constructed solution is better. Note that the selection is strict, i.e., a best solution is only exchanged if the new solution has strictly larger fitness.

In Step 4 the binary PSO performs some vector arithmetic to update the velocity vectors probabilistically in the direction of the particle's own best and the global best. The operator $\otimes$ denotes componentwise vector multiplication. To ensure convergence of the heuristic, every velocity vector is bounded componentwise by minimum and maximum values, i.e., to an interval $[-v_{\max}, v_{\max}]$. This reflects the common choice of a maximum velocity as studied by Shi and Eberhart [25]. For practical purposes, often velocities in the interval $[-4, 4]$ are proposed. Since we will however conduct an asymptotic analysis, we allow the maximum velocity to grow with the problem dimension n and confine the components to logarithmic values by letting $v_{\max} := \ln(n-1)$. We will justify this choice later.

Algorithm 1 (*Binary PSO*).
1. Initialize velocities with 0^n and all solutions with $\perp$.
2. Choose $r_1 \in (U[0, c_1])^n$ and $r_2 \in (U[0, c_2])^n$.
3. For $j := 1$ to μ do:
   For $i := 1$ to n do:
   Set $x_i^{(j)} := 1$ with probability $s(v_i^{(j)})$; otherwise set $x_i^{(j)} := 0$.
   If $f(x^{(j)}) > f(x^{(j)*})$ or $x^{(j)*} = \perp$ then $x^{(j)*} := x^{(j)}$.
   If $f(x^{(j)*}) > f(x^{**})$ or $x^{**} = \perp$ then $x^{**} := x^{(j)*}$.
4. For $j := 1$ to μ do:
   Set $v^{(j)} := v^{(j)} + r_1 \otimes (x^{(j)*} - x^{(j)}) + r_2 \otimes (x^{**} - x^{(j)})$.
   For $i := 1$ to n do:
   Set $v_i^{(j)} := \max\{v_i^{(j)}, -v_{\max}\}$.
   Set $v_i^{(j)} := \min\{v_i^{(j)}, v_{\max}\}$.
5. Go to 2.

Note that we do not define a stopping criterion. Instead, we consider the algorithm as an infinite process as we are only interested in the random time until the algorithm finds a global optimum. As performance measures we consider the number of generations as well as the number of constructed solutions.

We will deal with different parametrizations of the binary PSO, differing in the swarm size μ and the learning factors c_1 and c_2. In particular, we deal with a remarkably simple yet effective algorithm, the so-called 1-PSO using just one particle. With just one particle, its own best and the global best coincide. Therefore, it makes sense to turn off the social component by setting $c_2 = 0$. The cognitive learning factor is set to the default value $c_1 = 2$. Note that the same algorithm is described by the choice $c_1 = 0$ and $c_2 = 2$. Dropping the upper index in the notation, the 1-PSO can be stated as follows.

Algorithm 2 (*1-PSO*).
1. Initialize $v = 0^n$ and $x^{**} = \perp$.
2. Choose $r \in (U[0, 2])^n$.
3. For $i := 1$ to n do:
   Set $x_i := 1$ with probability $s(v_i)$; otherwise set $x_i := 0$.
   If $f(x) > f(x^{**})$ or $x^{**} = \perp$ then $x^{**} := x$.
4. Set $v := v + r \otimes (x^{**} - x)$.
   For $i := 1$ to n do:
   Set $v_i := \max\{v_i, -v_{\max}\}$.
   Set $v_i := \min\{v_i, v_{\max}\}$.
5. Go to 2.

Note that during the velocity update in expectation, just $x^{**} - x$ is added to v.

There are several good reasons for investigating the 1-PSO. One is that in the binary PSO without social component, i.e., with $c_2 = 0$, all particles behave like independent instances of the 1-PSO. Moreover, by analyzing the 1-PSO we gain insight into the probabilistic model underlying the binary PSO. This then helps in analyzing more complex PSO variants. Finally, the investigation of the 1-PSO is interesting in its own right as the 1-PSO turns out to be surprisingly effective.

2.1. A lower bound for the constant velocity range

The value $v_{\max}$ is often set to a constant value. This makes sense when dealing with problems of bounded size. However, one should be aware of the fact that for growing problem sizes a fixed value for $v_{\max}$ leads to an extreme decline in performance.

The following lower bound shows that if all velocities are restricted to constant values, then the binary PSO is too close to random search and the algorithm fails badly, even given exponential time and a large number of global optima.

Theorem 1. *Consider the binary PSO with arbitrary values for μ, c_1, and c_2, where $v_{\max}$ is redefined to a constant value. Then there is a constant $\alpha = \alpha(v_{\max})$ such that the following holds. If f contains at most $2^{\alpha n}$ global optima, the probability that the binary PSO finds a global optimum on f within $2^{\alpha n}$ constructed solutions is $2^{-\alpha n}$.*

Proof. Choose α such that $s(v_{\max}) = 2^{-3\alpha}$ and note that α is a positive constant if $v_{\max}$ is constant. We estimate the probability of constructing any specific solution x. Since the binary PSO treats 0-bits and 1-bits symmetrically, we can w.l.o.g. assume that x is the all-ones string 1^n. Then, even if all velocities are at $v_{\max}$, the probability of constructing x is still bounded by $(s(v_{\max}))^n = 2^{-3\alpha n}$. By the union bound, the probability of constructing any global optimum out of at most $2^{\alpha n}$ ones is bounded by $2^{\alpha n} \cdot 2^{-3\alpha n} = 2^{-2\alpha n}$. By the same argument, the probability that this happens at least once in $2^{\alpha n}$ solution constructions is at most $2^{\alpha n} \cdot 2^{-2\alpha n} = 2^{-\alpha n}$. □

For the common choice $v_{\max} := 4$, the constant α is approximately 0.00873. As $2^{0.00873 \cdot n}$ is small for small n, the bad runtime behavior can only be observed if the problem size is large enough. This certainly isn't the case for $n = 100$ where $2^{\alpha n} < 2$. However, for a problem size of $n = 10\,000$, the claimed bound has grown to $2^{\alpha n} > 10^{26}$ and we would not expect to live long enough to see the binary PSO find an optimum.

Theorem 1 rules out a constant default value for $v_{\max}$ that works well for all problem sizes. We therefore propose to let $v_{\max}$ scale with the problem size. More precisely, we set $v_{\max} = \ln(n-1)$. As $s(-v_{\max}) = 1/n$ and $s(v_{\max}) = 1 - 1/n$, the probability of setting a bit to 1 is always in the interval $[1/n,\ 1 - 1/n]$. This is inspired by standard mutation operators in evolutionary computation, where incorrectly set bits have a probability of $1/n$ of being corrected. We will see in the following that this choice leads to a surprisingly good runtime behavior.

3. A lower bound for the binary PSO

An important step towards runtime bounds for the binary PSO is to understand the dynamics of the probabilistic model underlying PSO, that is, the behavior of the velocity vector. Consider a single bit that is set to 1 both in the corresponding own best and in the global best. Then, as long as these solutions are not exchanged, its velocity value v is guided towards the upper bound $v_{\max}$. An important observation is that the velocity is only increased if the bit is set to 0 in the next solution constructed. The probability that this happens is given by

$$1 - s(v) = 1 - \frac{1}{1 + e^{-v}} = \frac{1}{1 + e^{v}},$$

and we see that this probability decreases rapidly with growing v. Hence, the closer the velocity is to the bound $v_{\max}$, the harder it is to get closer. A symmetric argument holds for velocities that are guided towards $-v_{\max}$.

As long as the v-values are not too close to the velocity bounds $-v_{\max}$ and $v_{\max}$, the search of the binary PSO is too random for it to find single optima with high probability. We can make this idea precise by means of the following, general lower bound, which holds for all practical choices of the learning factors c_1 and c_2 and a polynomial swarm size μ. The maximum change of a velocity per generation will be abbreviated as $\bar{c} := c_1 + c_2$ hereinafter.

Theorem 2. *Let f be a function with a unique global optimum, and let $\mu = \text{poly}(n)$ and $\bar{c} = O(1)$. Then the expected number of generations of the binary PSO on f is $\Omega(n/\log n)$.*

Central tools in our proofs, not only for Theorem 2, will be Chernoff and Hoeffding bounds. Already at this point, we summarize all such bounds that are needed in the course of the paper. Their formulations have been adapted to make the forthcoming applications as easy to present as possible.

Lemma 1 (*Chernoff [15] and Hoeffding [10] Bounds*). *Let $X_1, \ldots, X_n$ be independent random variables and define $X := X_1 + \cdots + X_n$. If $X_i \in \{0, 1\}$ for $1 \le i \le n$ then for $0 < \delta \le 1$*

$$P(X \le (1-\delta)E(X)) \le e^{-E(X)\delta^2/2} \quad \text{and} \quad P(X \ge (1+\delta)E(X)) \le e^{-E(X)\delta^2/3}.$$

If $0 \le X_i \le b$ for $1 \le i \le n$ then for $\delta > 0$

$$P(X \le (1-\delta)E(X)) \le e^{-2E(X)^2\delta^2/(nb^2)}.$$

Proof of Theorem 2. W.l.o.g. the global optimum is 1^n. Let $t := \alpha n/\ln n$ for a small constant $\alpha > 0$, which is chosen later. We show that the probability of not creating 1^n within t generations is $1 - o(1)$, which implies the claim of the theorem.

We consider an arbitrary bit in an arbitrary particle. The event of creating a 1 at this bit is called *success*. Let a bit be called *weak* if its success probability has been at most $p := 1 - \frac{e \ln(\mu n)}{n}$ up to and including the current generation. Let a set of bits be called weak if it contains only weak bits. We will show that with probability $1 - 2^{-\Omega(n)}$ after t generations of the 1-PSO, each particle still contains a weak subset of bits of size at least n/e. The probability of setting all bits of such a weak subset to 1 simultaneously is bounded from above by $p^{n/e} \leq 1/(\mu n)$ for each particle. Note that this event is necessary for creating 1^n in a particle. Thus, the probability of finding the optimum within t generations creating μ new solutions each is still less than $t\mu/(\mu n) = O(1/\log n) = o(1)$, which will prove the theorem.

