UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

The Cooperative Coevolutionary (1+1) EA

Thomas Jansen R. Paul Wiegand

No. CI-145/03

Technical Report ISSN 1433-3325 March 2003

Secretary of the SFB 531 · University of Dortmund · Dept. of Computer Science/XI
44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, "Computational Intelligence," at the University of Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

The Cooperative Coevolutionary (1+1) EA

Thomas Jansen

Fachbereich Informatik
Universität Dortmund
44221 Dortmund, Germany
Thomas.Jansen@udo.edu

R. Paul Wiegand*

Computer Science Department
George Mason University
Fairfax, VA 22030, USA
paul@tesseract.org

Abstract

Coevolutionary algorithms are a variant of evolutionary algorithms which are aimed for the solution of more complex tasks than traditional evolutionary algorithms. One example is a general cooperative coevolutionary framework for function optimization. A thorough and rigorous introductory research in which the optimization potential of cooperative coevolution is studied is presented.

Using the cooperative coevolutionary framework as a starting point, the CC (1+1) EA is defined and investigated. The main interest is in the analysis of the expected optimization time. The research concentrates on separability since this is a key property of objective functions. It is shown that separability alone is not sufficient to yield any advantage of the CC (1+1) EA over its traditional, non-coevolutionary counterpart. Such an advantage is demonstrated to have one basis in the increased explorative possibilities of the cooperative coevolutionary algorithm. For inseparable functions, the cooperative coevolutionary set-up can be harmful. We prove that for some objective functions the CC (1+1) EA fails to locate a global optimum with probability converging to 1 exponentially fast, even in infinite time; however, inseparability alone is not sufficient for an objective function to cause difficulties. It is demonstrated that the CC (1+1) EA may perform equal to its traditional counterpart and even may outperform it on certain inseparable functions.

When implementing the CC (1+1) the use of a parallel computer makes a big difference. For sequential and parallel implementations

*The author was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

different variants of the algorithm are more natural. It is proved that both variants are equivalent for separable objective functions but can show very different performance on inseparable functions. The two variants are compared when applied to the approximation of a carefully designed example problem.

1 Introduction

An increasingly more common extension to traditional evolutionary algorithms are so-called coevolutionary algorithms, in which individuals obtain fitness values corresponding to how well they behave in conjunction with other individuals. Indeed, coevolution often seems to be a plausible way to approach many types of problems, especially when no clear objective measure can be obtained, or when problems may benefit from partitioning potential solutions into smaller components that can be solved separately. It seems reasonable that the potential advantages with this second group may benefit many kinds of optimization applications. However, while it is clear that one can apply coevolution towards optimization problems, it is unclear how efficient it may be when compared to traditional evolutionary algorithms (EAs).

The contextual nature of the fitness evaluation in coevolutionary algorithms have frequently presented researchers with a great deal of consternation. Coevolutionary dynamics can be very complicated (Ficici and Pollack 2000; Wiegand, Liles, and De Jong 2002b), and even relatively straight forward issues in a traditional EA surrounding representation and progress measurement can be quite difficult in many types of coevolutionary algorithms (Cliff and Miller 1995; Ficici and Pollack 1998; Stanley and Miikkulainen 2002). As a result, most analytical efforts of coevolutionary algorithms have focussed on aspects of coevolutionary dynamics, representation, and progress measurement, while very little effort has been placed in establishing the efficiency of coevolutionary approaches as optimization methods. In this paper, we seek to bridge this gap by concentrating our attention on the analysis of the performance of a specific coevolutionary algorithm as an optimizer.

In order to focus on this question of optimization, we adopt the well-known cooperative coevolutionary framework provided by Potter and De Jong (1994), which has several advantages that suit our interests. First, questions of objective progress measurement are greatly simplified. Second, the framework provides a very general architecture for optimization applications. Additionally, the method has been applied successfully to a variety of applications (e.g., Eriksson and Olsson (1997, Leung, Wong, and King (1998, Iorio and Li (2002)). Finally, the framework allows that any evolutionary algorithm can be used as a piece of the architecture.

These cooperative coevolutionary algorithms (CCEAs) work by applying several EAs in an almost independent way to *components* of a larger, objective problem. Fitness is assessed by assembling an individual component with representative components from other EA populations. This process is static and symmetric in the sense that each EA has a specific role to play in the problem that does not alter during a given run, and a specific assembled string will receive the same reward, regardless of which component is currently being evaluated. As a result, objective progress measurement is not an obstacle to our analysis.

What remains an important issue, however, is representation. How one divides the problem representation into these static components is still very much a part of the design engineer's duties, and will certainly impact the efficiency of the optimization process in many cases. We concentrate our attention on maximization of pseudo-Boolean functions, $f : \{0,1\}^n \mapsto \mathbb{R}$, and make fairly natural and obvious decompositional decisions with respect to assignment of roles of the various EA populations. A bit string $x \in \{0,1\}^n$ of length n is divided into k disjoint components $x^{(1)}, \ldots, x^{(k)}$. Thus, there are k EAs, each operating on one of these k components. The choice of the underlying EA is of obvious importance, and will undoubtedly influence the performance of the CCEA. We concentrate our attention on the well-known (1+1) EA, since it is perhaps the simplest EA that still shares many important properties with more complex EAs.

The main question with respect to the decomposition is how the degree to which this decomposition matches the true separability of the problem might affect run time performance. Since the individual components are in some sense treated almost independently of one another, it is plausible to believe that CCEA may be able to exploit this property of f when it is appropriately divided, or may be hindered by properties of interdependency when f is poorly decomposed. Indeed, intuitively one might expect that the advantage of a CCEA over an EA grows with the degree of separability of the problem; however, we will show that separability alone is insufficient for the CCEA to gain an advantage. Moreover, we investigate this property of separability of the objective function both in the case when the problem is separable across population boundaries, as well as when it is not. In combination with problem division, the CCEA brings with it the potential for more focussed exploration of the individual components. Important parameters such as the mutation probability are often related to the length of the string being searched (e.g, $1/n$ for string length n). Since individuals in the sub-populations represent only components of a complete solution, and are therefor only a fraction of the search space, parameters like mutation probability often have more dramatic potential due to their increased rates. This enables the CCEA to search these components with greater exploratory power, while protecting the other components from the added disruption of this exploration by the nature of the problem decomposition. We will present a class of functions where this becomes very clear. Moreover, we present an example where we achieve an exponential separation between the EA and the CCEA in terms of optimization performance.

In the next section, we give precise definitions of the (1+1) EA, the CC (1+1) EA, the notion of separability, and the notion of expected optimization time. In the third section, we consider problems that are separable with respect to the population boundary. We first show that the CC (1+1) EA has surprisingly no advantage over the (1+1) EA on linear functions, despite the fact that such functions are fully separable. Then we present a class of functions that demonstrates that it is the explorative advantage in *conjunction* with the problem decomposition that leads to CC (1+1) EA superiority. We conclude this section by constructing an example that illustrates that it is possible for the (1+1) EA to perform better than the CC (1+1) EA, even when the problem is separable across the population boundaries. In Section 4, we consider the situation in which the problem is not separable across the population boundaries. We first demonstrate that there exist inseparable problems for whichthe CC (1+1) EA cannot find the global optimum. We follow this by next demonstrating that inseparability itself is an insufficient obstacle to performance by identifying a class of inseparable problems that are no more difficult for the CC (1+1) EA than for the (1+1) EA. We conclude this section by showing that there remain advantages to the CC (1+1) EA on some problems, despite inseparability, and that this advantage can be exponential. In Section 5, we consider a more parallel alternative to our sequential implementation of the CC (1+1) EA. We begin by showing there is no analytical difference between these two algorithms when the problem is separable across the population boundaries, and that both algorithms can be similarly prevented from global convergence by some inseparable functions.

We then present an inseparable function that clearly demonstrates a performance difference between the two algorithms. In our final section, we offer a short summary of the things learned by this research and a brief discussion of possible future directions for research.

2 Definitions

We choose to instantiate the cooperative coevolutionary function optimization framework of Potter and De Jong (1994) with the (1+1) EA as underlying search heuristic. This extremely simple evolutionary algorithm uses a population of size one, produces one offspring in each generation via bit-wise mutation and applies plus-selection known from evolution strategies: the offspring replaces its parent iff its fitness is at least as large. The advantage of choosing such a simple EA as underlying search heuristic is that the resulting cooperative coevolutionary algorithm (CCEA) is easier to analyze. Our motivation for choosing the (1+1) EA stems from the wealth of known analytical results (see for example Mühlenbein (1992), Rudolph (1997), Garnier, Kallel, and Schoenauer (1999), Droste, Jansen, and Wegener (2002), Scharnow, Tinnefeld, and Wegener (2002)) and tools and methods (see for example Wegener (2002)). We present a formal definition of the (1+1) EA in a form that is suitable for maximization of a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$.

Algorithm 1 (1+1 Evolutionary Algorithm ((1+1) EA)).

1. **Initialization**
   Choose $x_0 \in \{0,1\}^n$ *uniformly at random.*
2. $t := 0$
3. **Mutation**
   Create $y \in \{0,1\}$ *by copying* x_t *and, independently for each bit, flip this bit with probability* $\min\{1/n, 1/2\}$.
4. **Selection**
   If $f(y) \geq f(x_t)$, *then set* $x_{t+1} := y$, *else set* $x_{t+1} := x_t$.
5. $t := t + 1$
6. *Continue at line 3.*

We consider the (1+1) EA without stopping criterion and are mainly interested in the first point of time when a global optimum of f is encountered. We measure time by counting function evaluations and assume that the number of function evaluations is an accurate measure for the actual computation time. Note that the function value of the parent $f(x)$ can be assumed to be known in line four, since it has been computed in the previous generation. Thus, the number of function evaluations is larger than the number of generations by exactly one.

Definition 2. *Consider some randomized algorithm A optimizing some function $f\colon \{0,1\}^n \to \mathbb{R}$. Let the random variable T denote the number of function evaluations A makes before evaluating some f-optimal $x \in \{0,1\}^n$ for the first time. We call T the* optimization time *of A on f and $E(T)$ the* expected optimization time *of A on f. We call Prob$(T \leq t)$ the* success probability *of A on f after t steps.*

We use the (1+1) EA as the underlying search heuristic and obtain a cooperative coevolutionary (1+1) EA. We choose to use a numbering for the independent EA populations and make them *active* in this ordering. We consider it to be the most natural implementation for a sequential computing environment.

Algorithm 3 (Cooperative Coevolutionary (1+1) Evolutionary Algorithm (CC (1+1) EA)).

1. **Initialization**
   Independently for each $i \in \{1, \ldots, k\}$, choose $x_0^{(i)} \in \{0,1\}^l$ uniformly at random.
2. $t := -1$
3. $a := 1$; $t := t + 1$
4. **Mutation**
   Create $y^{(a)}$ by copying $x_t^{(a)}$ and, independently for each bit, flip this bit with probability $\min\{1/l, 1/2\}$.
5. **Selection**
   If $f(x_{t+1}^{(1)} \cdots y^{(a)} \cdots x_t^{(k)}) \geq f(x_{t+1}^{(1)} \cdots x^{(a)} \cdots x_t^{(k)})$, set $x_{t+1}^{(a)} := y^{(a)}$, else set $x_{t+1}^{(a)} := x_t^{(a)}$.
6. $a := a + 1$
7. *If $a > k$, then continue at line 3, else continue at line 4.*

Note that each (1+1) EA may have to make two function evaluations in each generation. Since the current version of the component from the other EAs is used to compute the function value, each EA may use a different parent bit string for the selection. Thus, the number of function evaluations is almost twice as large as the number of generations summed over all (1+1) EAs.

After k consecutive generations, each (1+1) EA was active exactly once. Therefore, we denote k consecutive generations as a *round*. Obviously, the number of function evaluations can be estimated quite accurately by $2k$ times the number of rounds. However, the factor of two is not important since we employ asymptotic analysis. For the sake of completeness, we give definitions of the notions we use to describe the asymptotic growth of functions.

Definition 4. *Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}$ be two functions. We say $f = O(g)$, if*

$$\exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+\colon \forall n \geq n_0\colon f(n) \leq c \cdot g(n)$$

holds. We say $f = \Omega(g)$, if $g = O(f)$ holds. We say $f = \Theta(g)$, if $f = O(g)$ and $f = \Omega(g)$ both hold. We say $f = o(g)$, if $\lim\limits_{n\to\infty} f(n)/g(n) = 0$ holds. We say $f = \omega(g)$, if $g = o(f)$ holds.

When optimizing a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$ using a CCEA, each bit string is divided into different components. It is well known that for some functions it is possible to find such a division in a way that the separate components do not interfere with each other. Such functions are called separable. Since separability is a key issue we have to discuss when analyzing the performance of the CC (1+1) EA, we give a precise formal definition.

Definition 5. *A function $f\colon \{0,1\}^n \to \mathbb{R}$ is called (r,s)*-separable*, where $r, s \in \{1, 2, \ldots, n\}$, if there exists a partition of $\{1, \ldots, n\}$ into r disjoint sets $I_1, \ldots, I_r$, and if there exist a matching number of pseudo-Boolean functions $g_1, \ldots, g_r$ with $g_j\colon \{0,1\}^{|I_j|} \to \mathbb{R}$ such that*

$$\forall x = x_1 \ldots x_n \in \{0,1\}^n\colon f(x) = \sum_{j=1}^{r} g_j \left(x_{i_{j,1}} x_{i_{j,2}} \cdots x_{i_{j,|I_j|}} \right)$$

holds, $I_j = \left\{ i_{j,1}, \ldots, i_{j,|I_j|} \right\}$ and $|I_j| \leq s$ for all $j \in \{1, \ldots, r\}$.

We say f is exactly (r,s)-separable *if f is (r,s)-separable but not (r', s')-separable for any $r' > r$ or $s' < s$.*

Obviously, the two parameters r and s are not totally unrelated. We have $n/r \leq s \leq n-(r-1)$. We consider s to be the more important parameter with respect to the potential performance of an evolutionary algorithm optimizing f. The parameter s gives an upper bound on the dimension of the search space for each of the sub-functions g_j. It is obvious that this dimension of the search space has large influence on the potential difficulty of f. Consider for example the case where we have $s = O(\log n)$. Then the size of each search space of the r sub-functions is polynomial in n and f can be optimized using exhaustive search of the separate search spaces in polynomial time.