We still have to show that with probability $\Omega(1)$, after t generations, there is a weak subset of size at least n/e in each particle. One step can increase the velocity by at most $\bar{c}$. Note that $p = 1 - O((\ln n)/n)$ since $\mu = \text{poly}(n)$. To reach success probability at least p, the current velocity must be between $s^{-1}(p) - \bar{c} = \ln(p/(1-p)) - \bar{c}$ and $s^{-1}(p)$ at least once. Pessimistically assuming the first value as the current velocity, the probability of not increasing it in a single step is at least

$$\frac{1}{1 + e^{-\ln(p/(1-p)) + \bar{c}}} = 1 - \frac{e^{\bar{c}}(1-p)}{p + e^{\bar{c}}(1-p)} \geq 1 - 2e^{\bar{c}}(1-p)$$

if n is large enough for $p \geq 1/2$ to hold. The last expression equals $1 - 2e^{\bar{c}+1} \ln(\mu n)/n$ by definition of p. Hence, along with $\bar{c} = O(1)$ and again $\mu = \text{poly}(n)$, the probability of not increasing the velocity within t steps is at least

$$\left(1 - \frac{2e^{\bar{c}+1} \ln(\mu n)}{n}\right)^t = \left(1 - \frac{O(\ln n)}{n}\right)^{\alpha n/\ln n} \geq 2e^{-1}$$

if α is chosen small enough. This means that each bit in each particle independently has a probability of at least $2e^{-1}$ of being weak at generation t. Using Chernoff bounds (Lemma 1), the probability of not having a weak set of size at least n/e in a specific particle is at most $e^{-\Omega(n)}$. As $\mu = \text{poly}(n)$, the probability that there exists a particle without a weak subset at generation t is still $\mu e^{-\Omega(n)} = e^{-\Omega(n)}$. □

4. Upper bounds for the binary PSO

In this section we derive an upper bound for the binary PSO. The lower bound from Theorem 2 relied on the fact that a velocity that is guided towards $v_{\max}$ does not reach this value in short time and so the binary PSO cannot find a single target efficiently. On the other hand, if we consider a longer period of time, velocities may reach the bounds $-v_{\max}$ or $v_{\max}$ that they are guided to. In this case we say that the velocity has been "frozen" as the only chance to alter the velocity again is to have an improvement of some own best or the global best. The random time until a bit is frozen is called the *freezing time*.

Observe that freezing is only possible for bits where its own best and the global best coincide or if one of the learning factors c_1, c_2 is 0. Freezing does not happen if, e.g., $c_1 = c_2 > 0$ and the corresponding own best and the global best differ for a bit i. W.l.o.g., the own best has bit value 1 and $x_i^{**} = 0$; then the new velocity v_i' for this bit equals

$$v_i' = v_i + r_{1,i}(1 - x_i) + r_{2,i}(0 - x_i) = v_i + r_{1,i} - (r_{1,i} + r_{2,i}) \cdot x_i,$$

where $r_{1,i}$ ($r_{2,i}$) denotes the i-th entry of the random vector r_1 (r_2). This means that if $v_i \gg 0$, there is a good chance that $x_i = 1$ and then $v_i' = v_i - r_{2,i}$. In contrast, if $v_i \ll 0$, it is likely that $x_i = 0$ and $v_i' = v_i + r_{1,i}$. In both cases the velocity v_i is driven towards velocity value 0. So, it makes sense to define the freezing time only for bits whose velocity can freeze at $-v_{\max}$ or $v_{\max}$. We summarize our sufficient conditions for freezing through the following definition.

Definition 1. A bit $x_i^{(j)}$ of a particle $x^{(j)}$ in the binary PSO is called *well-guided* with respect to a given time interval if at least one of the following conditions is met.

- The corresponding bit in the global best, x_i^{**}, remains fixed in the time interval and $c_1 = 0$.
- The corresponding bit in its own best, $x_i^{(j)*}$, remains fixed in the time interval and $c_2 = 0$.
- Both x_i^{**} and $x_i^{(j)*}$ remain fixed in the time interval and $x_i^{**} = x_i^{(j)*}$.

In order to arrive at an estimation of the expected freezing time, we investigate the random process of velocities moving towards a bound. Assume that we are interested in the random hitting time until a velocity has increased to a value of at least a, for some fixed real value a. Intuitively, we expect monotonicity to hold: larger initial velocities are always better than smaller ones, in terms of the expected hitting time. However, such monotonicity does not hold in general. The reason is that the probability of increasing a velocity decreases with the velocity itself and velocities very close to a can actually be inferior to values that are a little further away.

We give a simple example for the 1-PSO. Consider two bits with current velocity values v_1 and v_2, guided towards $v_{\max}$ with $v_{\max} > 3$. Assume that $v_1 = 2$ and $v_2 = 2.5$ and let V_1', V_2' denote the random velocities after the next step. We compare the probabilities that the random new velocities reach a value of at least 3 in the next step. As the current velocity

for v_2 is larger, we expect chances for this bit to be higher. In order to increase the velocity, we have to create value 0 at the bit. The amount of increase is then uniform over $[0, 2]$. Hence we have

$$P(V_1' \geq 3) = (1 - s(v_1)) \cdot \frac{1}{2} = s(-2) \cdot \frac{1}{2} \approx 0.0596.$$

On the other hand,

$$P(V_2' \geq 3) = (1 - s(v_2)) \cdot \frac{3}{4} = s(-2.5) \cdot \frac{3}{4} \approx 0.0569$$

and $P(V_2' \geq 3) < P(V_1' \geq 3)$, in contrast to our expectations. In this example, the larger velocity only *seems* to have a clear advantage over the smaller velocity. In fact, monotonicity does not hold, as the probability of increasing a current velocity of v equals $1 - s(v)$, i.e., decreases rapidly with v.

Nevertheless, our observation does not imply in general that small velocities are superior to large velocities. This only holds at a microscopic level. If, on the other hand, $v_2 \geq v_1 + \bar{c}$ is presupposed in our example (recall that $\bar{c} = c_1 + c_2$ denotes the maximum increase of velocity in a generation), then $P(V_2' \geq a) \geq P(V_1' \geq a)$ holds for every value of a. The reason is that the increase of a velocity is bounded from above by $\bar{c}$, so V_1' cannot "overtake" V_2' in this case. Unfortunately, the corresponding inequality $V_2' \geq V_1' + \bar{c}$ does not necessarily carry over to the next step. Therefore, we take a slightly different viewpoint. If the current velocity V is random and follows a probability distribution on the interval $[-v_{\max}, v_{\max}]$, we study the process $V - \bar{c}$ obtained by decreasing all velocities by the maximum change of velocity, namely $\bar{c}$. Values $v \in [-v_{\max}, v_{\max}]$ with positive probability mass are mapped to $v - \bar{c} \in [-v_{\max} - \bar{c}, v_{\max} - \bar{c}]$. If the modified process decides to move from $v - \bar{c}$ to some higher velocity, it does so with at least the same probability as the original process since its velocity is lower. Although the modified process might take a littler longer to reach the desired value a than the original, it has an advantage that is maintained for the following time steps.

In the following, we introduce a relaxed concept of domination reflecting the preceding considerations. One instance of the velocity-increasing process is allowed to be by up to $\bar{c}$ behind another instance and has an advantage from this perspective. For notational convenience, we add $\bar{c}$ to the velocity of the dominated process rather than subtracting $\bar{c}$ from the dominating one.

Definition 2. Suppose there is a bit that is permanently well-guided, w.l.o.g., to $+v_{\max}$. Imagine two separate and independent random velocity processes V_1, V_2 for this bit. Denote by $V_1^{(t)}$ and $V_2^{(t)}$ their random velocity values after t iterations have elapsed. We say that $V_1^{(t)}$ $\bar{c}$-dominates $V_2^{(t)}$, written as $V_1^{(t)} \succeq_{\bar{c}} V_2^{(t)}$, if for all $a \in \mathbb{R}$ it holds that

$$P(V_1^{(t)} \geq a) \geq P(V_2^{(t)} \geq a + \bar{c}).$$

If one velocity $\bar{c}$-dominates another one at some time, this property will be preserved in the following.

Lemma 2. *Assume that being well-guided holds in the time interval $[t, t+1]$. Then $V_1^{(t)} \succeq_{\bar{c}} V_2^{(t)} \Rightarrow V_1^{(t+1)} \succeq_{\bar{c}} V_2^{(t+1)}$.*

A proof is given in the Appendix. We are aiming at an upper bound on freezing times, i.e., the first point of time where a velocity value has reached the bound corresponding to the global best, which, w.l.o.g., is $+v_{\max}$. The concept of domination allows us to take the following worst-case perspective for the velocity value after a number of steps t. We shift the velocity scale up by $\bar{c}$ and arrive at a simplified Markov process $\tilde{v}_t$, $t \geq 0$, called *$\tilde{v}$-process*, defined on $[-v_{\max} + \bar{c}, v_{\max} + \bar{c}]$, as follows. Initially, $\tilde{v}_0 := -v_{\max} + \bar{c}$. For $t \geq 0$, the random state $\tilde{v}_{t+1}$ is obtained as follows:

$$\tilde{v}_{t+1} := \begin{cases} \min\{\tilde{v}_t + r_1 + r_2, v_{\max} + \bar{c}\} & \text{with probability } \frac{1}{1+e^{\tilde{v}_t}} \\ \tilde{v}_t & \text{otherwise,} \end{cases}$$

where $r_1 \in U[0, c_1]$ and $r_2 \in U[0, c_2]$ are drawn independently anew in each step. The reason for shifting the velocity scale is that we are looking for velocities that $\bar{c}$-dominate the $\tilde{v}$-process. Hence, a velocity of at least v_t in the current process at time t will be related to a velocity of at least $v_t + \bar{c}$ in the $\tilde{v}$-process. Note that an increase of the first velocity value happens with probability $1/(1 + e^{v_t})$, whereas this probability is only $1/(1 + e^{v_t + \bar{c}})$ with the second velocity. However, the ratio of the two probabilities can be bounded according to $(1 + e^{v_t})/(1 + e^{v_t + \bar{c}}) \geq e^{-\bar{c}}$, i.e., independently of v_t. From an opposite perspective, a (random) velocity $\tilde{v}_t$ of the $\tilde{v}_t$-process corresponds one to one to a (random) probability $\tilde{P}_t := 1/(1 + e^{-\tilde{v}_t})$ of setting the considered bit to 1 (called success) at time t. At velocity $\tilde{v}_t - \bar{c}$, the actual process has a success probability of only $P_t := 1/(1 + e^{-\tilde{v}_t - \bar{c}})$.

We justify why the $\tilde{v}$-process is really a worst case if we are interested in first hitting times for the upper velocity bound $v_{\max}$.

Lemma 3. *Consider a bit of a particle that is well-guided for t generations, w.l.o.g., to $+v_{\max}$, and let v_t and $P_t = s(v_t)$ denote its velocity and (random) success probability, respectively, after t generations have elapsed. Then v_t $\bar{c}$-dominates the $\tilde{v}$-process. Moreover, for every $p \geq 0$, $P(P_t \geq 1 - e^{\bar{c}} p) \geq P(\tilde{P}_t \geq 1 - p)$, where $\tilde{P}_t$ denotes the success probability of the $\tilde{v}$-process.*

Proof. We prove the first claim by induction. Obviously, any initialization $v_0 \in [-v_{\max}, v_{\max}]$ of the real velocity implies that v_0 $\bar{c}$-dominates $\tilde{v}_0 := -v_{\max} + \bar{c}$, the initial value of the $\tilde{v}$-process. Due to Lemma 2, this property is preserved for all following time steps where the bit is still well-guided.

The second property follows from the fact $(1 + e^{v_t})/(1 + e^{v_t+\bar{c}}) \geq e^{-\bar{c}}$ by means of $1 - 1/(1 + e^x) = 1/(1 + e^{-x})$. □

Having established the $\tilde{v}$-process as a worst-case perspective for the real velocity process, we can now move towards analyzing first hitting times for the real process. The following lemma will be used to bound the expected freezing time for specific bits. We state a more general result that also gives insight into the distribution of random velocities on their way to the velocity bounds. In fact, the lemma captures the random success probabilities induced by these velocities very precisely, depending on the number of generations t that the bit has been given to "evolve" its distribution towards the velocity bound. This will be important later on in Section 5, when performing a more detailed analysis on the specific function OneMax.

Formally, Lemma 4 makes a statement about the random probability P_t of setting a bit to 1, after it has been guided towards the velocity bound $v_{\max}$ for t time steps. The length of the time span t considered can be chosen almost arbitrarily, subject to a mild lower bound. For fixed t, the lemma presents lower bounds for the probability that P_t exceeds a family of probability thresholds. The thresholds are specified by a second value i; they decrease from high thresholds to lower thresholds as i increases. The probability that P_t exceeds the i-th threshold is at least $1 - e^{-i}$. This means that the probability of not overcoming the i-th threshold decreases exponentially with i. Intuitively, this accounts for the fact that low velocities are increased far more easily than high velocities. Failing to overcome low thresholds is, therefore, far more unlikely than failing on higher ones.

Lemma 4. *Consider a well-guided bit of a particle in the binary PSO. After a number t of elapsed generations, let $P_t = s(v_t)$ denote the random probability of setting the bit according to the bit value that it is guided to, w.l.o.g. value 1.*

Choosing $c^* := \min\{1, \bar{c}/2\}$, $t \geq (16 \ln n)/c^*$ and n large enough, it holds for every real-valued $\alpha \in [1, c^*t/192]$

$$P\left(P_t \geq \min\left\{1 - \frac{96\alpha e^{\bar{c}}}{c^* \cdot t},\ 1 - \frac{1}{n}\right\}\right) \geq 1 - e^{-\alpha}.$$

More generally, this statement holds if P_t denotes the success probability of a bit whose velocity $\bar{c}$-dominates the $\tilde{v}$-process at time t.

Note that if we were only interested in an upper bound on the expected freezing time, it would suffice to state the lemma for just one threshold, namely $P_t = 1 - 1/n$, which corresponds to a velocity at $v_{\max}$.

Proof. Instead of the actual velocity v_t after t steps, we consider the $\tilde{v}$-process. According to Lemma 3, this provides a pessimistic estimation of the actual (random) velocity value after t steps in terms of $\bar{c}$-domination and justifies the more general claim in the last sentence of the lemma. Moreover, the required success probability $1 - 96\alpha e^{\bar{c}}/(c^*t)$ at the bit in the worst case corresponds to a success probability $1 - 96\alpha/(c^*t)$ of the $\tilde{v}$-process. We show the following claim: with probability at least $1 - e^{-\alpha}$, it holds for the success probability $\tilde{P}_t$ of the $\tilde{v}$-process after t steps that $\tilde{P}_t \geq 1 - 96\alpha/(c^*t)$, or $\tilde{v}_t$ has reached $v_{\max} + \bar{c}$ anyway; the latter case will pessimistically be ignored hereinafter.

Recall that the probability of increasing the $\tilde{v}$-value decreases monotonically with the $\tilde{v}$-value. It is therefore bounded from below by $96\alpha/(c^*t)$ before the velocity has reached the desired value. Note that for smaller velocities this probability is much higher as it increases rapidly with increasing distance from the bound. The progress of the velocity from $\tilde{v}_0 = -v_{\max}+\bar{c}$ to the value corresponding to success probability $1 - 96\alpha/(c^*t)$ is divided into two big epochs. Throughout the first epoch, the $\tilde{v}$-value is at most 0, which, by definition of the $\tilde{v}$-process, implies that each generation increases the velocity with a probability of at least $1/2$, i.e., independent of t and α. In the second epoch, we have to be more careful. An increase of the velocity by a constant additive amount reduces the probability of a further increase by a constant factor. Hence, to obtain a good bound on the probability of an increase throughout the second epoch, we divide the progress into phases with exponentially decreasing probabilities.

We start with the first epoch. Our aim is to show that it ends after at most $t/2$ steps with high probability. For this to happen, the following two events together are sufficient:

1. In the first $t/2$ steps, there are at least $t/8$ increases, or a positive $\tilde{v}$-value is reached.
2. The total increase in $t/8$ increases is at least $c^*t/16$, or a positive $\tilde{v}$-value is reached.

Since $t \geq (16 \ln n)/c^*$ is assumed in this lemma, the proposed total increase is at least $v_{\max}$, i.e., sufficient to leave the velocity range $[-v_{\max} + \bar{c}, 0]$.

To estimate the joint probability of the two events, we use the Chernoff–Hoeffding bounds from Lemma 1. As we are in the first epoch, an increase happens with probability at least $1/2$. Using $E(X) \geq t/4$, $\delta := 1/2$ and the first bound of the lemma, the failure probability for the first event is at most $e^{-t/32}$, which is at most $e^{-6\alpha}$ since $\alpha \leq c^*t/192 \leq t/192$. For the second event, we apply the last bound of Lemma 1. Here we observe that the random amount of increase is uniform over $[0, \bar{c}]$ with an expectation of $\bar{c}/2$. On pessimistically minimizing $\bar{c}/2$ with 1 and making the distribution uniform over $[0, 2c^*]$, the new average amount of increase c^* yields a pessimistic estimation. Hence, the second bound of the lemma can be applied using $\delta = 1/2$, $E(X) \geq c^*t/8$ and $b \leq 2c^*$. We obtain a failure probability for the second event of at most $e^{-t/64} \leq e^{-3\alpha}$, altogether a failure probability of at most $e^{-3\alpha} + e^{-6\alpha}$ for the first epoch.

Assuming a positive velocity, i.e., a success probability greater than $1/2$ and a probability less than $1/2$ for an increase of the velocity, we have entered the second epoch. The number of phases that it is subdivided into equals $\lceil \log(c^*t/(192\alpha)) \rceil$. Note that some phases may be empty and phases are traversed in decreasing order. Phase k, $1 \leq k \leq \lceil \log(c^*t/(192\alpha)) \rceil$, is defined such that throughout the phase, the current success probability (i.e., the probability of *not* increasing the velocity) is within the interval $\left[1 - \frac{96\alpha 2^k}{c^*t},\ 1 - \frac{96\alpha 2^{k-1}}{c^*t}\right] \cap [1/2, 1]$; hence, the complementary probabilities exhibit an exponential decay. We use the inverse sigmoid function $s^{-1}(z) = \ln(z/(1-z))$ to map a success probability z of the $\tilde{v}$-process back to a velocity. This allows us to bound the length of the above-defined intervals on the velocity scale. If $1 - \frac{96\alpha 2^k}{c^*t} \geq 1/2$, the length is at most

$$\ln\left(\frac{1 - \frac{96\alpha 2^{k-1}}{c^*t}}{\frac{96\alpha 2^{k-1}}{c^*t}}\right) - \ln\left(\frac{1 - \frac{96\alpha 2^k}{c^*t}}{\frac{96\alpha 2^k}{c^*t}}\right) = \ln\left(\frac{2^k}{2^{k-1}} \cdot \frac{1 - \frac{96\alpha 2^{k-1}}{c^*t}}{1 - \frac{96\alpha 2^k}{c^*t}}\right) \leq \ln\left(\frac{2 \cdot 2^k}{2^{k-1}}\right) = \ln 4.$$

In the other case, which is equivalent to $\frac{96\alpha 2^k}{c^*t} > 1/2$, the interval ranges from 0 to $s^{-1}\left(1 - \frac{96\alpha \cdot 2^{k-1}}{c^*t}\right)$. We have $\frac{96\alpha 2^{k-1}}{c^*t} = \frac{1}{2} \cdot \frac{96\alpha 2^k}{c^*t} \geq \frac{1}{4}$. Hence, $s^{-1}\left(1 - \frac{96\alpha \cdot 2^{k-1}}{c^*t}\right) \leq \ln((1-1/4)/(1/4)) \leq \ln 4$, implying that the length of an interval on the velocity scale is bounded by $\ln 4$ in both cases. We will next show that, with high probability, we spend at most $\frac{tk}{2^{k+1}}$ steps in a phase. If this holds for all phases, the total time spent in all phases of the second epoch is less than $t \sum_{k=1}^{\infty} \frac{k}{2^{k+1}} = \frac{t}{2}$. Combining with the first epoch, the number of steps does not exceed t.