Note, however, that a high degree of separability corresponds to a small value of s. For $(n, 1)$-separable functions, the function value can be computed depending on single bits. An exactly $(1, n)$-separable function is not separable at all, the values of all n bits have to be known together in order to compute the function value.

As discussed in Section 1, we concentrate on the division of a bit string $x \in \{0, 1\}^n$ into k separate components of equal size n/k. Thus, $(k, n/k)$-separable functions are of special interest to us. We consider this case to be the most interesting. Cases with a division into sub-functions of varying dimension can be analyzed based on the results for the homogeneous decomposition.

When discussing the separability of a function, it is useful to note that each pseudo-Boolean function $f\colon \{0, 1\}^n \to \mathbb{R}$ has a unique representation as polynomial

$$f(x) = \sum_{I \subseteq \mathcal{P}(\{1,\dots,n\})} w_I \cdot \prod_{i \in I} x_i$$

with $w_I \in \mathbb{R}$. The w_I are called weights and it is obvious that a non-zero weight w_I implies that all the bits x_i with $i \in I$ cannot be separated.

Since an important part of our research concentrates on the relationship between the separability properties of a function and the decomposition chosen when implementing a CC (1+1) EA, it is often necessary to distinguish between these two concepts. We use the term *component* to mean a part of a representation, given by the algorithm, while we use the term *piece* to mean portions of the problem itself.

The difference between these terms is more obvious in some places than others, since the algorithm's decomposition may be closely aligned with the problem's true separation, or it may not. It will be helpful to provide terms to distinguish between such cases. When the problem pieces are separable between the components represented in the populations, we say that the decomposition *matches* the separability of the problem. More specifically, we say that the decomposition *exactly matches* the separability of the problem when the pieces as exactly separable. We define these terms more formally below.

Definition 6. *Let a function* $f : \{0,1\}^n \mapsto \mathbb{R}$ *be* (r,s)*-separable as in Definition 5. We say that a decomposition* matches *the separability of* f *if all bits that belong to one index set* I_j *are in one population.*

We say that the decomposition exactly matches *the separability of* f *if there are* r *components, and each EA operates on the bits contained in exactly one of the index sets* I_j*.*

Informally, we may also use the phrase *separable across population boundaries* to describe situations in which the algorithm's decomposition matches the separability of the problem. When the decomposition does not match the problem separation, we use the phrase *cross-population nonlinearities* to refer to the existence of nonlinear relationships between components resulting from decompositions that place portions of inseparable pieces in different populations. It should be clear that when decompositions do not match the problem's separability, there will be such nonlinearities by definition.

3 Exploiting separability

When using the cooperative coevolutionary function optimization framework by Potter and De Jong (1994) one has to decide how to separate a bit string into different components that are distributed to the separate EAs. In this section we concentrate on cases where the decomposition matches the separability of the objective function.

It makes sense to begin the investigation with an extreme case. If one believes that the CCEA framework is advantageous due to (explicitly) exploiting the separability of the objective function, one may speculate that this becomes most visible when the objective function is separable to an extreme degree. Thus, we begin our investigations with $(n,1)$-separable functions. Such functions can be written as $f(x) = w_0 + w_1 \cdot x_1 + \cdots + w_n \cdot x_n$ with fixed weights $w_0, w_1, \ldots, w_n \in \mathbb{R}$ and are known as linear functions. A particularly well known and well investigated linear function is OneMax, which can be defined by $w_0 = 0$ and $w_1 = \cdots = w_n = 1$. We will see in the next subsection that in spite of the maximal degree of separability the CC (1+1) EA has no advantage over the traditional (1+1) EA on linear functions. This motivates the search for reasons why the cooperative coevolutionary optimization framework fails to provide an advantage on linear functions. We define a class of functions in Section 3.2 that are exactly $(k, n/k)$-separable, where k is a parameter. We will see that the CC (1+1) EA possesses increased explorative possibilities, which can lead to an impressive speed-up compared to the (1+1) EA. The main results of Sections 3.1 and 3.2 have been published in (Jansen and Wiegand 2003), which proved a lower bound for the CC (1+1) EA for linear functions, as well as an upper bound for the (1+1) EA and lower bound for the CC (1+1) EA for a class of exactly $(k, n/k)$-separable example functions. Here, we give a more complete picture with upper and lower bounds for both algorithms. Finally, in Section 3.3, we investigate whether the attempt to exploit the separability of an objective function by means of the cooperative coevolutionary framework can possibly lead to an increase in expected optimization time.

3.1 Linear functions

The analysis of the expected optimization time of evolutionary algorithms is a difficult task. This is true even for simple objective functions like linear functions, and it is still true when considering a simple evolutionary algorithm like the (1+1) EA. Nevertheless, there are many results concerned with the expected optimization time of the (1+1) EA on different objective functions known. For linear functions, the expected optimization time is $O(n \log n)$ and this upper bound matches the lower bounds for typical linear functions. We cite the result by Droste, Jansen, and Wegener (2002).

Theorem 7. *The expected optimization time of the (1+1) EA on a linear function* $f: \{0,1\}^n \to \mathbb{R}$ *is* $O(n \log n)$*. If* f *only has non-zero weights, the expected optimization time of the (1+1) EA on* f *is* $\Theta(n \log n)$*.*

We consider the CC (1+1) EA on linear functions. Obviously, any decomposition matches the separability of linear or $(n, 1)$-separable functions. We consider any homogeneous decomposition where the number of components k divides n. This allows us to ignore special cases where the last component is either larger or smaller than all the other components. Such cases are not very different from the cases we discuss and have no properties of special interest. Since we are escpecially interested in seeing how much the cooperative coevolutionary function optimization framework increases the efficiency, we begin our investigations with a lower bound on the expected optimization time of the CC (1+1) EA on a linear function.

Theorem 8. *The expected optimization time of the CC (1+1) EA on a linear function $f\colon \{0,1\}^n \to \mathbb{R}$ with only non-zero weights is $E(T) = \Omega(n \log n)$.*

Proof. Since all weights are non-zero there is a unique global optimum x^*. Let x_t denote the current string of the CC (1+1) EA after t complete rounds. Let p_t be the probability that $x_t \neq x^*$ after t complete rounds. It is easy to see that $\mathrm{E}(T) \geq p_t \cdot t \cdot k$ holds.

After random initialization with probability at least 1/2 at least $\lfloor n/2 \rfloor$ of the bits in the inital string differ from x^*. Obviously, each of these bits has to be mutated at least once in order to find the global optimum x^*.

First, we assume that $k < n$ holds. We consider the situation after $(n-k)\ln n$ complete rounds, so each (1+1) EA was active $(l-1)\ln n$ times. The probability that a specific bit is not mutated $(l-1)\ln n$ times equals $(1-1/l)^{(l-1)\ln n}$. Thus, the probability that this bit is mutated at least once in that time equals $1-(1-1/l)^{(l-1)\ln n}$. The probability that $\lfloor n/2 \rfloor$ bits all do so equals $\left(1-(1-1/l)^{(l-1)\ln n}\right)^{\lfloor n/2 \rfloor}$. So, finally, the probability that among $\lfloor n/2 \rfloor$ bits which need to be mutated at least once there is at least one which is not within $(n-k)\ln n$ complete rounds equals $1-\left(1-(1-1/l)^{(l-1)\ln n}\right)^{\lfloor n/2 \rfloor}$. We see that we have

$$\mathrm{E}(T) \geq \frac{1}{2} \cdot \left(1 - \left(1 - \left(1 - \frac{1}{l}\right)^{(l-1)\ln n}\right)^{\lfloor n/2 \rfloor}\right) \cdot (n-k)\ln n$$

for $k < n$. For each l we have $(1-1/l)^{l-1} \geq 1/e$. Thus, $1-(1-1/l)^{(l-1)\ln n} \leq 1-1/n$ holds. This yields

$$\begin{aligned} \mathrm{E}(T) &\geq \frac{1}{2} \cdot \left(1 - \left(1 - \frac{1}{n}\right)^{\lfloor n/2 \rfloor}\right) \cdot (n-k)\ln n \\ &\geq \frac{1}{2} \cdot \left(1 - e^{-1/2}\right) \cdot (n-k)\ln n = \Omega(n \log n) \end{aligned}$$

for $k < n$.

For $k = n$ we have n (1+1) EAs each operating on exactly one bit. Each bit has an unique optimal value. We are waiting for the first point of time when each bit had this optimal value at least once. This is equivalent to throwing n coins independently and repeating this until each coin came up head at least once. Obviously, on average the number of coins that never came up head is decomposed by the factor of $p_f(t)\ln l$ each round. It follows that on average $\Theta(\log n)$ rounds are needed. This implies $\mathrm{E}(T) = \Omega(n\log n)$ in this case. □

Theorem 8 may be surprising: regardless of the way we choose the decomposition, the decomposition will always match the separability of the linear function. And yet, regardless of the decomposition, the CC (1+1) EA has no advantage over the traditional (1+1) EA working on the complete problem. Before we discuss how this can be explained, we try to get a complete picture of the performance of the CC (1+1) EA on linear functions. In order to derive an upper bound, we describe an upper bound technique that depends on the probability that the (1+1) EA takes longer than expected to optimize a linear function.

Lemma 9. *Let $p_f(t)$ denote the probability that the (1+1) EA does not optimize the linear function $f\colon \{0,1\}^n \to \mathbb{R}$ within $t \cdot n\ln n$ generations. The expected optimization time of the CC (1+1) EA on f with k (1+1) EAs operating on n/k bits each is*

$$E(T) = O\left(\frac{t\ln k}{-\ln p_f(t)} \cdot n\ln l\right).$$

Proof. We consider periods of time that consist of $tl\ln l$ rounds each. Each (1+1) EA is active $tl\ln l$ times in each such period. The probability that it finds an optimal bit string equals $1-p_f(t)$. Obviously, if an optimal bit string is found by a (1+1) EA, it cannot be lost again. Thus, after one such period the expected number of (1+1) EAs that are not yet optimal is decreased by the factor $p_f(t)$. Using standard arguments we see that the expected number of rounds that is needed is bounded above by $O(-\ln k/\ln p_f(t))$. □

The proof of Theorem 7 holds for arbitrary initial bit strings. Since the expected optimization time is $O(n\log n)$, there is a constant c such that it is less than $cn\ln n$. Markov's inequality yields that after $2cn\ln n$ generations the probability not to have optimized f is bounded above by 1/2. Thus, we have $p_f(2ct) \leq 2^{-t}$ for any linear function f and any $t \in \mathbb{N}$. This implies the following result.

Theorem 10. *The expected optimization time of the CC (1+1) EA on a linear function $f\colon \{0,1\}^n \to \mathbb{R}$ is $O(n\log^2 n)$.*

OneMax is a special linear function and we can give a better bound on $p_{\text{OneMax}}(t)$. The Hamming distance to the global optimum can never increase and the probability to increase the function value from i to $i+1$ is bounded below by $(n-i)/(en)$. Therefore, we can apply results from the coupon collector's problem (Motwani and Raghavan 1995). This yields $p_{\text{OneMax}}(2et) \leq n^{-t}$ for any $t \in \mathbb{N}$ and we can conclude the following result.

Theorem 11. *The expected optimization time of the CC (1+1) EA on* OneMax: $\{0,1\}^n \to \mathbb{R}$ *is* $O(n \log n)$.

We see that for OneMax the expected optimization time of the CC (1+1) EA is $\Theta(n \log n)$. Although the cooperative coevolutionary approach is not better than the traditional approach, it is at least not doing worse. For general linear functions, we cannot prove that the CC (1+1) EA is not doing worse; however, we conjecture that an upper bound of $O(n \log n)$ can be proved for all linear functions.

Conjecture 12. *The expected optimization time of the CC (1+1) EA on a linear function* $f\colon \{0,1\}^n \to \mathbb{R}$ *is* $O(n \log n)$.

3.2 Exploring the explorative advantage

We found out that in spite of the complete separability of linear functions, the CC (1+1) EA has no advantage over the (1+1) EA. Linear functions can be optimized by mutations of single bits alone. And since restricting the (1+1) EA to mutations of single bits does not slow it down on linear functions, we may speculate that linear functions are optimized using mainly mutations of single bits. One important aspect that we already pointed out in the introduction is the increased mutation probability used by the (1+1) EAs that are part of the CC (1+1) EA. What we expect from an increased mutation probability is, of course, an increased number of mutations. We investigate this a little further and classify a mutation by the number of mutated bits. We compare one round of the CC (1+1) EA, when each (1+1) EA was active once, with k generations of the traditional (1+1) EA. We concentrate on the first component, only. It is quite easy to calculate the expected number of mutations of exactly b bits. For a single (1+1) EA operating on l bits, this number equals $\binom{l}{b}(1/l)^b(1-1/l)^{n-b}$. Thus, for the (1+1) EA operating on n bits for k generations, this number equals

$k \cdot \binom{l}{b}(1/n)^b(1-1/n)^{n-b}$. For the CC (1+1) EA we get $\binom{l}{b}(1/l)^b(1-1/l)^{l-b}$. We consider the quotient of the two and get

$$\frac{\binom{l}{b}\left(\frac{1}{l}\right)^b\left(1-\frac{1}{l}\right)^{l-b}}{k\binom{l}{b}\left(\frac{1}{n}\right)^b\left(1-\frac{1}{n}\right)^{n-b}} = k^{b-1} \cdot \left(\frac{n-k}{n-l}\right)^{l-b} = \Theta\left(k^{b-1}\right).$$

With respect to linear functions it is interesting to note that for $b = 1$ the quotient does not grow. In spite of the increased mutation probability, the expected number of single bit mutations is not significantly larger for the CC (1+1) EA. This changes with mutations of at least two bits. The expected number grows exponentially in $(b-1)\ln k$. For small values of $b > 1$ even mutations of b specific bits occur in a polynomial number of generations. Hence, we can expect to see significant differences in the performance of the traditional (1+1) EA and its cooperative coevolutionary counterpart when such mutations play an important role. This motivates the definition of a family of functions where such mutations are crucial.

We start with a function which is exactly $(1, n)$-separable. The definition is inspired by the well-known LeadingOnes problem.