In a manner similar to (but more detailed than) that for the first epoch, the following set of conditions is sufficient for raising the velocity during the second epoch to the desired value. Note that Condition 2. suffices for leaving any interval of success probabilities considered.

For $k = \left\lceil \log(\frac{c^*t}{192\alpha}) \right\rceil$ down to 1 it holds that:

1. If the current success probability is in the interval
$$\left[1 - \frac{96\alpha 2^k}{c^*t},\ 1 - \frac{96\alpha 2^{k-1}}{c^*t}\right] \cap \left[\frac{1}{2},\ 1\right],$$
then $\frac{tk}{2^{k+1}}$ steps contain at least $12\alpha k/c^*$ increases.
2. The total increase in $12\alpha k/c^*$ increases is at least $\ln 4$.

It remains to estimate the probabilities of these events. This is done like for the first epoch by applying Lemma 1. For the first event we estimate $E(X) \geq 24\alpha k/c^*$ and use $\delta = 1/2$. For the second event, we use $E(X) \geq 12\alpha k$ and $\delta = 3/4$ since, clearly, $12\alpha k/4 > \ln 4$. Hence the failure probabilities for the two events can be bounded by $e^{-3\alpha k}$ and $e^{-27\alpha k/8}$, respectively. Combining with the first epoch, the sum of all failure probabilities is at most

$$e^{-3\alpha} + e^{-6\alpha} + \sum_{k=1}^{\infty}(e^{-3\alpha k} + e^{-27\alpha k/8}) = e^{-3\alpha} + e^{-6\alpha} + \frac{1}{e^{3\alpha} - 1} + \frac{1}{e^{27\alpha/8} - 1} \leq e^{-\alpha},$$

which proves the lemma. □

Finally, we can compute the expected time until bits freeze towards velocity bounds.

Lemma 5. *Consider a selection of $k \geq 1$ well-guided bits in the swarm of the binary PSO with $\bar{c} = \Theta(1)$. Assuming that being well-guided is preserved, the expected time until all k bits freeze to the velocity bounds that they are guided to is bounded by $O(n \ln(2k))$.*

Note that the factor 2 in $\ln(2k)$ cannot be dropped as otherwise for $k = 1$ we would have $\ln k = 0$. However, if $k > 1$, then $\ln(2k) = O(\ln k)$ and the bound simplifies to $O(n \ln k)$.

Proof. Let $c^* := \min\{1, \bar{c}/2\}$. Fix one of the selected bits and invoke Lemma 4 with $\alpha := \ln(2k)$ and $t := 96\alpha e^{\bar{c}} n/c^*$. Let P_t denote the probability that this bit is set to the bit value that it is guided to; then

$$P\left(P_t \geq \min\left\{1 - \frac{96\alpha e^{\bar{c}}}{c^*t},\ 1 - \frac{1}{n}\right\}\right) = P\left(P_t = 1 - \frac{1}{n}\right) \geq 1 - e^{-\ln(2k)} = 1 - \frac{1}{2k}.$$

The probability that this bit does not freeze within t steps is at most $1/(2k)$. Taking the union bound for k bits, the probability that any of these bits does not freeze within t steps is at most $1/2$. In this case, we repeat the argumentation with a new period of t steps. The expected number of periods until all bits are frozen is at most 2; hence we obtain the upper bound $2t = O(n \ln(2k))$. □

The result on the expected freezing time allows the derivation of upper bounds for the binary PSO like for evolutionary algorithms. For the sake of simplicity, we first describe this idea for the 1-PSO. Due to the strict selection in the 1-PSO, the global best is only exchanged if a better solution is discovered. This means that after some time either the global best has improved or all velocities in the global best are frozen. In the latter case, since $v_{\max} = \ln(n-1)$, the probability of creating a 1 for every bit x_i^{**} is now either $s(-v_{\max}) = 1/n$ or $s(v_{\max}) = 1 - 1/n$. The distribution of constructed solutions now equals the distribution of offspring for the $(1+1)$ evolutionary algorithm, for short $(1+1)$ EA, with the global best x^{**} as the current search point. For the sake of completeness, we give a definition of the $(1+1)$ EA.

Algorithm 3 *((1 + 1) EA).*

1. Choose an initial solution x^{**} uniformly at random.
2. For $i := 1$ to n do:
   Set $x_i := 1$ with probability $1 - 1/n$ if $x_i^{**} = 1$ and with probability $1/n$ if $x_i^{**} = 0$. Otherwise set $x_i := 0$.
3. If $f(x) \geq f(x^{**})$ then $x^{**} := x$.
4. Go to 2.

We also refer to the $(1+1)$ EA* as the $(1+1)$ EA with the condition in Step 3 replaced by $f(x) > f(x^{**})$. If for the 1-PSO all velocity values take their upper or lower bounds, the 1-PSO temporarily behaves like the $(1+1)$ EA* until a solution with larger fitness is encountered. This similarity between PSO and EAs can be used to transfer a well-known method for the runtime analysis from EAs to PSO, the fitness-level method. The underlying ideas were also used to transfer the method from EAs to ACO; see [9]. We present the fitness-level method, also called the method of f-based partitions (see [26]), in a restricted formulation.

Let $f_1 < f_2 < \cdots < f_m$ be an enumeration of all fitness values of f and let A_i, $1 \leq i \leq m$, contain all solutions with fitness f_i. We also say that A_i is the i-th fitness level. Note that the last fitness level A_m contains only optimal solutions. Now, let s_i, $1 \leq i \leq m-1$, be a lower bound on the probability for the $(1+1)$ EA (or, in this case equivalently, the $(1+1)$ EA*) creating an offspring in $A_{i+1} \cup \cdots \cup A_m$, provided the current population belongs to A_i. The expected waiting time until such an offspring is created is at most $1/s_i$ and then the i-th fitness level is left for good. As every fitness level has to be left at most once, the expected number of generations for the $(1+1)$ EA and the $(1+1)$ EA* to optimize f is bounded above by

$$\sum_{i=1}^{m-1} \frac{1}{s_i}. \tag{1}$$

A similar bound holds for the 1-PSO and, more generally, for the binary PSO.

Theorem 3. *Let A_i form the i-th fitness level of f and let s_i be the minimum probability for the $(1+1)$ EA leaving A_i going towards $A_{i+1} \cup \ldots \cup A_m$. If $\bar{c} = \Theta(1)$, the expected number of generations for the binary PSO to optimize f is bounded from above by*

$$O(mn \log n) + \sum_{i=1}^{m-1} \frac{1}{s_i}.$$

The right-hand sum is the upper bound obtained for the $(1+1)$ EA and $(1+1)$ EA* from (1). Comparing to $(1+1)$ EA, the bound on the number of generations only increases by an additive term $O(mn \log n)$.

Also note that the upper bound does not depend on the swarm size μ. However, when taking the number of constructed solutions as the performance measure instead of the number of generations, the bound has to be multiplied by μ.

Proof of Theorem 3. Assume that the global best is located on the i-th fitness level. As long as no improvement happens, for this particle its own best and the global best coincide. Hence, all bits in the particle are well-guided. By Lemma 5 the expected time until all n bits in x^{**} are frozen or an improvement happens in the meantime is $O(n \log(2n)) = O(n \log n)$.

If all bits in x^{**} are frozen, there is a probability of at least s_i that the new position for x^{**} is an improvement. The expected waiting time for an improvement is thus bounded from above by $1/s_i$ and, altogether, the expected time on the i-th fitness level is bounded by $O(n \log n) + 1/s_i$. Summing up expected waiting times for the first $m-1$ fitness levels proves the claim. □

In the proof of Theorem 3 we only relied on the global best for finding an improvement. It may happen, though, that other particles find improvements somewhere else or that other particles are attracted by the global best. The latter case can lead to a better estimate of the probability of an improvement. If there are several particles whose velocities have been frozen at borders corresponding to the global best, several particles search in the region around x^{**} in parallel. We make this precise for the binary PSO using only the social component, i.e., $c_1 = 0$. In this extreme case, all particles follow the global best and, as long as the global best is not exchanged, all bits are well-guided. After freezing, the swarm reduces the expected time for finding an improvement by a factor of order μ. This behavior resembles a $(1+\lambda)$ EA that creates $\lambda = \mu$ offspring in each generation.

Theorem 4. *Let A_i form the i-th fitness level of f and let s_i be the minimum probability for the $(1+1)$ EA leaving A_i going towards $A_{i+1} \cup \ldots \cup A_m$. If $\mu = \text{poly}(n)$, $c_1 = 0$, and $c_2 = \Theta(1)$, the expected number of generations for the binary PSO to optimize f is bounded from above by*

$$O\left(mn \log n + \frac{1}{\mu} \cdot \sum_{i=1}^{m-1} \frac{1}{s_i}\right).$$

Proof. We only need to prove that the expected number of generations for increasing the fitness from the i-th fitness level is at most $O(n \log n + 1/(\mu s_i))$.

Invoking Lemma 5 for all μn bits in the swarm, the expected time until all bits in the swarm are frozen to the bounds in x^{**} (or an improvement happened anyway) is $O(n \log(2\mu n)) = O(n \log n)$ as $\log(2\mu n) = O(\log n)$ follows from $\mu = \text{poly}(n)$.