Definition 13. *For $n \in \mathbb{N}$ and $b \in \{1, \ldots, n\}$ with $n/b \in \mathbb{N}$, we define the function $\mathrm{LOB}_b \colon \{0,1\}^n \to \mathbb{R}$ (short for LeadingOnesBlocks) by*

$$\mathrm{LOB}_b(x) := \sum_{i=1}^{n/b} \prod_{j=1}^{b \cdot i} x_j$$

for each $x = x_1 \cdots x_n \in \{0,1\}^n$.

The function LOB_b is identical to the so-called Royal Staircase function which was introduced by van Nimwegen and Crutchfield (2001) in a different context. The function value equals the number of blocks of size b that have all bits set to 1 (scanning x from left to right). Nevertheless, it is not clear that mutations of b bits in one step are needed in order to optimize LOB_b. Since the initial bit string is chosen uniformly at random, it may be the case that on average $b/2$-bit mutations are sufficient. In order to overcome technical difficulties that arise from such uncertainties, we embed LOB_b in another function. In this function we give leading ones blocks a higher weight and subtract OneMax in order to force all the other bits to be set to 0. Finally, we use a well-known technique to achieve a controllable degree of separability: we define the function $\mathrm{CLOB}_{b,k}$ as k concatenated copies of this function.

Definition 14. *For* $n \in \mathbb{N}$, $k \in \{1, \ldots, k\}$ *with* $n/k \in \mathbb{N}$, *and* $b \in \{1, \ldots, n/k\}$ *with* $n/(bk) \in \mathbb{N}$, *we define the function* $\mathrm{CLOB}_{b,k} \colon \{0,1\}^n \to \mathbb{R}$ *by*

$$\mathrm{CLOB}_{b,k}(x) := \left(\sum_{h=1}^{k} n \cdot \mathrm{LOB}_b \left(x_{(h-1)l+1} \cdots x_{hl} \right) \right) - \textsc{OneMax}(x)$$

for all $x = x_1 \cdots x_n \in \{0,1\}^n$, *with* $l := n/k$.

Since LOB_b is exactly $(1, n)$-separable, it is obvious that $\mathrm{CLOB}_{b,k}$ is exactly (k, l)-separable with $l = n/k$. Since we are interested in finding out whether the increased mutation probability of the CC (1+1) EA proves to be beneficial, we concentrate on $\mathrm{CLOB}_{b,k}$ with $b > 1$ and use a decomposition that exactly matches the separability of the problem. We start our investigations with an upper bound on the expected optimization time of the CC (1+1) EA.

Theorem 15. *The expected optimization time of the CC (1+1) EA on the function* $\mathrm{CLOB}_{b,k} \colon \{0,1\}^n \to \mathbb{R}$ *is* $\Theta\left(kl^b \left(\frac{l}{b} + \ln k\right)\right)$ *with* $l := n/k$, *if the CC (1+1) EA exactly matches the function's separability with k (1+1) EAs, and* $2 \leq b \leq n/k$, $1 \leq k \leq n/4$, *and* $n/(bk) \in \mathbb{N}$ *hold.*

Proof. Since we have $n/(bk) \in \mathbb{N}$ we have k components $x^{(1)}, \ldots, x^{(k)}$ of length $l := n/k$ each. In each component the size of the blocks rewarded by $\mathrm{CLOB}_{b,k}$ equals b and there are exactly $l/b \in \mathbb{N}$ such blocks in each component.

We begin with the upper bound and consider the first (1+1) EA operating on $x^{(1)}$. As long as $x^{(1)}$ differs from 1^l, there is always a mutation of at most b specific bits that increases the number of leading ones blocks by at least one. After at most l/b such mutations $x^{(1)} = 1^l$ holds. The probability of such a mutation is bounded below by $(1/l)^b(1 - 1/l)^{l-b} \geq 1/(el^b)$. We consider $k \cdot 10e \cdot l^b((l/b) + \ln k)$ generations. The first (1+1) EA is active in $10e \cdot l^b((l/b) + \ln k)$ generations. The expected number of such mutations is bounded below by $10((l/b) + \ln k)$. Chernoff bounds (Motwani and Raghavan 1995) yield that the probability not to have at least $(l/b) + \ln k$ such mutations is bounded above by $e^{-4((l/b)+\ln k)} \leq \min\{e^{-4}, k^{-4}\}$. In the case $k = 1$ this immediately implies the claimed upper bound on the expected optimization time. Otherwise, the probability that there is a component different from 1^l is bounded above by $k \cdot (1/k^4) = 1/k^3$. This again implies the claimed upper bound.

The proof of the lower bound consists of two parts. First, we prove that, with probability close to 1, there are $\Omega(k)$ sub-populations where $x^{(i)} = 0^l$ holds after $o\left(k \cdot l^2\right)$ generations. Second, we prove that for these sub-populations $x^{(i)} \neq 1^l$ holds with a probability that is sufficiently large for the prove of the lower bound on the expected optimization time.

Each component equals 0^l with probability 2^{-l} after random initialization. If $l = O(1)$, this yields that each sub-population is initialized 0^l with probability $\Omega(1)$. Now we deal with the case where l grows with n. We consider the first component. With probability $1/4$ the first two bits have value 0 after random initialization. The probability that $x^{(1)}$ does not have the form $1^{jb}0^{l-jb}$ for some $j \in \mathbb{N}_0$ after $t \cdot k \cdot 2el \log l$ generations is bounded above by 2^{-t}. The probability that the first two bits are mutated in the first $t \cdot k \cdot 2el \log l$ generations is bounded above by $(t \cdot 2el \log l)/l^2$. Thus, we have $x^{(1)} = 0^l$ after $k \cdot 10el \log l$ generations with probability at least $1-((3/4) + 2^{-5} + (10e \log l)/l) = \Omega(1)$. Application of Chernoff bounds yield in both cases that with probability close to 1 the number of sub-populations equal to 0^l is $\Omega(k)$ after $O(n \log l)$ generations.

For the second part of the proof, we distinguish two cases. First, assume $\ln k \geq l/b$ holds. We consider a sub-population with $x^{(i)} = 0^l$. We consider the first $k(l^b - 1) \ln k$ generations after this is the case. Note that $l \geq 2$ implies $k(l^b - 1) \ln k = \Omega(kl^b \ln k)$. Obviously, if there is no mutation of the first b bits in $x^{(i)}$ in these generations, $x^{(i)} \neq 1^l$ still holds afterwards. The sub-population is active for $(l^b - 1) \ln k$ generations during the considered time interval. Therefore, the probability for such an event is bounded below by

$$\left(1 - \frac{1}{l^b}\right)^{(l^b-1)\ln k} \geq e^{-\ln k} = \frac{1}{k}.$$

We know from the first part of the proof that there are at least ck such populations for some positive constant $c < 1$. They all are independent due to the definition of the CC (1+1) EA. Therefore, the probability that at least one sub-population has $x^{(i)} \neq 1^l$ is bounded below by

$$\left(1 - \frac{1}{k}\right)^{ck} > \frac{e^{-c}}{2}$$

which is a positive constant. This yields $\Omega\left(kl^b \ln k\right)$ as lower bound on the expected optimization time.

Now we deal with the case $\ln k < l/b$. Again, we consider a sub-population with $x^{(i)} = 0^l$. We consider the first $(kl^b \cdot l/b)/2$ generations after this is the case. In order to increase the number of leading ones blocks by i, a mutation of i specific bits is necessary and sufficient. Such a mutation occurs with probability l^{-ib}. Since i/l^{ib} is maximal for $i = 1$, it suffices to consider mutations of b bits. The sub-population is active for $(l^b \cdot l/b)/2$ generations. By Chernoff bounds, the probability to have at least l/b such mutations is bounded above by $(e/4)^{l/(2b)} < \sqrt{3}/2$. Thus, with probability at least $1 - \sqrt{3}/2$ the global optimum is not found after $\Omega\left(kl^b(l/b)\right)$ generations. $\square$

The expected optimization time $\Theta(kl^b((l/b) + \ln k))$ grows exponentially with b, as could be expected. Note, however, that the basis is l, the length of each piece. This supports our intuition that the exploitation of the separability together with the increased mutation probability help the CC (1+1) EA to be more efficient on $\mathrm{CLOB}_{b,k}$. We now prove this belief to be correct by analyzing the expected optimization time of the (1+1) EA.

Theorem 16. *The expected optimization time of the (1+1) EA on the function* $\mathrm{CLOB}_{b,k}\colon \{0,1\}^n \to \mathbb{R}$ *is* $\Theta\left(n^b\left(n/(bk) + \ln k\right)\right)$, *if* $2 \le b \le n/k$, $1 \le k \le n/4$, *and* $n/(bk) \in \mathbb{N}$ *hold.*

Proof. We begin with a proof of the lower bound. This proof consists of two main steps. First, we prove that with probability at least 1/8 the (1+1) EA needs to make at least $\lceil k/8 \rceil \cdot l/b$ mutations of b specific bits to find the optimum of $\mathrm{CLOB}_{b,k}$. Second, we estimate the expected waiting time for this number of mutations.

Consider some bit string $x \in \{0,1\}^n$. It is divided into k pieces of length $l = n/k$ each. Each piece contains l/b blocks of length b. Since each leading block that contains 1-bits only contributes $n - b$ to the function value, these 1-blocks are most important.

Consider one mutation generating an offspring y. Of course, y is divided into pieces and blocks in the same way as x. But the bit values may be different. We distinguish three different types of mutation steps that create y from x. Note that our classification is complete, i. e., no other mutations are possible.

First, the number of leading 1-blocks may be smaller in y than in x. We can ignore such mutations since we have $\mathrm{CLOB}_{b,k}(y) < \mathrm{CLOB}_{b,k}(x)$ in this case. Then y will not replace its parent x.

Second, the number of leading 1-blocks may be the same in x and y. Again, mutations with $\mathrm{CLOB}_{b,k}(y) < \mathrm{CLOB}_{b,k}(x)$ can be ignored. Thus, we are only concerned with the case $\mathrm{CLOB}_{b,k}(y) \geq \mathrm{CLOB}_{b,k}(x)$. Since the number of leading 1-blocks is the same in x and y, the number of 0-bits cannot be smaller in y compared to x. This is due to the $-$OneMax part in $\mathrm{CLOB}_{b,k}$.

Third, the number of 1-blocks may be larger in y than in x. For blocks with at least two 0-bits in x the probability to become a 1-block in y is bounded above by $1/n^2$. We know that the $-$OneMax part of $\mathrm{CLOB}_{b,k}$ leads the (1+1) EA to all zero blocks in $O(n \log n)$ steps. Thus, with probability $O((\log n)/n)$ such steps do not occur before we have a string of the form

$$1^{j_1 \cdot b} 0^{((l/b)-j_1) \cdot b} 1^{j_2 \cdot b} 0^{((l/b)-j_2) \cdot b} \cdots 1^{j_k \cdot b} 0^{((l/b)-j_k) \cdot b}$$

as current string of the (1+1) EA.

The probability that we have at least two 0-bits in the first block of a specific piece after random initialization is bounded below by 1/4. It is easy to see that with probability at least 1/4 we have at least $\lceil k/8 \rceil$ such pieces after random initialization. This implies that with probability at least 1/8 we have at least $\lceil k/8 \rceil$ pieces which are of the form 0^l after $O(n \log n)$ generations. This completes the first part of the lower bound proof.

Each 0-block can only become a 1-block by a specific mutation of b bits all flipping in one step. Furthermore, only the leftmost 0-block in each piece is available for such a mutation leading to an offspring y that replaces its parent x. Let i be the number of 0-blocks in x. For $i \leq k$, there are up to i blocks available for such mutations. Thus, the probability for such a mutation is bounded above by i/n^b in this case. For $i > k$, there cannot be more than k 0-blocks available for such mutations, since we have at most one leftmost 0-block in each of the k pieces. Thus, for $i > k$, the probability for such a mutation is bounded above by k/n^b. This yields

$$\frac{1}{8} \cdot \left(\sum_{i=1}^{k} \frac{n^b}{i} + \sum_{i=k+1}^{\lceil k/8 \rceil l/b} \frac{n^b}{k} \right) \geq \frac{n^b}{8} \cdot \left(\ln k + \frac{kl}{8bk} \right) = \Omega\left(n^b \cdot \left(\frac{n}{bk} + \log n \right) \right)$$

as lower bound on the expected optimization.

For an upper bound, we apply a method similar to f-based partitions (Droste, Jansen, and Wegener 2002). We have k pieces $x^{(1)}, \ldots, x^{(k)}$ of length

$l := n/k$ each. In each piece, there are b bits in each of the blocks rewarded by $\text{CLOB}_{b,k}$ and there are exactly $l/b \in \mathbb{N}$ such blocks in each piece.

For $i \in \{1, \ldots, k\}$, $x \in \{0,1\}^n$ we define the following. The set of first-bit positions in every block of the i^{th} piece is given by

$$B_i := \left\{ j \cdot b + 1 | j \in \left\{ (i-1)\frac{l}{b}, (i-1)\frac{l}{b} + 1, \ldots, i\frac{l}{b} - 1 \right\} \right\}.$$

The set of bit positions in the j^{th} block of the i^{th} piece which are 0 is given by

$$Z_{i,j}(x) := \{h | j \leq h \leq j + l - 1 \wedge x_j = 0\}$$

for $j \in B_i$. The first-bit position of the left-most block in the i^{th} piece which is not the all one string is given by

$$z_i := \min \{\{j \in B_i | Z_{i,j}(x) \neq \} \cup \{\max B_i\}\}.$$

Note that z_i indicates the first-bit position of the right-most block of the piece if the entire piece is the all one string. Finally, we define the event $A(x)$, an *advancing step*, to be the situation in which there exists exactly one $i \in \{1, \ldots, k\}$ such that all bits in $Z_{i,z_i(x)}(x)$ mutate and all other bits in x do not mutate.

First observe that $\forall x : Pr\{A(x)\} \geq \binom{k}{1}\frac{1}{n^b}\left(1 - \frac{1}{n}\right)^{n-b} \geq \frac{k}{en^b}$ and that each event $A(x)$ belongs to exactly 1 piece. Observing that there are l/b blocks in each piece, for symmetry reasons it is sufficient to see that the global optimum is reached after at most l/b events in all pieces.

Since the $A(x)$ events are independent and we are interested only in bounding the waiting time to obtain a specific number of such events in each piece, we can consider this process to be equivalent to a generalized form of the coupon collector's problem where we wait for the first point of time with at least l/b balls in each bin. Here the pieces are bins and the advancing step events are balls. Applying Lemma 17 we see that $O(k \ln k + k\frac{l}{b})$ steps are on average sufficient to insure that each piece has experienced at least l/b of the $A(x)$ events. Since we expect to wait $O\left(\frac{n^b}{k}\right)$ generations for each advancing step, the total expected waiting time until the global optimum is reached is $O(n^b \ln k)$. □

Lemma 17. *Consider balls which are thrown independently and uniformly at random into k bins. Let M denote the minimal number of balls thrown such that each bin contains at least h balls.*

$$E(M) = O(k \ln k + kh)$$

Proof. For $k = 1$ the process is deterministic and the statement obviously true. Thus, we assume $k > 1$. We begin with a lower bound on M. If we have at least h balls in each bin, the sum of all balls is at least kh. Thus $M \geq kh$ holds. Now, consider the situation after $(k-1)\ln k$ balls. The probability that the first bin is empty equals $(1 - 1/k)^{(k-1)\ln k} > e^{-\ln k} = 1/k$. The probability that there is an empty bin among the k bins is bounded below by $1 - (1 - 1/k)^k \geq 1 - e^{-1}$. Thus, we have $M \geq (1 - 1/e) \cdot (k-1)\ln n$. Together, we have $M = \Omega(kh + k\ln k)$.

For the upper bound, consider the situation after $4kh + 4k\ln k$ balls. Let B_1 denote the number of balls in the first bin. We have $\mathrm{E}(B_1) = 4h + 4\ln k$. Using Chernoff bounds we get

$$\begin{aligned}
\mathrm{Prob}(B_1 < h) &= \mathrm{Prob}\left(B_1 < \left(1 - \frac{3h + 4\ln k}{4h + 4\ln k}\right)(4h + 4\ln k)\right) \\
&< e^{-(4h+4\ln k)((3h+4\ln k)/(4h+4\ln k))^2/2} < e^{-(1/2)\cdot(3/4)\cdot 4\ln k} \\
&= k^{-3/2}.
\end{aligned}$$

Thus, with probability at most $k \cdot k^{-3/2} = 1/\sqrt{k}$ there is a bin with less than h balls after $4kh + 4k\ln k$ balls. This yields $M \leq \left(1 - 1/\sqrt{k}\right)^{-1} \cdot (4kh + 4k\ln k) = O(kh + k\ln k)$. ☐

We want to see the benefits the increased mutation probability due to the cooperative coevolutionary approach can cause. Thus, our interest is not specifically concentrated on the concrete expected optimization times of the (1+1) EA and the CC (1+1) EA on $\mathrm{CLOB}_{b,k}$. Rather, we are much more interested in a comparison. When comparing (expected) run times of two algorithms solving the same problem, it is most often sensible to consider the ratio of the two (expected) run times. Therefore, we consider the expected optimization time of the (1+1) EA divided by the expected optimization time of the CC (1+1) EA, both on $\mathrm{CLOB}_{b,k}$. We see that

$$\frac{\Theta\left(n^b \cdot \left(\frac{n}{bk} + \ln k\right)\right)}{\Theta\left(kl^b\left(\frac{l}{b} + \ln k\right)\right)} = \Theta\left(k^{b-1}\right)$$

holds. We can say that the CC (1+1) EA has an advantage of order k^{b-1}. The parameter b is a parameter of the problem. In our special setting, this holds for k, too, since we divide the problem as much as possible. Using c components, where $c \leq k$, would reveal that this parameter c influences the advantage of the CC (1+1) EA in a way k does in the expression above. Obviously, c is a parameter of the algorithm. Choosing c as large as the objective function $\mathrm{CLOB}_{b,k}$ allows yields the best result. This confirms our intuition that the separability of the problem should be exploited as much as possible. We see that for some values of k and b this can decrease the expected optimization time from super-polynomial for the (1+1) EA to polynomial for the CC (1+1) EA. This is, for example, the case for $k = n(\log\log n)/(2\log n)$ and $b = (\log n)/\log\log n$.

It should be clear that simply increasing the mutation probability in the (1+1) EA will not resolve the difference. Increased mutation probabilities lead to a larger number of steps where the offspring y does not replace its parents x since the number of leading ones blocks is decreased due to mutations. After some time it will be necessary not to mutate at least half of the bits in order to further increase the function value. Using the same mutation probability for the (1+1) EA as for each sub-population of the CC (1+1) EA, i.e. $1/l$, the waiting time for only one such mutation is on average $\left((1-1/l)^{n/2}\right)^{-1} \approx e^{k/2}$ generations. As a result, the CC (1+1) EA gains clear advantage over the (1+1) EA on this $\mathrm{CLOB}_{b,k}$ class of functions. Moreover, this advantage is drawn from more than a simple partitioning of the problem. The advantage stems from the coevolutionary algorithm's ability to increase the focus of attention of the mutation operator, while using the partitioning mechanism to protect the remaining components from the increased disruption.

3.3 Separability considered harmful

We have seen that the CC (1+1) EA can achieve a tremendous speed-up compared to the (1+1) EA on separable functions when its increased explorative possibilities are helpful. However, separability by itself is not sufficient to make the cooperative coevolutionary approach beneficial, as the consideration of linear functions revealed. Now we want to find out whether there can be disadvantages due to this approach even on separable functions.

We should be careful about what we want to call a disadvantage. First of all, we assume that the considered objective function is separable, and that the decomposition of the problem for the CC (1+1) EA matches this separability. Since we are, as usual, interested in extreme cases we assume that the decomposition matches exactly the separability of the problem. We know that the different mutation probabilities employed by the CC (1+1) EA compared to the (1+1) EA can cause huge performance differences. Moreover, it is known that the most common mutation probability $1/l$ for strings of length l is not optimal for all functions (Jansen and Wegener 2000). Different mutation probabilities can mean the difference between polynomial and super-polynomial expected optimization time. We saw that the superior performance of the CC (1+1) EA on $\mathrm{CLOB}_{b,k}$ was not due to the increased mutation probability alone. Thus, here we are not satisfied with an example where the (1+1) EA is superior due to the use of a more appropriate mutation probability.

Our goal is to investigate the following. Is it possible that the CC (1+1) EA is outperformed by the (1+1) EA on a separable problem, where the CC (1+1) EA makes full use of this separability by using a decomposition which exactly matches the problem's separability and where for each component the CC (1+1) EA uses an optimal mutation probability? In order to answer this question we define an example problem where this is exactly the case. However, we can show that if the (1+1) EA has polynomial expected optimization time, then the CC (1+1) EA optimizes this problem within a polynomial number of steps with a probability that quickly converges to 1.

Assume the objective function $f\colon \{0,1\}^n \to \mathbb{R}$ is (r,s)-separable and the expected optimization time of the (1+1) EA on f is $\mathrm{E}(T) = t(n)$. Since f is (r,s)-separable, there exist index sets $I_1, \dots I_r$ and functions $g_1, \dots, g_r$ such that

$$f(x_1 \dots x_n) = g_1\left(x_{i_{1,1}} \cdots x_{i_{1,|I_1|}}\right) + \cdots + g_r\left(x_{i_{r,1}} \cdots x_{i_{r,|I_r|}}\right)$$

for all $x_1 \cdots x_n \in \{0,1\}^n$. We consider functions $f_1, \dots, f_r$ which we define by

$$f_j(x_1 \dots x_n) = g_j\left(x_{i_{j,1}} \cdots x_{i_{j,\left|I_j\right|}}\right)$$

for all $j \in \{1, \dots, r\}$. Obviously, $\mathrm{E}(T_j) \leq t(n)$ holds for the expected optimization time $\mathrm{E}(T_j)$ of the (1+1) EA on f_j for each $j \in \{1, \dots, r\}$. We divide a bit string into components exactly matching the separability of f but use $1/n$ as mutation probability for each component. We consider $r \cdot n^2 \cdot t(n) = O(n^3 t(n))$ generations of the CC (1+1) EA. Note, that since $t(n)$ is polynomial $O(n^3 t(n))$ is polynomially bounded, too. For each component we apply Markov's inequality and see that with probability at most $1/n^2$ this component is not optimized. Thus, with probability at most $r/n^2 \leq 1/n$

there is a component that is not optimized. We conclude that, with probability $1 - O(1/n)$, the CC (1+1) EA optimizes f within $O(n^3 t(n))$ generations.

Definition 18. *For $l \in \mathbb{N}$, we define the function $g_l \colon \{0,1\}^l \to \mathbb{R}$ by*

$$g_l(x) := \begin{cases} l+i & \text{if } x = 1^i 0^{l-i} \text{ with } i \in \{0,1,2,\ldots,l\} \\ l+i & \text{if } x = 0^{l-i} 1^i \text{ with } i \in \{4\} \cup \{6,9,\ldots,3\lfloor l/3 \rfloor\} \\ l - \textsc{OneMax}(x) & \text{otherwise} \end{cases}$$

for each $x = x_1 \cdots x_l \in \{0,1\}^l$. For $l \in \mathbb{N}$ and $k \in \mathbb{N}$ we define $n := l \cdot k$ and the function $f_{k,l} \colon \{0,1\}^n \to \mathbb{R}$ by

$$f_{k,l}(x) := \sum_{i=1}^{k} g_l\left(x_{(i-1)l+1} \cdots x_{il}\right)$$

for each $x = x_1 \cdots x_n \in \{0,1\}^n$.

It is easy to see g_l is exactly $(1,l)$-separable. We conclude that $f_{k,l}$ is exactly (k,l)-separable. We claim that the (1+1) EA outperforms the CC (1+1) EA on $f_{k,l}$ when performance is measured by expected optimization time and the CC (1+1) EA makes full use of the separability of $f_{k,l}$, even if the CC (1+1) EA uses optimal mutation probabilities for each component. We begin the formal analysis with an upper bound for the (1+1) EA using standard mutation probability $1/n$.

Theorem 19. *The expected optimization time of the (1+1) EA on the function $f_{k,l} \colon \{0,1\}^n \to \mathbb{R}$ is $O(n \cdot l)$ for each $k, l \in \mathbb{N}$ with $l = \Omega(\log n)$ and $n = k \cdot l$.*

Proof. First, we prove that the expected optimization time of the (1+1) EA with mutation probability $1/n$ on g_n is $O(n^2)$. We consider a run of the (1+1) EA on g_n. Let P denote the set of points $\{0^{n-i}1^i \mid i \in \{4\} \cup \{6,\ldots,3\lfloor n/3 \rfloor\}$. Let F denote the event that we have $x \in P$ for the current string x at some point of time during the run before reaching the global optimum 1^n. Let $\overline{F}$ denote the event that this is not the case. Using the method of f-based partitions (Droste, Jansen, and Wegener 2002) we see that for the expected optimization time of the (1+1) EA on g_n $\mathrm{E}\left(T \mid \overline{F}\right) = O(n^2)$ and $\mathrm{E}\left(T \mid F\right) = O(n^4)$ hold. We have

$$\mathrm{E}\left(T\right) \leq \mathrm{E}\left(T \mid \overline{F}\right) + \mathrm{Prob}\left(F\right) \cdot \mathrm{E}\left(T \mid F\right).$$

Therefore, it is sufficient to prove $\text{Prob}(F) = O(1/n^2)$.

After random initialization, we have $x \notin P$ with probability exponentially close to 1. We consider a run of the (1+1) EA until we have $x \in P$ or $x \in \{1^i 0^{n-i} \mid i \in \{0, 1, \ldots, n\}\}$ for the first time. The function value is then given by $n - \text{OneMax}(x)$. Consider $B_i := \{x \in \{0,1\}^n \mid \text{OneMax}(x) = i\}$. For symmetry reasons, each point $y \in B_i$ has equal probability to become current search point x of the (1+1) EA. We conclude that we have $x \in P$ at the end of this phase with probability $O(1/n^3)$. Let x denote the current string of the (1+1) EA and let $x = 1^i 0^{n-i}$ hold for some $i \in \{0, 1, \ldots, n-1\}$. Let x' denote the first offspring with $g_n(x') > g_n(x)$. The Hamming distance between x and the closest point in P equals $i+3$ for $i > 0$ and 4 otherwise. The second closest point in P has Hamming distance $i+6$. The next point with larger g_n-value has Hamming distance 1. Thus, the probability to have $x' \in P$ is bounded above by

$$\frac{(1/n)^{i+3} + n \cdot (1/n)^{i+6}}{1/(en)} = O\left(\frac{1}{n^{i+2}}\right)$$

for $i > 0$ and $O(1/n^3)$ otherwise. This implies $\text{Prob}(F) = O(1/n^3) + \sum_{i=1}^{n-1} O(1/n^{i+2}) = O(1/n^3)$.

Obviously, on $f_{k,l}$ the expected time until the first of the k g_l-pieces becomes all 1 is bounded above by $O(n \cdot l)$. Furthermore, we see that, with probability $1 - O(1/n^3)$, the first piece becomes all 1 within $O(n \cdot l)$ generations. Finally, if we have that the (1+1) EA enters P in the first component, the expected number of generations before this piece becomes all 1 is bounded above by $O(n^3 \cdot l)$. All this holds for all k pieces. The probability that there is one piece that is different from 1^l after $O(n \cdot l)$ generations is bounded above by $O(k/n^3)$. Thus, the expected optimization time is bounded above by

$$O(n \cdot l) + O\left(\frac{k}{n^3} \cdot n^3 \cdot l\right) = O(n \cdot l).$$

□

Theorem 20. *For $l \in \mathbb{N}$, $k := l^4$, and $n := k \cdot l$, the expected optimization time of the CC (1+1) EA on $f_{k,l}\colon \{0,1\}^n \to \mathbb{R}$ with k components that match the function's separability is $\Omega(n \cdot l \cdot l^{1/3})$ regardless of the mutation probabilities used for the k components.*

Proof. We start our analysis with the CC (1+1) EA using standard mutation probability $1/l$ in each component. We know from the proof of Theorem 19 that for each component we have $x = 0^{n-4}1111$ with probability $\Theta(1/l^3)$. The probability to have this in at least 1 of the k independent components is $p(l,k) := 1 - (1 - \Theta(1/l^3))^k$. Since we have $k = l^4 \gg l^3$, we know that $p(l,k)$ converges to 1 exponentially fast as l grows. Then $\Theta(l)$ mutations of exactly 3 bits are needed to reach the global optimum. This implies $\mathrm{E}(T) = \Omega(k \cdot l \cdot l^3) = \Omega(n \cdot l^3) = \omega(n \cdot l \cdot l^{1/3})$ as lower bound on the expected optimization time.