Once all bits are frozen to the corresponding bounds of x^{**}, all particles behave equally until the next improvement. This implies that the binary PSO performs μ trials in each generation to create a solution with higher fitness and the probability for an improvement in one trial is bounded below by s_i. The probability that the binary PSO does not find an improvement within a period of $1/s_i$ trials is bounded by

$$1 - (1 - s_i)^{1/s_i} \geq 1 - \mathrm{e}^{-1} = \frac{\mathrm{e} - 1}{\mathrm{e}}.$$

The expected number of periods is therefore bounded by $\mathrm{e}/(\mathrm{e} - 1)$. The number of generations needed to have a period of $1/s_i$ trials equals $\lceil 1/(\mu s_i) \rceil$. Hence the expected number of generations for increasing the fitness is bounded by

$$\frac{\mathrm{e}}{\mathrm{e} - 1} \cdot \left(\frac{1}{\mu s_i} + 1\right) = O\left(\frac{1}{\mu s_i} + 1\right).$$

Adding the expected freezing time for the swarm yields the claimed bound $O(n \log n + 1/(\mu s_i))$ for fitness level i. □

The additive term $O(mn \log n)$ in the bounds from Theorems 3 and 4 results from the (pessimistic) assumption that on all fitness levels, the binary PSO has to wait until all velocities are frozen in order to find a better solution. Nevertheless, fitness-level arguments represent a powerful tool that can easily be applied to various problems. We present as an example an application for unimodal functions.

A function f is called unimodal if it has exactly one local optimum with respect to the Hamming distance. Hence, if the global best x^{**} is not the unique optimum, there is always at least one Hamming neighbor (a solution with Hamming distance 1 to x^{**}) with larger fitness. The probability for the $(1 + 1)$ EA* to create a specific Hamming neighbor as offspring equals $1/n \cdot (1 - 1/n)^{n-1} \geq 1/(\mathrm{e}n)$. We conclude that $s_i \geq 1/(\mathrm{e}n)$ for every non-optimal fitness level. Theorem 3 yields the following bound.

Corollary 1. *Let f be a unimodal function with m different function values. Then the expected number of generations for the binary PSO to optimize f is bounded by*

$$O\left(mn \log n + \sum_{i=1}^{m-1} \mathrm{e}n\right) = O(mn \log n + m \cdot \mathrm{e}n) = O(mn \log n).$$

5. The 1-PSO on OneMax

The function OneMax, defined by

$$\textsc{OneMax}(x) := \sum_{i=1}^{n} x_i,$$

simply counts the number of bits with value 1. Due to its simple structure, it is one of the most well-known test functions for randomized search heuristics. For several classes of heuristics, initial theoretical studies focused on the analysis of OneMax. Probably the first rigorous runtime analysis of an evolutionary algorithm was done for this function by Mühlenbein [16]. Droste [5] presented a first rigorous runtime analysis of an estimation-of-distribution algorithm on OneMax and other linear functions. Finally, Neumann and Witt [21] and Gutjahr [8] independently presented the first rigorous runtime analyses of an ant colony optimization algorithm on OneMax. It makes sense to continue the tradition and to consider the 1-PSO on OneMax in more detail.

Corollary 1 yields a bound $O(n^2 \log n)$ on the expected optimization time of the 1-PSO on OneMax. Compared to the bound $\Theta(n \log n)$ that holds for the (1+1) EA in this setting (see [16,6]), this seems relatively rough. To improve the $O(n^2 \log n)$ bound, we have to show that it is not necessary for the 1-PSO to spend $\Theta(n \log n)$ steps on each fitness level until all bits have been frozen to the velocity bounds of the global best.

In the following, we will improve the optimization time bound of the 1-PSO on OneMax to $O(n \log n)$. This implies that the 1-PSO has the same asymptotic upper runtime bound as the $(1 + 1)$ EA. For the proof, we will basically show that $O(\log n)$ steps adjusting the velocity entries are enough on each fitness level for the 1-PSO to attain almost the same probability of improving the current fitness as the (1+1) EA.

Consider a bit that has been well-guided towards the velocity bound $v_{\max}$ for t steps. Then the expected velocity value increases with t. This means that a bit whose entry in the global best has been fixed to 1 for a long time has a good chance of contributing to the OneMax-value of new solutions. We introduce a handy notation for such a bit.

Definition 3. A bit is called *t-strong* for $t \in \mathbb{N}_0$ if its (random) velocity $\bar{c}$-dominates the $\tilde{v}$-process at time t.

As a consequence from Lemma 2, we obtain the following statement.

Lemma 6. *Consider a bit that is currently t-strong and assume it to be well-guided towards value 1 throughout the next s steps. Then the bit will be $(t+s)$-strong afterwards.*

Proof. Let v_0 denote the current velocity at the bit and let v_s denote its velocity after s steps. By the definition of being strong and $\bar{c}$-dominance, $P(v_0 \geq a) \geq P(\tilde{v}_t \geq a + \bar{c})$ for all $a \in \mathbb{R}$. By Lemma 2 and the assumption that the x^{**}-entry stays fixed, $\bar{c}$-dominance is maintained for all following steps. After s steps, the current velocity v_s $\bar{c}$-dominates $\tilde{v}_{t+s}$; hence, by definition, the bit is $(t+s)$-strong. □

We take a fresh look at Lemma 4 and note that its last paragraph just repeats the definition of a t-strong bit. Let us recall the arguments leading to the lemma. The random velocity v_t of a t-strong bit gives us a random probability P_t of setting the bit considered to 1, i.e., a success. If we just plug in a single value for α in the lemma, we know that P_t respects the given bound on the success probability with probability at least $1 - \mathrm{e}^{-\alpha}$. However, if the bound is not respected, no immediate conclusion except the trivial bound $P_t \geq 1/n$ can be drawn.

Still, there is always a well-defined probability of a success at a t-strong bit. This probability equals the expectation of the random P_t since $E(P_t) = \int p \cdot f_{P_t}(p)\,\mathrm{d}p$ (or, if P_t has a discrete distribution, $E(P_t) = \sum p \cdot P(P_t = p)$). Since the distribution of P_t has already been characterized quite precisely in Lemma 4, it is not too difficult to derive good lower bounds on $E(P_t)$, the expected success probability, using the lemma. Note that $\bar{c} = c_1 + c_2 = 2$ and $c^* = \min\{1, \bar{c}/2\} = 1$ hold as the 1-PSO is considered.

Lemma 7. *The expected success probability of a t-strong bit, $t \geq 384 \ln n$, is at least $1 - O(1/t + 1/n)$.*

Proof. Note that the bound to be proven is never better than $1 - O(1/n)$. Hence, in the following, we ignore the upper bound $1 - 1/n$ on success probabilities and allow such a probability to become arbitrarily close to 1; the possible error introduced is $O(1/n)$. In order to apply Lemma 4, we consider a random variable X with support $\{1 - 96\mathrm{e}^2\alpha/t \mid 1 \leq \alpha \leq t/192,\ \alpha \in \mathbb{N}\}$, i.e., the support matches the bounds on P_t given in the lemma. We define a distribution for X according to

$$
\begin{aligned}
&P(X = 1 - 96\mathrm{e}^2/t) = 1 - \mathrm{e}^{-1},\\
&P(X = 1 - 96\mathrm{e}^2\alpha/t) = \mathrm{e}^{-\alpha+1} - \mathrm{e}^{-\alpha} \quad \text{for } 2 \leq \alpha \leq t/192,\\
&\text{and} \quad P(X = 1/n) = \mathrm{e}^{-t/192}.
\end{aligned}
$$

Obviously, all probabilities sum up to 1.

Using Lemma 4, it follows that the random variable corresponding to the success probability of the t-strong bit stochastically dominates X. It is therefore enough to bound the expected success probability according to X from below. Using the law of total probability, this expectation is at least

$$
\begin{aligned}
&(1 - \mathrm{e}^{-1}) \cdot \left(1 - \frac{96\mathrm{e}^2}{t}\right) + \sum_{\alpha=2}^{t/192} \left(\mathrm{e}^{-\alpha+1} - \mathrm{e}^{-\alpha}\right) \cdot \left(1 - \frac{96\mathrm{e}^2\alpha}{t}\right)\\
&\quad \geq (1 - \mathrm{e}^{-1}) \cdot \left(1 - \frac{96\mathrm{e}^2}{t}\right) + (\mathrm{e}^{-1} - \mathrm{e}^{-t/192}) - \sum_{\alpha=2}^{t/192} \mathrm{e}^{-\alpha+1} \cdot \frac{96\mathrm{e}^2\alpha}{t}\\
&\quad \geq (1 - \mathrm{e}^{-t/192}) \left(1 - \frac{96\mathrm{e}^2}{t}\right) - O\left(\frac{1}{t}\right) = 1 - O\left(\frac{1}{t}\right).
\end{aligned}
$$

Subtracting $O(1/n)$ for the possible error, we obtain the claim of the lemma. □

The expected success probability that we have just bounded is the actual probability of having a success at a t-strong bit. Hence, the lemma will be useful when we study OneMax and are interested in reproducing 1-bits. However, we need an additional statement. Given that a bit is t-strong, we are interested in the first point of time where the bit is actually set to 1. Assuming that the bit remains well-guided towards 1, the expected time until we have a success at such a bit decreases as t increases. Hence, we pessimistically assume that the bit is always exactly t-strong and wait for a success at the bit. More precisely, we are looking for the expected waiting time. If the success probability P_t was deterministic, say $P_t = p$ with probability 1, we could derive the expected waiting time by inspecting a geometrically distributed random variable, which would imply that the expectation would be $1/p$. However, since we are dealing with random probabilities, the expected waiting time is not necessarily the reciprocal of the expected success probability. We need more arguments, as described by the following lemma. Interestingly, since P_t follows a quite concentrated distribution, the following asymptotic upper bound is still the reciprocal of the bound derived in Lemma 7.