We may use mutation probabilities different from $1/l$. Larger mutation probabilities can only increase the probability to reach $0^{n-4}1111$. It is easy to see that the probability for 3-bit mutations is maximal for the mutation probability $3/l$. But mutation probabilities $\Theta(1/l)$ do not lead to a smaller lower bound on the expected optimization time. Mutation probabilities $o(1/l)$ can decrease the probability to enter any point in P (see proof of Theorem 19 for a definition of P). But as long as $\Omega(1/l^{4/3})$ is a lower bound on the mutation probability, the probability to have $x = 0^{n-4}1111$ at some point of time is still bounded below by some positive constant. Now, we take into account that smaller mutation probabilities increase the expected time spent on the 1^i0^{n-i} path. For $l = O(1/l^{4/3})$ we already have $\Omega(k \cdot l \cdot l^{4/3}) = \Omega(n \cdot l \cdot l^{1/3})$ as lower bound on the expected optimization time. □

Using $k = l^4$ and $n = k \cdot l$ the expected optimization time of the (1+1) EA is $O(n^{6/5})$ whereas it is $\Omega(n^{19/15})$ for the CC (1+1) EA. Thus, the performance advantage of the (1+1) EA is a factor of $\Omega(n^{1/15})$. We can increase the relative performance advantage when using a more complicated example function.

The idea of g_l is that there are two paths leading to the global optimum. The expected time spent on the paths is very different, the “slower” path is significantly less likely. Since the probability depends on the mutation probability, the CC (1+1) EA is much more likely to use the “slow path” in at least one component. The relative advantage is not too impressive, since with increasing mutation probability that time spent on the slow path decreases. The reason for this is easy to see: we need 3-bit mutations in order to advance on the slow path. We know from Section 3.2 that the CC (1+1) EA is much faster in finding appropriate b-bit mutations than the (1+1) EA for all $b > 1$. If we can replace the slow path by another kind of slow path, we can increase the relative performance difference. We want a path where the time spent on the path is long but still polynomial and where the advance on the path is mainly caused by single-bit mutations. Generalized long paths (Horn, Goldberg, and Deb (1994), Rudolph (1997)) have this property. It is known that the (1+1) EA operating on $\sqrt{n-1}$-long paths defined on n bits takes $\Theta(\sqrt{n} \cdot 2\sqrt{n})$ steps on the path and that only mutations of at least $\sqrt{n}-1$ bits can decrease the time on the path significantly (Droste, Jansen, and Wegener 1998). Defining such a $\sqrt{c-1}$-long path on c bits with $c = \log^j n$, where $j \in \mathbb{N}$ is an arbitrarily large constant, yields a relative performance difference that can be arbitrarily large, yet polynomial. Since the construction is complicated, the analysis is tedious, and the results are purely asymptotic in the sense that they are unlikely to occur for reasonably small values of n, we do not discuss this further. Note, that the claimed relative performance ratio of $\Theta(n^c)$ for $c \in \mathbb{N}$ does not contradict our reasoning above: with probability close to 1 the relative performance ratio in a single run is bounded above by $O(n^3)$.

4 Coping with inseparability

Though the presence of separability may not be sufficient to guarantee that the CC (1+1) EA outperforms the more traditional (1+1) EA, it remains to be seen whether or not its absence may in contrast produce unique challenges for coevolution. Indeed, much empirical research has focused on the question of whether or not cross-population nonlinearities of a problem may require the special attention of a design engineer who chooses to use coevolution (Bull 1997; Wiegand, Liles, and De Jong 2001; Wiegand, Liles, and De Jong 2002a).

It isn't hard to imagine why researchers have focused on this issue. Applying a CCEA typically involves a static partitioning of the problem space by the algorithm's designer. This partitioned decomposition is exactly what allows the CC (1+1) EA to leverage its increased mutation for superior performance when the problem separation was commiserate with the algorithm's internal decomposition. When the decomposition is *not* related to the problem, or when the problem *cannot* be separated, coevolution may suffer as a result of an inappropriate decomposition.

As such, the exact nature and extent of the effects of inseparability on coevolution is an important research topic. Is it the case that the CC (1+1) EA will perform much worse than the (1+1) EA when the problem cannot, or is not, decomposed appropriately across the population boundaries? Alternatively, perhaps the property of inseparability makes little difference, and provides no real advantage to the (1+1) EA? Or, is it possible that the CC (1+1) EA can still gain advantage over the (1+1) EA, in spite of the presence of strong cross-population nonlinearities? In fact, all three of these can be true depending on the problem at hand, as we will show in the remainder of this section.

4.1 Difficulties due to inseparability

It is clearly the case that the property of inseparability may give a performance advantage to the (1+1) EA over its coevolutionary analog, though the degree of the effect may depend on the function being optimized. This being the case, perhaps the most interesting question might be: how severe can the difference become?

To try to get a clearer perspective, we look more closely at what the CC (1+1) EA is doing when it traverses the search space. In some sense, it is essentially a kind of line search, moving only along a particular projection of the space at any given step. Given this, intuition tells us that a coevolutionary algorithm as naive as our (1+1) mechanism may easily get "tricked" by a fitness landscape and become locked away from the global optimum in a way in which the traditional EA would not.

In fact, this is undoubtedly the case for some inseparable functions. Definition 21 below describes the very well-known, exactly $(1, n)$-separable TRAP function, which has precisely the effect we describe above: the CC (1+1) EA cannot reach the global optimum. Theorem 22 below, and its subsequent proof, not only justify our intuition, but also help to give our intuition a little more clarity.

Definition 21. *For $n \in \mathbb{N}$, we define* $\textsc{Trap}\colon \{0,1\}^n \to \mathbb{R}$ *by*

$$\textsc{Trap}(x) := \left(n \cdot \prod_{i=1}^{n} x_i \right) + n - \textsc{OneMax}(x)$$

for all $x = x_1 \cdots x_n \in \{0,1\}^n$.

Theorem 22. *Let* $\textsc{Trap}\colon \{0,1\}^n \to \mathbb{R}$, *be decomposed into $k \geq 4$ equal sized components of length $l \geq 2$, such that $n = kl$. The sequential CC (1+1) EA will fail to converge to the global optimum of* $\textsc{Trap}$ *with probability* $1-2^{-\Omega(n)}$.

Proof. We define the term *solved component* to mean a component that is the all one string, 1^l, and the term *unsolved component* to mean a component that contains at least one 0.

The proof consists of two basic parts. First, we prove that with a probability exponentially approaching 1, there are at least two subpopulations of the sequential CC (1+1) EA whose individual is an unsolved component. Next we show that, given there are at least two unsolved components after initialization, the mechanism of the algorithm itself will prevent it from accepting mutations that will lead it to the global optimum.

For the first part of the proof, we begin by observing that the probability that a given population's individual contains the all one string after initialization equals 2^{-l}, so the expected number of such individuals in k subpopulations equals $\frac{k}{2^l}$. Using Chernoff bounds we can see that the probability that at most half of the k subpopulations are solved after initialization is at most

$$
\begin{aligned}
& \left[\frac{e^{2^{l-1}-1}}{(2^{l-1})^{2^{l-1}}}\right]^{k/2^l} \\
= & \frac{1}{e^{k/2^l}}\left(\frac{e}{2^{l-1}}\right)^{k/2^l \cdot 2^{l-1}} \\
= & \frac{1}{e^{k/2^l}}\left(\frac{2e}{2^l}\right)^{k/2} \\
= & 2^{k/2-kl/2} \cdot e^{k/2-k/2^l} \\
= & 2^{-\Omega(kl)} = 2^{-\Omega(n)}
\end{aligned}
$$

So the probability that at least half of the k populations are unsolved is $1 - 2^{-\Omega(n)}$. Thus, given $k \geq 4$, there are at least two unsolved components with probability exponentially approaching 1.

For the second component of the proof, let us first suppose that all the subpopulations contained the all one string except for two of them, one that is the current active subpopulation under consideration by the algorithm, population a, and another that has yet to be considered during the current round, population b. Suppose that the necessary mutations are performed in $x^{(a)}$ such that the offspring, $y^{(a)}$ is the all one string. The offspring $y^{(a)}$ can be accepted if and only if $\textsc{Trap}(x^{(1)} \cdots y^{(a)} \cdots x^{(b)} \cdots x^{(k)}) \geq \textsc{Trap}(x^{(1)} \cdots x^{(a)} \cdots x^{(b)} \cdots x^{(k)})$, which cannot be true, since $x^{(b)}$ does not contain the all one string. Therefore the offspring will not be accepted. Since the parent, $x^{(a)}$ was not the all one string, the same event will be true symmetrically for $x^{(b)}$.

Given that $k \geq 4$, there cannot be fewer than two unsolved components. Having more than two unsolved components cannot resolve the problem, since a offspring that contains more ones than the parent will have a lower fitness than the parent if at least one other component is unsolved. □

Here the very partitioning mechanism of the CC (1+1) EA that helps gain advantage over the (1+1) EA in the previous section now works against it. In order to solve the Trap function, an algorithm must be able to make that final n-bit leap out of the trap, and this cannot be done in a single step by the CC (1+1) EA. Although exceedingly unlikely, it *can* be done for the (1+1) EA. Thus, while the expected waiting time for the (1+1) EA on Trap is exponential with respect to the size of the trap, it will eventually find the unique global optimum, whereas the CC (1+1) EA will almost certainly not do so. The difference in their performance, in terms of expected waiting time until the global optimum is reached, is infinite.

One may be tempted to believe that the coevolutionary algorithm did quite well: it found the second best point in the search space, and it did so just as quickly as the (1+1) EA (since the function is essentially OneMax when the global optimum is omitted). However, the gap between the point found and the global optimum can easily be quite large. Imagine a set of concatenated Trap functions, for instance, such that there are several populations working on each Trap. The global optimum still cannot be reached, and the best reachable point may be quite far from the optimum.

Additionally, one may be tempted to imagine that it was simply the *degree* of separability, since the Trap function is exactly $(1, n)$-separable. However, the same effect can be produced when the degree of separability matches the size of the individual components represented in the algorithm. Imagine a concatenation of Traps, each of length l, but a *decomposition* of the problem for the CC (1+1) EA that forces the Trap functions to be split across the population boundaries. Again, the global optimum cannot be reached (as long as l is sufficiently large), but the problem is no less separable in degree than was the $\mathrm{CLOB}_{b,k}$ problem discussed in the last section.

4.2 When inseparability is irrelevant

Our intuition is assuaged: the property of separability seems to have definite relevance to the performance of coevolution for some problems. In the case where we have problems that are separable along population boundaries, we are comforted by the knowledge that reaching the global optimum is possible; whereas, we have no such guarantee when problems are inseparable. Still this discovery is hardly surprising, and it says nothing about whether or not inseparability is a sufficient enough property to be considered a general foil to coevolutionary effectiveness. Indeed, as we will show in this subsection, it is not. The mere presence of inseparability tells us little about how easy or difficult the problem may be to solve, either for the (1+1) EA or the CC (1+1) EA.

Consider the well-known LEADINGONES problem, defined below. This function merely returns the number of leading, consecutive 1-bits from left to right in the bit string.

Definition 23. *The function* LEADINGONES: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\text{LEADINGONES}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

for all $x = x_1 \cdots x_n \in \{0,1\}^n$.

This problem is certainly exactly $(1, n)$-separable, so there exists no way to partition it into smaller, separable components. Yet, the expected waiting time for a (1+1) EA to find the global optimum can be tightly bound by $\Theta(n^2)$ (Droste, Jansen, and Wegener 2002). Further, as the theorem and proof below show, the CC (1+1) EA's expected optimization time is *also* $\Theta(n^2)$.

Theorem 24. *The expected optimization time for the CC (1+1) EA on the function* LEADINGONES *is* $\Theta(n^2)$.

Proof. The proof for the upper bound is very straight forward. We pessimistically assume that the algorithm solves its components from left to right, one at a time. We know from Droste, Jansen, and Wegener (2002) that the expected number of active steps needed for a given component of length l to reach the all string can be bounded above by $2el^2$. Since a component is active every k steps, it will require a given component at most $2el^2k$ steps to reach 1^l. Components are solved one at a time, so we can take the sum of the expectations to find the expected waiting time for the entire process, $\sum_{i=1}^{k} 2el^2k = O(n^2)$. For the lower bound, we begin by defining the term *progressive component* to mean the left-most component that is not the all one string. We note that all of the components to the right of the progressive component evolve without influencing the function value. Since the original string is drawn at random, and the mutation events are generated independently and uniformly at random, the result for each of these components to the right of the progressive component must contain random strings. Thus we can treat the process as successive solutions the individual leading ones problems for each component. The lower bound for a given component can be obtained from Droste, Jansen, and Wegener (2002), except that now we must show that no advantage can be obtained from the partition.

First, we prove the lower bound for the case $l > 3$. We define the term *head start bits* to be the number of bits that lead a component when that component first becomes the progressive component. Since the string is random until the component becomes progressive, the expected number of such bits can be bound above by $l/3$. Using Chernoff bounds, we can see that the probability that there are more than $\frac{2}{3}l$ head start bits is bounded above by $\frac{3}{4}^{l/3}$. Thus, from Droste, Jansen, and Wegener (2002) we know that the lower bound of the expected number of active steps is $\Omega(l - \frac{2}{3}l)^2) = \Omega(l^2/9)$. A step is active every k generations, there are $\Omega(kl^2/9)$ such generations. We can slow the algorithm down by assuming that after reaching 1^l in the progressive component, there are no more mutations in the current round. Since each component becomes all one only once, this slows down the optimization by less than k^2. Now, in each round at most one component can be solved which yields $\sum_{i=1}^{k} kl^2/9 = n^2/9$ as lower bound. Altogether this yields $n^2/9 - k^2 = \Omega(n^2)$ for $l > 3$.