Lemma 8. *The expected time for a success at a t-strong bit, $t \geq 384 \ln n$, is bounded by $1 + O(1/t + 1/n)$.*

Proof. As in the proof of Lemma 7, we ignore the upper bound $1 - 1/n$ on success probabilities. Since the expected time for a success at success probability $1 - 1/n$ equals $1/(1 - 1/n) = 1 + O(1/n)$, the asymptotic upper bound of the lemma is not affected.

We reconsider the random variable X defined in the proof of Lemma 7. Since $t \geq 384 \ln n$, the last line of its definition implies $P(X = 1/n) \leq 1/n^2$. Since the success probability of the t-strong bit stochastically dominates X, it is enough to bound the expected time for a success according to X from above. Given that X has the value p, the waiting time for a success follows a geometric distribution with expectation $1/p$. We can bound the reciprocals of the single success probabilities according to $1/(1 - 96e^2\alpha/t) \leq 1 + 192e^2\alpha/t$ if n is not too small. By the law of total probability, the unconditional expected waiting time is at most

$$\begin{aligned}
&(1 - e^{-1}) \cdot \left(1 + \frac{192e^2}{t}\right) + \sum_{\alpha=2}^{t/192} \left(\left(e^{-\alpha+1} - e^{-\alpha}\right) \cdot \left(1 + \frac{192e^2\alpha}{t}\right)\right) + \frac{1}{n^2} \cdot n \\
&\quad \leq \left(1 + \frac{192e^2}{t}\right) + \sum_{\alpha=2}^{t/192} \left(\left(e^{-\alpha+1} - e^{-\alpha}\right) \frac{192e^2\alpha}{t}\right) + \frac{1}{n} \\
&\quad \leq \left(1 + \frac{192e^2}{t}\right) + \sum_{\alpha=2}^{t/192} \left(e^{-\alpha+1} \frac{192e^2\alpha}{t}\right) + \frac{1}{n} = 1 + O\left(\frac{1}{t} + \frac{1}{n}\right). \quad \square
\end{aligned}$$

During an improvement of the OneMax-value, some bits in x^{**} may be turned from 1 to 0, but a larger number of bits is turned from 0 to 1. We refer to the latter as new 1-bits being gained during this improvement. In the following analysis, we will consider random velocities of 1-bits gained while optimizing OneMax. Freshly gained 1-bits tend to be weaker than older ones. Sorting the bits from weak to strong, this is reflected by the following layering.

Definition 4. We say that an ordered sequence of bits $1, \ldots, k$ forms an *m-layer*, $m \in \mathbb{N}$, if and only if bit j, $1 \leq j \leq k$, is jm-strong. A set of k bits forms an m-layer if and only if it can be arranged as an m-layer.

For the following theorem, we will consider layers where the j-th bit is basically $\Theta(j \ln n)$-strong. Defining that a set of bits is successful if all have successes simultaneously, we show the following lemma. Note that we assume x^{**} to be fixed to 1 for these bits.

Lemma 9. *Let $k \leq n$ bits form a $(384 \ln n)$-layer. Then the expected time until all have a success simultaneously is bounded by $O(1)$.*

Proof. Recall that the success probability of a bit strictly increases with its velocity. We ignore the fact that velocities may increase while waiting for a success and pessimistically assume that the velocities remain fixed. However, the velocities and, therefore, the success probabilities at the k bits are still random variables. Let us consider such a random variable for some fixed bit. While waiting for a success, we consider the same random variable in all time steps. Hence, if we know the value of the variable at the beginning of our considerations, the variable will take the same value in all following time steps.

We claim that multiplying the expected success times for the single bits as bounded in Lemma 8 yields an upper bound on the expected time until all bits have successes simultaneously. Bits are independently set to 1 by the 1-PSO with the same (random) success probability over time; hence, conditioning on all outcomes of the k success probabilities, the waiting time follows a geometric distribution. Applying the law of total probability in the same way as in the proof of Lemma 8 proves the claim.

According to our assumption, the j-th bit, $1 \leq j \leq k$, is $384j \ln n$-strong. By Lemma 8, its expected success time is bounded by $1 + \frac{\kappa}{384j \ln n} + \frac{\kappa}{n}$ for some large constant κ. Taking the product over all j, we obtain

$$\prod_{j=1}^{k} \left(1 + \frac{\kappa}{384j \ln n} + \frac{\kappa}{n}\right) \leq e^{\sum_{j=1}^{n} \frac{\kappa}{384j \ln n} + \frac{\kappa}{n}},$$

which is $O(1)$ since $\sum_{j=1}^{n} 1/j = O(\ln n)$. $\square$

Now we can state the improved bound for OneMax.

Theorem 5. *The expected optimization time of the 1-PSO on* OneMax *is $O(n \log n)$.*

Proof. The basic proof idea is to keep track of the velocities of the newly gained 1-bits after improvements of the global best x^{**}. We wait on average $O(\log n)$ steps after an improvement and show that after that, the probability of improving is at least in the same order as for the (1+1) EA.

A difficulty with these arguments is that 1-bits in x^{**} may be set to 0 if the global best is exchanged. The current OneMax-value may only increase, but specific 1-bits may be lost if enough new 1-bits are gained somewhere else. We say that the lost 1-bits are reset. Resets may disturb the velocity increase on 1-bits as strong 1-bits may be replaced by weaker 1-bits.

In order to simplify the argumentation, we first describe an analysis for an idealized setting and then argue on how to extend the arguments to the real setting. Assume in the following that the 1-PSO does not accept resets of 1-bits, i.e., an improvement of the OneMax-value is only accepted if all 1-bits are set to 1 in the new global best.

We now divide a run of the 1-PSO into phases. Phase i for $0 \leq i \leq n-1$ starts with the end of the previous phase (Phase 0 starts after initialization) and it ends when the following two conditions are met:

1. The best-so-far OneMax-value is at least $i+1$.
2. At least $i+1$ 1-bits form an m-layer for $m := 384 \ln n$.

Note that the second condition will be fulfilled throughout the run as all 1-bits are maintained forever in our idealized setting and hence their velocities are well-guided and monotone increasing over time.

We claim that the expected time spent in Phase i is bounded above by $m + O(n/(n-i))$ for each $0 \leq i \leq n-1$. Note that phases may be empty. Moreover, when finishing Phase $n-1$ the global optimum has been found. Hence, the expected time for finding a global optimum is bounded by

$$\sum_{i=0}^{n-1} \left(m + O\left(\frac{n}{n-i}\right)\right) = O(n \log n) + O(n) \cdot \sum_{i=1}^{n} \frac{1}{i} = O(n \log n).$$

Consider the 1-PSO at the time it enters Phase i and assume pessimistically that the best-so-far OneMax-value is still i. As Phase $i-1$ has been completed, i 1-bits form an m-layer. According to Lemma 9, all these bits are set to 1 simultaneously after an expected number of $O(1)$ steps. Independently of these bits, the 1-PSO turns at least one of the $n-i$ 0-bits to 1 with a probability bounded below as follows. The probability of creating a 0-value at any 0-bit is at most $s(-v_{\max}) = 1 - 1/n$. The probability that not all $n-i$ 0-bits result in 0-values can be estimated, using $1+x \leq \exp(x) \leq 1+x/2$ for $-1 \leq x \leq 0$, by

$$1 - \left(1 - \frac{1}{n}\right)^{n-i} \geq 1 - \exp\left(-\frac{n-i}{n}\right) \geq \frac{n-i}{2n}.$$

The expected waiting time for this event is at most $2n/(n-i)$. As 1-bits and 0-bits are treated independently, we can multiply expectations using the justification from the proof of Lemma 9. Altogether, the expected time until constructing a solution with OneMax-value at least $i+1$ has been bounded from above by $O(n/(n-i))$.

Once the best-so-far OneMax-value has increased to at least $i+1$, the velocities on $i+1$ 1-bits are well-guided and monotone increasing. At the beginning of the phase, i 1-bits form an m-layer. In order to obtain an m-layer of $i+1$ bits, we can consider any other 1-bit and wait for it to become m-strong. As a consequence from Lemma 6, if x^{**} remains fixed for these $i+1$ bits, they form an m-layer of size $i+1$ after m steps. Together, the claimed bound $m + O(n/(n-i))$ follows for the expected time in Phase i. This also finishes the analysis for the idealized setting without resets.

A reset of a bit can destroy the velocity layers as a strong 1-bit might be exchanged by a weak 1-bit. In the worst case, such a new 1-bit is only 0-strong. If an improvement resets d bits, an m-layer of i bits may shrink to an m-layer of $i-d$ 1-bits. By an amortized analysis (see Chapter 17 in [3]), we wait for the velocities to recover so that we end up with an m-layer of i bits again.

Consider an improvement in a setting where k bits form an m-layer. A t-strong bit is reset with probability at most $O(1/t + 1/n)$ according to Lemma 7. The expected number of bits among these k layered bits reset during this improvement is therefore bounded from above by

$$\sum_{j=1}^{k} \left(\frac{O(1)}{384 j \ln n} + \frac{O(1)}{n} \right) = O(1).$$

Hence, an improvement prolongs the time spent in the current phase in expectation by $O(\ln n)$. Note that we can repeat the argumentation if another improvement occurs in the meantime since we only consider reset probabilities for all bits in a layer. As we can only have n improvements, we obtain an additional term $O(n \log n)$ in our runtime bound, which proves the time bound $O(n \log n)$ for the real setting. □

6. Experiments for OneMax

The expected runtime of the $(1+1)$ EA on OneMax is known to be of order $\Theta(n \log n)$. Theorems 2 and 5 show that the expected runtime of the 1-PSO is between $\Omega(n/\log n)$ and $O(n \log n)$. We conjecture that the runtime of the 1-PSO is of order $\Theta(n \log n)$ as well, but proving a corresponding lower bound seems difficult. We therefore perform experiments to get an impression of the actual runtimes for various problem dimensions and to compare the 1-PSO with the $(1+1)$ EA.

A drawback of the 1-PSO might be that the velocities of 1-bits increase only slowly, which may yield an increased waiting time for rediscovering previously gained 1-bits. On the other hand, the 1-PSO has the advantage that the velocities of 0-bits tend to $-v_{\max}$ only slowly as well, and so an incorrect decision may be reversed more easily. In contrast to the 1-PSO, the

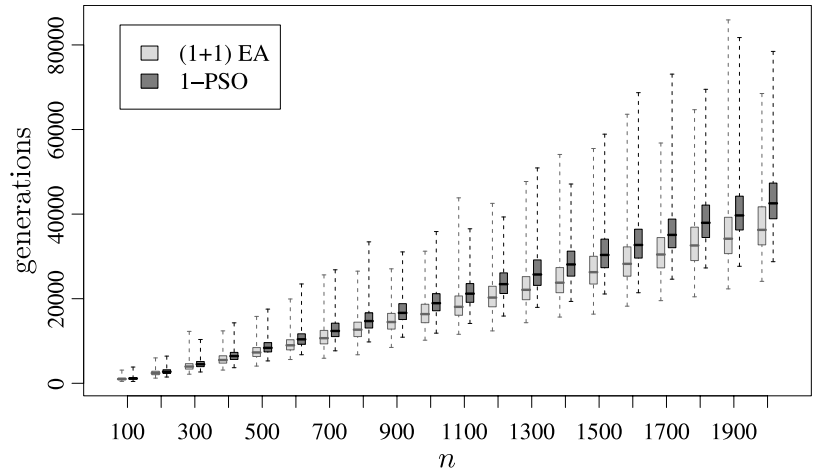

Fig. 2. Box-and-whisker plot of the runtimes of the $(1+1)$ EA (light gray) and the 1-PSO (dark gray) for $n = 100, 200, \ldots, 2000$ and 1000 runs for each setting. The whiskers show minimum and maximum values.

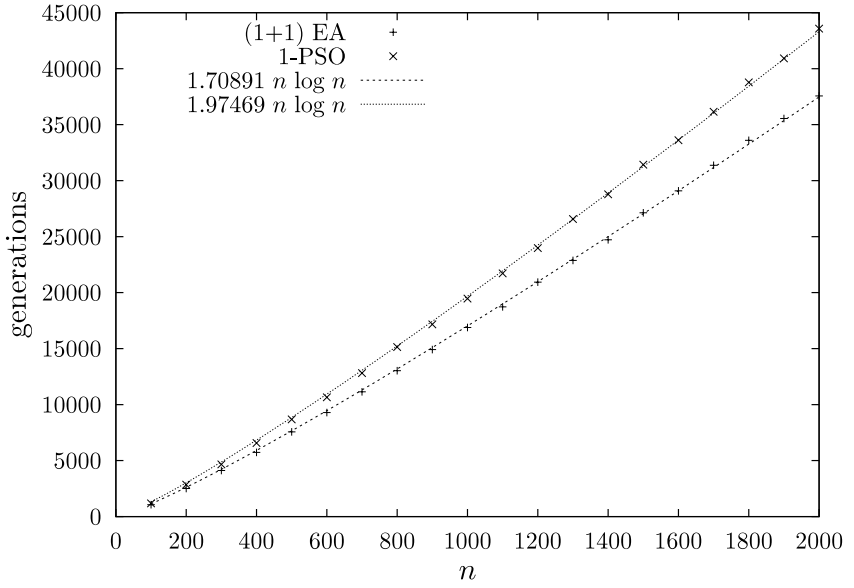

Fig. 3. Average runtimes for the $(1+1)$ EA and the 1-PSO and the fitted functions $1.70891n \log n$ for the $(1+1)$ EA and $1.97469n \log n$ for the 1-PSO.

$(1+1)$ EA only has a probability of $1/n$ of correcting an incorrectly set bit. Since it typically starts with a linear number of such bits, an argument similar to the Coupon Collector's Theorem [15] can be used to prove the lower bound $\Omega(n \log n)$. To conclude, the slow adaptation of velocities can have effects in opposite directions; we cannot tell a priori which effect will dominate the runtime and decide which algorithm performs better.

We therefore perform experiments for both the $(1+1)$ EA and the 1-PSO with problem dimensions $n = 100, 200, \ldots, 2000$. For each setting 1000 independent runs are performed and the number of generations until the optimum is found is measured. Fig. 2 shows box-and-whisker plots of the resulting runtimes. The box ranges from the first to the third quartile, the horizontal line within the box denotes the median, and the whiskers indicate minimum and maximum values.

One can observe that the median for the 1-PSO is consistently larger than the median for the $(1+1)$ EA. Even stronger, the boxes and the minimum values for the 1-PSO are consistently higher than those for the $(1+1)$ EA. Another observation is that the size of the boxes is quite small, which implies that the runtime has a low variance. The variances are similar for the two algorithms.

As this study focuses on expected runtimes, we investigate the average runtimes in more detail. Fig. 3 shows the average runtimes for the $(1+1)$ EA and the 1-PSO. A regression analysis was done using gnuplot 4.2 with the standard settings. Both data sets were fitted to functions $c \cdot n \log n$ for a constant c, log denoting the logarithm to the base 2. For the $(1+1)$ EA the function $1.70891n \log n$ resulted in the best fit and for the 1-PSO the function $1.97469n \log n$ was best. Comparing the two constants indicates that the expected runtime for the 1-PSO is about 15% larger than that for the $(1+1)$ EA.

Finally, to see whether the observed differences in runtime are statistically significant, we applied non-parametric Mann–Whitney tests to each sample of 1000 runs. For each problem dimension n, a two-sided test resulted in highly significant differences with $p \ll 0.0001$. A one-sided test confirmed that the differences were in favor of the $(1+1)$ EA in all cases.

7. Conclusions

We have considered the runtime behavior of the binary particle swarm optimizer by Kennedy and Eberhart. Thereby, we adapted the choice of the maximum velocity $v_{\max}$ to growing problem sizes and showed why this adaptation is essential when dealing with different problem sizes. For the resulting binary PSO we have proved a lower bound $\Omega(n/\log n)$ on the expected number of generations for every function where the global optimum is unique. This bound holds for almost every choice of the swarm size and the learning factors for the cognitive and the social component of PSO.

Moreover, we were able to transfer a fitness-level argument from the analysis of evolutionary algorithms (EAs) to PSO. This can be used as a tool for proving upper bounds for various parametrizations of the binary PSO and various functions. An example application to the class of all unimodal functions showed that the binary PSO is effective for these functions.

A more detailed analysis for the function OneMax revealed the runtime bound $O(n \log n)$ for the 1-PSO and hence the same upper bound as known for the $(1+1)$ EA. Experimental results suggest that the expected runtimes of the 1-PSO and of the $(1+1)$ EA have the same order of growth, but the 1-PSO is roughly 15% slower than the $(1+1)$ EA. The observed differences were confirmed by statistical tests.

Several open questions remain. One possible topic for future work is the impact of the swarm's neighborhood structure. Often the so-called *lbest* model is recommended [2] where the swarm is due to an underlying ring topology. Each particle is hence only connected to two other particles in the swarm. The effects of the topology on the global behavior of the swarm are not well-understood from a theoretical perspective.

Another obvious topic for future work is presenting rigorous theoretical analyses for PSO in continuous spaces. A first step has been made by Witt [28] for one of the few PSO variants that are guaranteed to converge to optima in continuous search spaces. Finally, the techniques used in this work might be extended towards other optimization paradigms such as estimation-of-distribution algorithms.

Acknowledgements

The authors thank Tobias Friedrich, Frank Neumann, and Pietro Oliveto for discussions that started the present paper. Financial support by the Deutsche Forschungsgemeinschaft (DFG) in terms of the Collaborative Research Center "Computational Intelligence" (SFB 531) is gratefully acknowledged. Dirk Sudholt was partially supported by a postdoctoral fellowship from the German Academic Exchange Service.

Appendix

Proof of Lemma 2. We assume that the distributions of $V_1^{(t)}$ and $V_2^{(t)}$ are continuous. If they are discrete or have discrete components, the following arguments can be proven in the same manner with a slight change of notation.

Suppose that we are at time t and that the current velocity x is sampled according to $V_1^{(t)}$. For any $a \in \mathbb{R}$, we are interested in the event $V_1^{(t+1)} \geq a$. If $x \geq a$, this happens with probability 1 since, by our assumptions, velocities are not decreased. If $x < a - \bar{c}$, the event is excluded since a velocity is increased by at most $\bar{c}$ in a step. Finally, if $x \in [a-\bar{c}, a)$, the binary PSO has to sample a 0-bit, which happens with probability $s(-x)$, and choose $r_1 + r_2 \geq a - x$ afterwards, where (slightly abusing notation) $r_1 \in U[0, c_1]$ and $r_2 \in U[0, c_2]$. Let $G(s) := P(r_1 + r_2 \geq s)$ denote the distribution function for the sum of r_1 and r_2. Writing $f_1^{(t)}(x)$ for the density function of $V_1^{(t)}$ at x, our considerations yield

$$P(V_1^{(t+1)} \geq a) = P(V_1^{(t)} \geq a) + \int_{a-\bar{c}}^{a} f_1^{(t)}(x) \cdot s(-x) \cdot G(x - a + \bar{c})\, \mathrm{d}x \tag{2}$$

for every $a \in \mathbb{R}$. Analogously, we have

$$P(V_2^{(t+1)} \geq a + \bar{c}) = P(V_2^{(t)} \geq a + \bar{c}) + \int_{a}^{a+\bar{c}} f_2^{(t)}(x) \cdot s(-x) \cdot G(x - a)\, \mathrm{d}x. \tag{3}$$

Using an index transformation, we rewrite (2) as

$$P(V_1^{(t+1)} \geq a) = P(V_1^{(t)} \geq a) + \int_{a}^{a+\bar{c}} f_1^{(t)}(x - \bar{c}) \cdot s(-x + \bar{c}) \cdot G(x - a)\, \mathrm{d}x \tag{4}$$

as we will concentrate on a comparison of the integrals.