For $l \leq 3$ it suffices to see that the expected number of leading ones gained in one round is constant. This implies that there are on average $\Omega(k) = \Omega(n)$ rounds before the optimum is reached. In each round there are $\Omega(n)$ function evaluations. □

Interestingly, it is impossible for a function to be less separable than this LeadingOnes problem, and yet this fact poses neither a general difficulty for solving the problem, nor a specific disadvantage to the coevolutionary algorithm. The algorithm's performance seems to be unaffected by the inseparability of the problem.

There are at least two reasons for this. First of all, this problem must be solved in a specific order in terms of the bit positions in the string, regardless of which algorithm it uses. As a result, the partitioning makes very little difference to the problem's solution, one way or the other. Second, there is nothing intrinsically contradictory in the interrelated bits. The LeadingOnes function cannot drive the algorithm to a point where its partitioning makes it impossible to make the necessary jump to the solution.

In fact, it is not hard to envision a general technique capable of demonstrating run time performance similar to OneMax for a variety of inseparable functions by simply aggregating OneMax to an inseparable function. For example, consider the NeedleOM_c function below. It is clear that both algorithms behave on this function as they would on OneMax, even with an arbitrarily small positive constant c. With this technique it is easy to see that there may be a large number of inseparable functions that remain relatively easy to solve by both the (1+1) EA and CC (1+1) EA.

Definition 25. *For any $c > 0$, the function* $\text{NeedleOM}_c\colon \{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\textit{NeedleOM}_c(x) := c \cdot \text{OneMax}(x) + n \cdot \prod_{i=1}^{n} x_i$$

for all $x = x_1 \cdots x_n \in \{0,1\}^n$.

Regardless, even though inseparability has a part to play in understanding how and why coevolutionary algorithms perform the way they do, it is clear there is something more than this at work here. Indeed, in the previous section we learned that separability alone is insufficient to guarantee superior performance of the CC (1+1) EA, and now we begin to see that *in*separability alone is insufficient to guarantee its inferior performance.

4.3 An exponential gap between CC (1+1) EA and (1+1) EA performance

We have seen that, though inseparability can prove to be a stumbling block to CC (1+1) EA success in terms of optimization, there are also some inseparable problems that are no more or less difficult for the CC (1+1) EA than for its more traditional analog. As it turns out, it is also true that the CC (1+1) EA can preserve its advantage over a (1+1) EA in spite of the existence of inseparability in the problem.

Here, we want to prove an exponential difference in performance between the CC (1+1) EA and the (1+1) EA. We do so using a general technique that delivers such results (Witt 2003). Assume that you have two randomized search heuristics A and A' and you want to demonstrate an exponential difference in performance of the two algorithms used as function optimizers. Assume that you know two functions $g\colon \{0,1\}^n \to \mathbb{R}$ and $g'\colon \{0,1\}^n \to \mathbb{R}$ with the following properties. Algorithm A is clearly faster on g than on g', whereas algorithm A' is clearly faster on g' than on g. Assume that both, g and g', have a unique global optimum. Let x_{opt} be the optimal solution for g and x'_{opt} be the optimal solution for g'. Now, define a function $f\colon \{0,1\}^{2n} \to \mathbb{R}$ with

$$f(x) = \begin{cases} g\left(x_1 \cdots x_n\right) + g'\left(x_{n+1} \cdots x_{2n}\right) & \begin{array}{l}\text{if } x_1 \cdots x_n \neq x_{\text{opt}} \text{ or} \\ \mathrm{H}\left(x'_{\text{opt}}, x_{n+1} \cdots x_{2n}\right) < \Delta\end{array} \\ 2\left(g\left(x_{\text{opt}}\right) + g'\left(x'_{\text{opt}}\right)\right) - \mathrm{H}\left(\overline{x'_{\text{opt}}}, x\right) & \text{otherwise} \end{cases}$$

where $\overline{x'_{\text{opt}}}$ denotes the bit-wise complement of x'_{opt} and $x = x_1 \cdots x_{2n} \in \{0,1\}^{2n}$. Since algorithm A is significantly faster on g than on g', we expect it to reach x_{opt}, the global optimum of g, while still having a Hamming distance of more than Δ to x'_{opt}. Then "the landscape changes": the algorithm is encouraged to stay on x_{opt} and is lead to $\overline{x'_{\text{opt}}}$. Therefore, it will be easy to find the global optimum of f, which is the concatenation of x_{opt} and $\overline{x'_{\text{opt}}}$. Algorithm A' is expected to find x'_{opt} before x_{opt}. Then, the only way to find the global optimum of f is to flip at least Δ bits simultaneously. Obviously, this is very unlikely. All our considerations implicitly assumed that changes are only made to the first n bits or to the last n bits. Changes effecting both, the front and the rear part, may cause unwanted effects. This may make changes to the general setup described above necessary.

We apply this technique to separate the CC (1+1) EA and the (1+1) EA. The two functions we use are LOB_2 and LEADINGONES. We know that the CC (1+1) EA clearly outperforms the (1+1) EA on LOB_2. We know that the advantage is larger on $\text{CLOB}_{b,l}$ with larger b, but that is not necessary for this technique. Therefore, we prefer this simpler function.

Definition 26. *For any constant $\varepsilon \in (0, 2/3)$, any constant $\delta \in (\max\{\varepsilon, 3\varepsilon - 1\}, 1)$, and any $n \in \mathbb{N}$ with $n \geq \lceil 2^{1/\varepsilon} \rceil$, we define $m := \lceil n^{\varepsilon} \rceil$ and the function*

$f_{\varepsilon,\delta}$ *by*

$$f_{\varepsilon,\delta}(x) := \begin{cases} n^{\varepsilon}\mathrm{LOB}_2\left(x_1 \cdots x_m\right) & \text{if } x_1 \cdots x_m \neq 1^m \text{ or} \\ -\textsc{OneMax}\left(x_1 \cdots x_m\right) & \textsc{LeadingOnes}(x_{m+1} \cdots x_n) \\ +n\textsc{LeadingOnes}\left(x_{m+1} \cdots x_n\right) & \geq n^{\delta} \\ n^3 - \textsc{OneMax}\left(x_{m+1} \cdots x_n\right) & \text{otherwise} \end{cases}$$

for all $x = x_1 \cdots x_n \in \{0,1\}^n$.

Theorem 27. *For any constant* ε *with* $0 < \varepsilon < 2/3$*, and any constant* $\delta \in (\max\{\varepsilon, 3\varepsilon - 1\}, 1)$*, the CC (1+1) EA operating with the two components* $x^{(1)} = x_1 \cdots x_m$ *and* $x^{(2)} = x_{m+1} \cdots x_n$ *on the function* $f_{\varepsilon,\delta}\colon \{0,1\}^n \to \mathbb{R}$ *has optimization time* T *with*

$$\mathit{Prob}\left(T = O\left(n^{1+\varepsilon}\log n\right)\right) = 1 - 2^{-\Omega(n^{\varepsilon})}.$$

Proof. We consider a run of the CC (1+1) EA on $f_{\varepsilon,\delta}$ with $0 < \varepsilon < 2/3$ and $\max\{\varepsilon, 3\varepsilon - 1\} < \delta < 1$. We describe conditions C_1, C_2, …, C_5 for a run. If a run satisfies all these conditions, then it is a run where the global optimum is reached within $O\left(n^{1+\varepsilon}\log n\right)$ generations. For each condition, we give an upper bound on the probability that it is violated. Showing that the sum of these upper bounds is $2^{-\Omega(n^{\varepsilon})}$ completes the proof.

We start with a description of the five conditions. Note, that the conditions need not to be independent. It is quite typical that some condition C_i makes sense only when some other condition C_j with $j < i$ is not violated.

C_1**:** After random initialization, $\textsc{OneMax}\left(x^{(2)}\right) \leq 3n^{\delta}/4$ holds.

C_2**:** Within the first $2en^{3\varepsilon}$ generations, we have that $x^{(1)} = 1^m$ holds for the first current string $x^{(1)}$.

C_3**:** Within the first $2en^{3\varepsilon}$ generations, the leftmost bit with value 0 in $x^{(2)}$ is mutated at most $n^{\delta}/12$ times. This condition assumes that C_1 is not violated.

C_4**:** In up to $n^{\delta}/12$ generations where the leftmost bit with value 0 in $x^{(2)}$ is mutated, the number of leading ones in $x^{(2)}$ is increased by at most $n^{\delta}/6$. This condition assumes that C_3 is not violated.

C_5: Let $x^{(1)} = 1^m$ and LeadingOnes $\left(x^{(2)}\right) < n^\delta$ hold for the current strings of the CC (1+1) EA. Then the global optimum of $f_{\varepsilon,\delta}$ is reached within $4en^{1+\varepsilon} \log n$ generations.

If C_2 holds, we have $x^{(1)} = 1^m$ within the first $2en^{3\varepsilon}$ generations. Due to the definition of $f_{\varepsilon,\delta}$ and the CC (1+1) EA, this will never change again. If C_1, C_3 and C_4 additionally hold, we have $x^{(1)} = 1^m$ and LeadingOnes $\left(x^{(2)}\right) \leq 11n^\delta/12$ within the first $4en^{3\varepsilon}$ generations. If C_5 holds, within the next $4en^{1+\varepsilon} \log n$ generations the global optimum is reached. We see that $T = O\left(n^{1+\varepsilon} \log n\right)$ holds in this case. Now we derive upper bounds on the probability that a condition is violated.

C_1: Initially, $x^{(2)}$ is drawn uniformly at random from $\{0,1\}^{n-m}$. Application of Chernoff bounds yields that with probability $2^{-\Omega\left(n^\delta\right)}$ condition C_1 is violated.

C_2: As long as $x^{(1)} \neq 1^m$ holds, there is always a mutation of at most two specific bits that increases the number of leading ones by at least two. The probability for such a mutation when $x^{(1)}$ is active is bounded below by $e^{-1} \cdot (1/m^2)$. When $x^{(1)}$ is not active, it cannot change. Thus, after at most $m/2$ such mutations, $x^{(1)} = 1^m$ holds. In $2en^{3\varepsilon}$ generations, $x^{(1)}$ is active $en^{3\varepsilon}$ times. Chernoff bounds yield that the probability not to have at least $m/2$ such mutations in this time is bounded above by $2^{-\Omega(n^\varepsilon)}$.

C_3: Within the first $2en^{3\varepsilon}$ generations, $x^{(2)}$ is active $en^{3\varepsilon}$ times. The probability to mutate a specific bit equals $1/(n-m) < 2/n$. It follows by Chernoff bounds that the number of such mutations is larger than $n^\delta/12$ with probability at most $2^{-\Omega\left(n^\delta\right)}$.

C_4: We consider up to $n^\delta/12$ steps where the number of leading ones in $x^{(2)}$ is increased via a direct mutation of the left most bit with value zero. We follow the analysis of the (1+1) EA on LeadingOnes (Droste, Jansen, and Wegener 2002). In r such steps, the number of leading ones is increased by $r + a$ where a is the number of bits right to the mutated bit with value 0 that happen to have value 1. The key observation is that all bits that are right of the leftmost bit with value 0 are random at all times. Thus, $\mathrm{E}(a) \leq n^\delta/24$ holds. This implies that C_4 is violated with probability $2^{-\Omega\left(n^\delta\right)}$.

C_5**:** The expected number of generations the CC (1+1) EA needs to optimize OneMax: $\{0,1\}^{n-m} \to \mathbb{N}$ is bounded above by $c(n-m)\log(n-m) \leq cn \log n$ (Rudolph 1997). Markov's inequality yields that $x^{(2)} = 1^{n-m}$ is achieved within $2cn \log n$ generations where $x^{(2)}$ is active with probability at least 1/2. Thus, with probability $2^{-\Omega(n^{\varepsilon})}$ this is not the case after $4cn^{1+\varepsilon} \log n$ generations.

□

Theorem 28. *For any constant* ε *with* $0 < \varepsilon < 2/3$*, and any constant* $\delta \in (\max\{\varepsilon, 3\varepsilon - 1\}, 1)$*,*

the (1+1) EA on the function $f_{\varepsilon,\delta}\colon \{0,1\}^n \to \mathbb{R}$ *has optimization time* T *with*

$$\mathit{Prob}\left(T \geq n^{n-2n^{\delta}}\right) = 1 - 2^{-\Omega(n^{\varepsilon})}.$$

Proof. It is convenient to use the notation $x^{(1)} := x_1 \cdots x_m$ and $x^{(2)} := x_{m+1} \cdots x_n$. After random initialization we have OneMax $\left(x^{(1)}\right) \leq 2m/3$ with probability $1 - 2^{-\Omega(n^{\varepsilon})}$. As long as $x^{(1)} \neq 1^m$ holds, the number of leading ones in $x^{(2)}$ cannot decrease due to the definition of $f_{\varepsilon,\delta}$.

It is easy to see that for LOB_2 all bits that are right of the left most block which is not all 1 are random at all times. Obviously, in $n^{\varepsilon}\mathrm{LOB}_2 - $OneMax there is an increased probability for these bits to have value 0. Thus, after the first $O\left(n^{1+\varepsilon}\log n\right)$ generations after random initialization we have at most $m/4$ leading ones blocks and at least $m/8$ blocks which are all 0 in $x^{(1)}$ with probability $1 - 2^{-\Omega(n^{\varepsilon})}$. This implies that for the next $O\left(n^2\right)$ generations $x^{(1)} \neq 1^m$ holds with probability $1 - 2^{-\Omega(n^{\varepsilon})}$.

From the results of Droste, Jansen, and Wegener (2002) it follows that with probability at least $1 - 2^{-\Omega(n)}$ we have $x^{(2)} = 1^{n-m}$ within $O\left(n^2\right)$ generations. Then, a mutation of at least $n - m - n^{\delta}$ bits simultaneously is necessary. Such a mutation has probability at most $n^{-n+n^{\varepsilon}+n^{\delta}}$. The probability for such a mutation within $n^{n-2n^{\delta}}$ steps is bounded above by $2^{-\Omega(n^{\varepsilon})}$. □

5 Implementing the CC (1+1) EA

Until this point, we have discussed only one possible implementation of the CC (1+1) EA, but there are clearly other choices one may make with respect to the relationship between the current component and the collaborating individuals from the other populations. Our choice is perhaps the simplest and most natural choice for a sequential machine of some type; however, there are other choices one might make that may be more natural for other settings, such as parallel computing environments.

This section presents such an alternative implementation and offers some initial analysis of this parallel CC (1+1) EA. We will show clearly when the two variants can be expected to have the same advantages and disadvantages when compared to a traditional (1+1) EA, as well as provide an example that demonstrates performance differences between the two variants.

5.1 Sequential and parallel implementations

The CC (1+1) EA presented earlier in this article makes a subtle, but potentially important implied decision with respect to how complete solutions are assembled: it always uses the most current version of the components in the other populations. While this is a reasonable choice, it is not at all the only one. One plausible alternative is to evaluate the current offspring in the context of the population from the last round. This idea is in some sense a more naturally parallel implementation, since in a given round the individual EAs may be processed in any order, asynchronously. Synchronization happens after a round is complete, rather than after each EA generation. Here each EA can be run on a separate processor, whereas Algorithm 3 requires that the EAs are executed one after another. Algorithm 29 below describes this new parallel variant in more detail. For clarity, we will now refer to the CC (1+1) EA defined in Section 2 as the *sequential* CC (1+1) EA, and this new algorithm as the *parallel* CC (1+1 EA, whenever this distinction is necessary.

Algorithm 29. (Parallel Cooperative Coevolutionary (1+1) Evolutionary Algorithm (Parallel CC (1+1) EA))

1. **Initialization**
Independently for each $i \in \{1, \ldots, k\}$, *choose* $x_0^{(i)} \in \{0,1\}^l$ *uniformly at random.*
2. $t := -1$
3. $a := 1; t := t + 1$
4. **Mutation**
Create $y^{(a)}$ *by copying* $x_t^{(a)}$ *and, independently for each bit,*
flip this bit with probability $\min\{1/l, 1/2\}$.
5. **Selection**

If $f(x_t^{(1)} \cdots y^{(a)} \cdots x_t^{(k)}) \geq f(x_t^{(1)} \cdots x^{(a)} \cdots x_t^{(k)})$*, set* $x_{t+1}^{(a)} := y^{(a)}$.
else set $x_{t+1}^{(a)} := x_t^{(a)}$.

6. $a := a + 1$
7. *If* $a > k$*, then continue at line 3, else continue at line 4*

There are several things the reader should be careful to note about the above algorithm. First, the only difference between this algorithm and the sequential variant is the selection step. Second, using the previous round's individuals for the evaluation affects not only the offspring evaluation, but the parent's as well. In fact, in the parallel variant we need only evaluate the parent once a round, since its value will never change while the round is in progress. This gives the parallel implementation an implicit advantage in terms of the number of evaluations performed, since the sequential CC (1+1) EA must evaluate the parent each EA generation; however, this difference is a mere constant factor and does not influence the asymptotic analysis.

5.2 Comparison on separable functions

Since we began our analysis of the sequential CC (1+1) EA with problems that are separable across population boundaries, perhaps it is most fitting to begin analysis of the parallel variant here as well. It turns out, as is shown in the Theorem 30, this task is made easier by the fact that the bounding complexities for their respective run times must be the same in all cases where the decomposition matches the separability. As such, all previously stated results for the sequential CC (1+1) EA functions that are separable across the population boundaries apply to the parallel variant. Indeed, there is no substantive analytical difference between these two algorithms for such functions.

Theorem 30. *The parallel and sequential forms of the the CC (1+1) EA perform identically on all problems that are separable across populations boundaries.*

Proof. We show that two algorithms must have the same performance because they behave identically in all cases against separable problems. Given that they behave identically, their respective expected waiting times for the the global optimum cannot differ by more than a factor of two.

For there to be any difference in behaviors, the algorithms must make different decisions during the selection step. Given that the function is separable across population boundaries, the linear contribution of the active component to the total fitness, $g_a : \{0,1\}^l \mapsto \mathbb{R}$, can be the only difference in the total fitness score. As a result, for a different selection decision to be made, it must be true that $g_a(y^{(a)}) \geq g_a(x_t^{(a)}) \Rightarrow g_a(y^{(a)}) < g_a(x_t^{(a)})$. Since both inequalities cannot be simultaneously true, the implication is false and there is a contradiction. □

5.3 Parallel CC (1+1) EA and global optimization

Since separability draws no distinction between these two variants of the CC (1+1) EA, if we are to discover a difference, that difference must be found in problems that have cross-population nonlinearities. Indeed, one may be tempted to intuit that the parallel version may be less susceptible to the pitfalls of the sequential version. After all, the algorithm essentially reserves judgment about a next potential step until after information about all offspring has been acquired. Perhaps this problem where the algorithm can fail to find the global optimum may not exist for the parallel version? Unfortunately, as is shown below, this problem exists for both variants.

Theorem 31. *Let* TRAP $: \{0,1\}^n \mapsto \mathbb{R}$*, be decomposed into* $k \geq 4$ *equal sized components of length* $l \geq 2$*, such that* $n := kl$*. The parallel CC (1+1) EA will fail to find the global optimum of* TRAP *with probability* $1 - 2^{-\Omega(n)}$.

Proof. As with the proof for Theorem 22, this proof consists of two steps. First we show that with a probability exponentially approaching 1, there are at least two populations of the parallel CC (1+1) EA whose individual do not contain the all one string after initialization. Next we show that, given there are at least two unsolved components after initialization, the mechanism of the algorithm itself will prevent it from accepting mutations that will lead it to the global optimum.

The first part of the proof holds directly from the proof for Theorem 22, since the initialization process for the two algorithms is the same.

For the second component of the proof, we once again consider the situation when all but two populations have individuals containing the all one string. It suffices to observe that the sequential CC (1+1) EA cannot make a different decision than the parallel version under this circumstance. The $y^{(a)}$ offspring again cannot be accepted since, $x^{(b)}$ does not contain the all one string, and again by symmetry the same is true with respect to $y^{(b)}$. □

As Theorem 31 shows, the parallel CC (1+1) EA can also be locked away from the global optimum, just as the sequential version is. Understanding the cause for the pathology in the first place may help shed light on why this is so for both algorithms. The algorithm becomes trapped specifically because of the partitioning. In order to solve the TRAP function, all n bits must flip simultaneously; however, because the algorithm considers only a component of the problem at a time, this cannot occur. In this (1+1) setting, there is no way to resolve this problem as long as there is this essential partitioning of the problem, regardless of how one implements the selection step. Moreover, it should be clear that for many population-based approaches this difficulty will still exist for the CCEA.

5.4 Separating the two CC (1+1) EA variants

Despite the fact that the two variations of the CC (1+1) EA perform the same on problems that are separable across population boundaries, and are both subject to some of the same pitfalls in terms of failure to find the global optimum for some inseparable problems, they are nevertheless different algorithms. As such, we should expect that for some problems with cross-population nonlinearities there may be a performance advantage of one over the other.

Indeed, in this subsection we prove by example that there can be enormous differences in performance of these two algorithms. We construct such an example in two stages. First, consider the g function below.

Definition 32. *For any $m \in \mathbb{N}$ with $m \geq 6$, we define $n := 2m$ and the function $g\colon \{0,1\}^n \to \mathbb{R}$. For $x = x_1 \cdots x_n \in \{0,1\}^n$ we write $x = x'x''$ with $x' = x_1 \cdots x_m \in \{0,1\}^m$ and $x'' = x_{m+1} \cdots x_n \in \{0,1\}^m$. We define g by*

$$g(x) := \begin{cases} 2n+2 & \text{if } x = 0^n \\ 2n + n \cdot \textsc{OneMax}(x) & \text{if } x' = 0^m \text{ or } x' = 0^m \\ 2n+1 & \begin{array}{l}\text{if } x' \neq 0^m \text{ and } x'' \neq 0^m \text{ and} \\ \textsc{OneMax}(x) = 2\end{array} \\ 2n + 2 \cdot \textsc{OneMax}(x) & \begin{array}{l}\text{if } x' \neq 0^m \text{ and } x'' \neq 0^m \text{ and} \\ \textsc{OneMax}(x) > 2 \text{ and} \\ (\textsc{OneMax}(x') < \lceil m/3 \rceil \text{ or} \\ \textsc{OneMax}(x'') < \lceil m/3 \rceil)\end{array} \\ n^4 + \textsc{OneMax}(x) & \text{otherwise} \end{cases}$$

for all $x \in \{0,1\}^n$.

It is easy to see that $g\colon \{0,1\}^n \to \mathbb{R}$ is not (r,s)-separable for any $s < n$. We will be interested in the performance of the two CC (1+1) EA variants on g when using two sub-populations, one operating on x' and the other operating on x''. Splitting these two, interrelated pieces up in this way will help provide opportunities for the parallel and sequential algorithms to come to different conclusions based on the information they have at their disposal when processing a given generation. In order to analyze these effects, we embed g into another function, SepFun, where the desired separation of the two variants will become very clear and quite easy to prove.

Definition 33. *For any $k \in \mathbb{N}$ with $k \geq 6$, we define $n := 4k^2$, $m := n/2 = 2k^2$ and the function* $\textsc{SepFun}\colon \{0,1\}^n \to \mathbb{R}$. *For $x = x_1 \cdots x_n \in \{0,1\}$ we write $x = x'x''$ with $x' = x_1 \cdots x_m \in \{0,1\}^m$ and $x'' = x_{m+1} \cdots x_n \in \{0,1\}^m$. For $x' = x_1 \cdots x_m \in \{0,1\}^m$ we write $x' = x^{(1)}x^{(2)} \cdots x^{(k)}$ with $x^{(i)} = x_{2(i-1)k+1} \cdots x_{2ik}$ for all $i \in \{1,\ldots,k\}$. We define* SepFun *by*

$$\textsc{SepFun}(x) := \begin{cases} n - \textsc{OneMax}(x') + \textsc{LeadingOnes}(x'') & \text{if } x'' \neq 1^m \\ n + \sum\limits_{i=1}^{k} g\left(x^{(i)}\right) & \text{otherwise} \end{cases}$$

for all $x \in \{0,1\}^n$.

We already know that neither of the two CC (1+1) EA variants guarantees global optimization. In particular, we do not expect that the CC (1+1) EA finds the global optimum of SepFun and instead change our perspective to one of approximation. We want to discover which of the two algorithm finds the larger function value in a number of generations, fixed in advance. We will see that for SepFun this is almost the same as letting the algorithms run until there is no improvement in function value for a fixed number of generations. Specifically, we concentrate on quite large, but still polynomial, numbers of generations.

Theorem 34. *Consider the sequential variant of the CC (1+1) EA on the function* $\textsc{SepFun}\colon \{0,1\}^n \to \mathbb{R}$ *($n = 4k^2, k \in \mathbb{N}, k \geq 6$) with at least $2k+1$ sub-populations operating on $x_{(i-1)k+1} \cdots x_{ik}$ for $i \in \{1,\ldots,2k\}$ and an arbitrary number of sub-populations on $x_{(n/2)+1} \cdots x_n$. Let F_t denote the maximum of all function values encountered in the first t generations. For all t with $t \geq 6n^{5/2} + 6n^2$ and $t = n^{O(1)}$*

$$\mathit{Prob}\left(F_t = \frac{n^2}{4} + n^{3/2} + n\right) = 1 - 2^{-\Omega\left(\sqrt{n}\right)}$$

holds.

Proof. We use the notation from Definition 32 and Definition 33. Similar to the proof of Theorem 27 we formulate a list of conditions on a run of the described CC (1+1) EA on SepFun. The run has polynomial length $t \geq 6n^{5/2} + n^2$. If the run complies with all conditions, the maximal encountered function value is $n^2/4 + n^{3/2} + n$. It is important to note that $f(x) = n^2/4 + n^{3/2} + n$ is equivalent to $x \in \{y_1 y_2 \cdots y_k 1^n \mid y_i \in \{0^k 1^k, 1^k 0^k\}\}$. We know from Theorem 24 that the number of sub-populations on $x_{(n/2)+1} \cdots x_n$ has no influence. Thus, we assume that we have just one sub-population operating there. It will be convenient to consider the algorithm in rounds, not generations, where a round consists of $2k+1$ generations.

C_1**:** For the first $\lceil n^{3/2} \log^2 n \rceil$ rounds, $x = x'x''$ with $x'' \neq 1^m$ holds.

C_2**:** After $\lceil n^{3/2} \log^2 n \rceil$ rounds, $x = 0^m x''$ with $x'' \neq 1^m$ holds and $F_t \leq n^2/4 + n^{3/2} + n$ holds during this time.

C_3**:** Given that C_1 and C_2 hold, within the first $\lceil 2en^2 \rceil$ rounds $x = 0^m 1^m$ holds for at least one generation and $F_t \leq n^2/4 + n^{3/2} + n$ holds during these rounds.

C_4**:** Given that C_1, C_2, and C_3 hold, after $6n^2$ rounds, $F_t = f(x) = n^2/4 + n^{3/2} + n$ holds.

C_5**:** Given that C_1, C_2, C_3 and C_4 hold, $f(x)$ does not change within the next t generations.

Condition C_4 guarantees $F_t = n^2/4 + n^{3/2} + n$ after round $\lceil 3en^2 \rceil$. Since condition C_5 states that there is no change of the function value within the next t generations, obviously $F_t = n^2/4 + n^{3/2} + n$ holds at the end of the t generations. Now we derive upper bounds on the probability that a condition may be violated.

C_1**:** It suffices to note that the sub-population operating on x'' deals with LeadingOnes: $\{0,1\}^{n/2} \to \mathbb{R}$. It is known that there exists a constant $c > 0$, such that the probability that $1^{n/2}$ is found in less than $c \cdot n^2$ generations is bounded above by $e^{-\Omega(n)}$ (Droste, Jansen, and Wegener 2002). Thus, condition C_1 is violated with probability $e^{-\Omega(n)}$.

C_2: Note, that $x'' \neq 1^m$ implies the bound on F_t. Furthermore, as long as $x'' \neq 1^m$ holds, if $x' = 0^m$ holds at some point of time this cannot change. We know from condition C_1 that $x'' \neq 1^m$ holds with probability $1 - e^{-\Omega(n)}$. Therefore, we only have to estimate for the probability that $x' = 0^m$ does not hold at some point of time within the first $\lceil n^{3/2} \log^2 n \rceil$ rounds. We know that there exists some constant c' such that after $c'n \log n$ steps we have $x^{(i)} = 0^{2k}$ for a sub-population with probability at least $1/2$. Thus, after $\lceil n^{3/2} \log^2 n \rceil > \sqrt{n} \cdot c'n \log n$ rounds, we have $x' = 0^m$ with probability $1 - e^{-\Omega\left(\sqrt{n}\right)}$.