Using the abbreviation $\Delta^{(t)}(a) := P(V_1^{(t)} \geq a) - P(V_2^{(t)} \geq a + \bar{c})$, our aim is to show that $\Delta^{(t)}(a) \geq 0$ implies $\Delta^{(t+1)}(a) \geq 0$. We obtain a pessimistic estimation of $\Delta^{(t+1)}(a)$ if we estimate $P(V_2^{(t+1)} \geq a + \bar{c})$ from above and $P(V_1^{(t+1)} \geq a)$ from below. Since $s(-x+\bar{c}) \geq s(-a) \geq s(-x)$ for $x \in [a, a+\bar{c}]$, we obtain appropriate estimations by replacing all sigmoid terms in (4)

and (3) by $s(-a)$. Hence,

$$\begin{aligned}\Delta^{(t+1)}(a) &= P(V_1^{(t+1)} \geq a) - P(V_2^{(t+1)} \geq a + \bar{c}) \\ &\geq \Delta^{(t)}(a) + \int_a^{a+\bar{c}} (f_1^{(t)}(x - \bar{c}) - f_2^{(t)}(x)) \cdot s(-a) \cdot G(x - a)\,\mathrm{d}x \\ &= \Delta^{(t)}(a) + \int_a^{a+\bar{c}} g(x) \cdot s(-a) \cdot G(x - a)\,\mathrm{d}x, \end{aligned} \tag{5}$$

where $g(x) := f_1^{(t)}(x - \bar{c}) - f_2^{(t)}(x)$. Recall that our aim is to bound (5) by at least 0. To this end, it is helpful to also express $\Delta^{(t)}(a)$ in terms of the density functions. After the same index transformation as for (4), we obtain

$$\Delta^{(t)}(a) = \int_{a+\bar{c}}^{v_{\max}+\bar{c}} (f_1^{(t)}(x - \bar{c}) - f_2^{(t)}(x))\,\mathrm{d}x = \int_{a+\bar{c}}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x.$$

Now (5) can be written with unified integrals as

$$\begin{aligned}\Delta^{(t+1)}(a) &= \int_{a+\bar{c}}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x + \int_a^{a+\bar{c}} g(x) \cdot s(-a) \cdot G(x - a)\,\mathrm{d}x \\ &= \int_a^{v_{\max}+\bar{c}} g(x) \cdot w(x)\,\mathrm{d}x, \end{aligned} \tag{6}$$

where the weight function $w(x)$ is defined by $w(x) := s(-a) \cdot G(x - a)$ for $x \in [a, a + \bar{c}]$ and $w(x) := 1$ for $x \geq a + \bar{c}$. The aim is still to bound the right-hand side in (6) by at least 0. This is done by considering portions of the integral and bounding these terms from below. Note that $0 \leq w(x) \leq 1$ for $x \geq a$ and that $w(x)$ increases monotonically in x.

The function $g(x)$ may be positive or negative on certain portions of the integral. In the unweighted case, without $w(x)$, we have for every $b \in \mathbb{R}$

$$\int_b^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x = \Delta^{(t)}(b - \bar{c}) \geq 0 \tag{7}$$

by our assumptions for time t. This implies that $g(x)$ must be positive for x sufficiently close to $v_{\max} + \bar{c}$. Hence, if there is a root of $g(x)$ in the interval $[a, v_{\max} + \bar{c}]$, then $g(x) \geq 0$ if x is greater than the largest root. If even $g(x) \geq 0$ within the whole interval $[a, v_{\max} + \bar{c}]$ then

$$\int_a^{v_{\max}+\bar{c}} g(x) \cdot w(x)\,\mathrm{d}x \geq \int_a^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x \geq 0.$$

If $g(x)$ changes signs in the interval, we denote by $r^{**} < r^*$ the largest two roots of $g(x)$ in the interval (defining $r^{**} = a$ if there is only a single root) and recall that $g(x) \geq 0$ for $x \geq r^*$. Here it is crucial that $w(x)$ increases, implying that

$$\begin{aligned}\int_{r^{**}}^{v_{\max}+\bar{c}} g(x) \cdot w(x)\,\mathrm{d}x &= \underbrace{\int_{r^{**}}^{r^*} g(x) \cdot w(x)\,\mathrm{d}x}_{\leq 0} + \underbrace{\int_{r^*}^{v_{\max}+\bar{c}} g(x) \cdot w(x)\,\mathrm{d}x}_{\geq 0} \\ &\geq \int_{r^{**}}^{r^*} g(x) \cdot w(r^*)\,\mathrm{d}x + \int_{r^*}^{v_{\max}+\bar{c}} g(x) \cdot w(r^*)\,\mathrm{d}x \\ &= w(r^*) \cdot \int_{r^{**}}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x \geq 0, \end{aligned}$$

the last inequality following from (7). If there are further roots (say, r^{***} and r^{****}), this argument can be iterated to yield $\int_{r^{****}}^{r^{**}} g(x) \cdot w(x)\,\mathrm{d}x \geq 0$, which inductively proves $\int_a^{v_{\max}+\bar{c}} g(x) \cdot w(x)\,\mathrm{d}x \geq 0$ for every number of roots. □

References

[1] N. Attiratanasunthron, J. Fakcharoenphol, A running time analysis of an ant colony optimization algorithm for shortest paths in directed acyclic graphs, Information Processing Letters 105 (3) (2008) 88–92.
[2] D. Bratton, J. Kennedy, Defining a standard for particle swarm optimization, in: Proc. of Swarm Intelligence Symposium, SIS 2007, IEEE Press, 2007, pp. 120–127.
[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd edition, The MIT Press, 2001.
[4] B. Doerr, F. Neumann, D. Sudholt, C. Witt, On the runtime analysis of the 1-ANT ACO algorithm, in: Proc. of GECCO'07, ACM Press, 2007, pp. 33–40.
[5] S. Droste, A rigorous analysis of the compact genetic algorithm for linear functions, Natural Computing 5 (3) (2006) 257–283.
[6] S. Droste, T. Jansen, I. Wegener, On the analysis of the (1 + 1) evolutionary algorithm, Theoretical Computer Science 276 (2002) 51–81.
[7] O. Giel, I. Wegener, Evolutionary algorithms and the maximum matching problem, in: Proc. of STACS'03, in: LNCS, vol. 2607, 2003, pp. 415–426.
[8] W.J. Gutjahr, First steps to the runtime complexity analysis of ant colony optimization, Computers and Operations Research 35 (9) (2008) 2711–2727.

[9] W.J. Gutjahr, G. Sebastiani, Runtime analysis of ant colony optimization with best-so-far reinforcement, Methodology and Computing in Applied Probability 10 (2008) 409–433.
[10] W. Hoeffding, Probability inequalities for sums of bounded random variables, American Statistical Association Journal 58 (301) (1963) 13–30.
[11] C. Horoba, D. Sudholt, Running time analysis of ACO systems for shortest path problems, in: Proc. of SLS 2009, in: LNCS, vol. 5752, 2009, pp. 76–91.
[12] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proc. of the IEEE International Conference on Neural Networks, IEEE Press, 1995, pp. 1942–1948.
[13] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, in: Proc. of the World Multiconference on Systemics, Cybernetics and Informatics, WMSCI, 1997, pp. 4104–4109.
[14] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufmann, 2001.
[15] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[16] H. Mühlenbein, How genetic algorithms really work: Mutation and hillclimbing, in: Parallel Problem Solving from Nature, PPSN II, 1992, pp. 15–26.
[17] Frank Neumann, Dirk Sudholt, Carsten Witt, Analysis of different MMAS ACO algorithms on unimodal functions and plateaus, Swarm Intelligence 3 (1) (2009) 35–68.
[18] F. Neumann, I. Wegener, Minimum spanning trees made easier via multi-objective optimization, Natural Computing 5 (3) (2006) 305–319.
[19] F. Neumann, I. Wegener, Randomized local search, evolutionary algorithms, and the minimum spanning tree problem, Theoretical Computer Science 378 (1) (2007) 32–40.
[20] F. Neumann, C. Witt, Ant Colony Optimization and the minimum spanning tree problem, in: Proceedings of Learning and Intelligent Optimization, LION'07, in: LNCS, vol. 5313, Springer, 2008, pp. 153–166.
[21] F. Neumann, C. Witt, Runtime analysis of a simple Ant Colony Optimization algorithm, Algorithmica 54 (2) (2009) 243–255.
[22] P.S. Oliveto, J. He, X. Yao, Time complexity of evolutionary algorithms for combinatorial optimization: a decade of results, International Journal of Automation and Computing 4 (3) (2007) 281–293.
[23] J. Reichel, M. Skutella, Evolutionary algorithms and matroid optimization problems, in: Proc. of GECCO'07, ACM Press, 2007, pp. 947–954.
[24] J. Scharnow, K. Tinnefeld, I. Wegener, The analysis of evolutionary algorithms on sorting and shortest paths problems, Journal of Mathematical Modelling and Algorithms (2004) 349–366.
[25] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: Proc. of the Seventh Annual Conference on Evolutionary Programming, 1998, pp. 591–600.
[26] I. Wegener, Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions, in: R. Sarker, X. Yao, M. Mohammadian (Eds.), Evolutionary Optimization, Kluwer, 2002, pp. 349–369.
[27] C. Witt, Worst-case and average-case approximations by simple randomized search heuristics, in: Proc. of STACS '05, in: LNCS, vol. 3404, 2005, pp. 44–56.
[28] C. Witt, Why standard particle swarm optimisers elude a theoretical runtime analysis, in: Foundations of Genetic Algorithms 10, FOGA'09, ACM Press, 2009, pp. 13–20.
[29] Christian Horoba, Dirk Sudholt, Ant colony optimization for stochastic shortest path problems, in: Proc. of GECCO'10, 2010 (in press).