C_3: We start our considerations with a current string $x = 0^m x''$, $x'' \neq 1^m$. As long as $x'' \neq 1^m$, we have $x = 0^m$ with probability 1. The probability not to have $x'' = 1^m$ within the first $\lceil 2en^2 \rceil$ rounds is bounded above by $e^{-\Omega(n)}$ (Droste, Jansen, and Wegener 2002). Due to the definition of the sequential CC (1+1) EA, in order to have $F_t > n^2/4 + n^{3/2} + n$ a mutation of at least $k/3$ bits is necessary in one of the sub-populations. The probability for such a mutation is bounded below by $(1/(2k))^{k/3} = n^{-\sqrt{n}/12}$. The probability that such a mutation occurs within $\lceil 2en^2 \rceil$ rounds is bounded above by $n^{3-\sqrt{n}/12}$.

C_4: It is easy to see that we have $F_t \leq n^2/4 + n^{3/2} + n$ with probability $1 - e{-}\Omega\left(\sqrt{n}\right)$. This can be shown in the same way as we did for condition C_3. It remains to be proven that $F_t \geq n^2/4 + n^{3/2} + n$ holds within the considered interval. In the worst case, we have $x = 0^m 1^m$ after round $\lceil 2en^2 \rceil$. Then there are $6n^2 - \lceil 2en^2 \rceil = \Omega\left(n^2\right)$ rounds left to consider. We can rule out mutations of at least $k/3$ bits with probability $1 - e^{-\sqrt{n}}$. Thus, it remains to prove that within more than $n^2/2$ rounds, each sub-population finds a bit string of the form $0^k 1^k$ or $1^k 0^k$. Since the probabilities of increasing the number of ones in $x^{(i)\prime}$ and $x^{(i)\prime\prime}$ are equal to those when optimizing OneMax, we see that this is the case with probability $1 - e^{-\Omega(n/\log n)}$.

C_5: If we have $f(x) = n^2/4 + n^{3/2} + n$, we can conclude that

$$x \in \left\{ y_1 y_2 \cdots y_k 1^n \mid y_i \in \left\{ 0^k 1^k, 1^k 0^k \right\} \right\}$$

holds. Due to the definition of g, x can only change to some string with different function value, if a mutation of at least $k/3$ bits occurs. The probability for such a mutation is bounded below by $(1/(2k))^{k/3} =$

$n^{-\sqrt{n}/12}$. The probability that such a mutation occurs within $n^{O(1)}$ generations is bounded above by $n^{-\Omega(\sqrt{n})}$.

□

For the parallel variant of the CC (1+1) EA, it is more difficult to predict a precise value for F_t. But we can prove a lower bound that is clearly larger than the value from Theorem 34. In order to have a fair comparison, we use the same division into sub-problems.

Theorem 35. *Consider the parallel variant of the CC (1+1) EA on the function* SEPFUN: $\{0,1\}^n \to \mathbb{R}$ *(*$n = 4k^2$*,* $k \in \mathbb{N}$*,* $k \geq 6$*) with at least* $2k+1$ *sub-populations operating on* $x_{(i-1)k+1} \cdots x_{ik}$ *for all* $i \in \{1, \ldots, 2k\}$ *and* $x_{(n/2)+1} \cdots x_n$*. Let* F_t *denote the maximum of all function values encountered in the first* t *generations. For all* t *with* $t \geq 6n^{5/2} + 6n^2$ *and* $t = n^{O(1)}$

$$\mathit{Prob}\left(F_t > n^4\right) > \mathit{Prob}\left(F_t = \Theta\left(n^{9/2}\right)\right) = 1 - 2^{-\Omega\left(\sqrt{n}\right)}$$

holds.

Proof. We follow the structure of the proof of Theorem 34. We consider the conditions C_1, C_2, and C_3 from there. The probability that a condition does not hold here is the same as there. Now we consider the situation when $x = 0^m 1^m$ holds. We group the sub-populations in k pairs operating on the bits $x_{(i-1)2k+1} \cdots x_{2ik}$. Such a pair operates on one piece of the function SEPFUN where the fitness is given by $n+g$. We consider the situation that the current string is 0^{2k} for each of these pairs. Thus, obviously, each pair is independent. For each pair the probability that after one round the fitness decreased from $3n+2$ to $3n+1$ is bounded below by $\left(\binom{k}{1}(1/k)(1-1/k)^{k-1}\right)^2 > e^{-2}$. In the following rounds, the probability to increase the number of 1-bits in each such sub-population is $\Omega(1)$ whereas the probability to reduce it is bounded above by $1/n^i$ when there are i 1-bits. Thus, with probability at least $e^{-2}/2$ we reach 1^{2k} in one such pair. The probability not to have this in at least one of the k pairs is bounded above by $2^{-\Omega(k)} = 2^{-\Omega\left(\sqrt{n}\right)}$. The same bound holds for the probability not to have at least $\Omega(k)$ such pairs. □

We have seen that the parallel variant is by far superior to the sequential implementation on the SEPFUN problem. As a note of caution, we explicitly add that this is due to the specific structure of SEPFUN and is, of course, no general property of the two variants. In fact, it is very easy to present a very similar function where the opposite behavior can be observed. We begin with defining a function SepFun′ that is very similar to SepFun.

Definition 36. *For any $k \in \mathbb{N}$ with $k \geq 6$, we define $n := 4k^2$, $m := n/2 = 2k^2$ and the function* SepFun′: $\{0,1\}^n \to \mathbb{R}$. *For $x = x_1 \cdots x_n \in \{0,1\}$ we write $x = x'x''$ with $x' = x_1 \cdots x_m \in \{0,1\}^m$ and $x'' = x_{m+1} \cdots x_n \in \{0,1\}^m$. For $x' = x_1 \cdots x_m \in \{0,1\}^m$ we write $x' = x^{(1)}x^{(2)} \cdots x^{(k)}$ with $x^{(i)} = x_{2(i-1)k+1} \cdots x_{2ik}$ for all $i \in \{1,\ldots,k\}$. We define* SepFun′ *by*

$$\textsc{SepFun}'(x) := \begin{cases} n - \textsc{OneMax}(x') + \textsc{LeadingOnes}(x'') & \text{if } x'' \neq 1^m \\ n + \sum\limits_{i=1}^{k} g'\left(x^{(i)}\right) & \text{otherwise} \end{cases}$$

for all $x \in \{0,1\}^n$.

The only difference is the use of g' instead of g. Again, the function g' is very similar to g.

Definition 37. *For any $m \in \mathbb{N}$ with $m \geq 6$, we define $n := 2m$ and the function $g'\colon \{0,1\}^n \to \mathbb{R}$. For $x = x_1 \cdots x_n \in \{0,1\}^n$ we write $x = x'x''$ with $x' = x_1 \cdots x_m \in \{0,1\}^m$ and $x'' = x_{m+1} \cdots x_n \in \{0,1\}^m$. We define g' by*

$$g'(x) := \begin{cases} 2n+2 & \text{if } x = 0^n \\ 2n + n \cdot \textsc{OneMax}(x) & \text{if } x' = 0^m \text{ or } x'' = 0^m \\ 2n+1 & \text{if } x' \neq 0^m \text{ and } x'' \neq 0^m \text{ and } \\ & \textsc{OneMax}(x) = 2 \\ 2n + 2 \cdot \textsc{OneMax}(x) & \text{if } x' \neq 0^m \text{ and } x'' \neq 0^m \text{ and } \\ & \textsc{OneMax}(x) > 2 \end{cases}$$

for all $x \in \{0,1\}^n$.

The behavior of the two algorithms will be similar on SepFun′ and on SepFun. But for SepFun′ the sequential implementation is clearly superior, because having $x^{(i)} \in \left\{1^k0^k, 0^k1^k\right\}$ yields much better pay-off than $x^{(i)} = 1^{2k}$. This suggests the reasonable idea that, while in other cases the parallel variant clearly performs better than the sequential algorithm, in some cases the reverse is true. The basic advantage of the parallel over the sequential CC (1+1) EA seems to center around whether the accrual of local information during a round will be misleading, while this advantage is reversed if the omission of such information is misleading.

6 Conclusions

While the application of coevolutionary algorithms towards optimization problems has gained increased popularity, our understanding of how such algorithms work and when they should be applied has, until recently, made less progress. This paper begins to bridge this gap by bringing traditional run time analysis tools to bear on a simple (1+1) form of the general cooperative coevolutionary architecture introduced by Potter and De Jong (1994). The end result is the early stages of work that helps demonstrate when and how the CC (1+1) EA may perform better than a (1+1) EA, as well as some of the reasons of when and how it may not. Additionally, we have considered a parallel variant of the CC (1+1) EA, also illustrating its similarities and differences from both the traditional (1+1) EA, as well as the sequential CC (1+1) EA. We have performed our analysis in the context of the important property of separability, commonly believed to be especially important for the success or failure of coevolutionary algorithms in general. The results are the beginnings of a deeper understanding of how cooperative coevolutionary algorithms work on optimization problems.

Perhaps the most important issue uncovered by this research is the clear dismissal of the property of separability as the main deciding factor in coevolutionary performance. We have provided analysis that clearly shows that neither is the property of separability a sufficient one to imply an advantage to the CC (1+1) EA over the (1+1) EA, nor is inseparability a sufficient enough property to imply a disadvantage.

With respect to separability, we have shown that the expected waiting time for both algorithms is $\Theta(n \log n)$ for OneMax, which is fully separable. Moreover, for linear functions in general the lower bound for the CC (1+1) EA is $\Omega(n \log n)$ and the expected waiting time for the (1+1) EA is $\Theta(n \log n)$. This again suggests no advantage to the CC (1+1) EA in spite of full separation of the problem. Further, while we have only demonstrated an upper bound of $O(n \log^2 n)$ for the CC (1+1) EA, we nevertheless conjecture that the lower bound is tight here, too.

With respect to inseparability, we have shown that there exist inseparable problems that can still easily be solved by the CC (1+1) EA, as well as the (1+1) EA. We provided a proof for the well-known LeadingOnes problem showing that both algorithms can be expected to find the global optimum in $\Theta(n^2)$ time, as well as provided a general technique for showing how there may be many inseparable problems as "easy" as OneMax by simply aggregating an inseparable function with OneMax. However, it should be noted that the inseparability property is not totally irrelevant. One can only guarantee that global optimum can be found when pieces of the problem may be linearly separated. While inseparability doesn't always mean that the CC (1+1) EA cannot find the global optimum, separability does always mean that it can.

Thus, coevolutionary advantage in this (1+1) framework does not arise from presence of the separability property, but arises when problems benefit from *both* the partitioning *and* the increased exploratory focus of genetic operators of the CC (1+1) EA. Problems that may benefit from these two elements working in conjunction with one another may be separable or inseparable, but the advantage is likely to increase as the need for greater exploratory power increases. In the case of the the $\text{CLOB}_{b,k}$ problem, the CC (1+1) EA is advantageous because a b-bit mutation is far more likely for it than for its EA analog, leading to a potentially super-polynomial advantage to coevolution. Increasing the mutation rate for the (1+1) EA will not help, since there will be significantly more disruption in the total string. When problems have such properties, the partitioning of coevolution can act as a protection against disruption for the other non-active pieces.

With this in mind, we constructed a function with cross-population nonlinearities that intentionally separates these two algorithms based on the principals described below. We prove that on this $f_{\varepsilon,\delta}$ function, the CC (1+1) EA outperforms the (1+1) EA exponentially in time in spite of the presence of inseparable problem pieces.

Our presentation of an alternative parallel implementation of the CC (1+1) EA revealed that the circumstances under which these two variants differ is perhaps less than obvious. The parallel and sequential forms of the CC (1+1) EA must perform identically on problems that are separable across the population boundary, but may even have the same performance on some problems with cross-population nonlinearities. We are able to show an example function that exploits the sequential CC (1+1) EA's potential to be misled by individual generations within a round; however, by reversing the roles of parts of this problem, we can reverse their performance roles as well. This counters the intuition that the parallel variant is less likely to be misled by a problem than the sequential algorithm, since either algorithm might be misled depending on the content of the local information likely to be uncovered during a round.

The cumulative result of this research is a much clearer picture of how the CC (1+1) EA works, as well as its relationship to a more traditional evolutionary approach. Moreover, many of the ideas discussed here clearly reflect more generally on the nature of CCEAs as applied to optimization tasks. For example, now that we know that separability is not the defining property for problem difficulty for the CC (1+1) EA, it is easy to see that this fact will be true for more complex CCEAs. Additionally, we presented a better understanding of when many types of coevolutionary algorithms can fail to find the global optimum. Finally, we've identified problem properties that allow CCEAs to gain advantage over EAs, which we believe are far more general than the (1+1) algorithms discussed here.

At least as important, we have laid the groundwork for future analysis by bringing the tools of run time analysis of randomized algorithms to bear on these questions of coevolution. With these tools, we would like to continue looking at the CCEA further, leaving the tight upper bound for linear functions as an open issue, but instead turning our attention towards examining population-based approaches. It will also be interesting to consider adaptive problem decompositions and explore ways in which the separability of a function may be estimated from sample points taken during the run. In combination, we hope that such efforts will yield algorithms that are not only useful optimization methods, but are also quite well understood.

References

L. Bull (1997). Evolutionary computing in multi-agent environments: Partners. In T. Baeck (Ed.), *Proceedings from the Seventh International Conference on Genetic Algorithms*, 370–377. Morgan Kaufmann.

D. Cliff and G. F. Miller (1995). Tracking the red queen: Measurements of adaptive progress in co–evolutionary sumulations. In *Proceedings of the Third European Conference on Artificial Life*, 200–218. Springer–Verlag.

S. Droste, T. Jansen, and I. Wegener (1998). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature (PPSN V)*, Berlin, Germany, 47–56. Springer.

S. Droste, T. Jansen, and I. Wegener (2002). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science 276*, 51–81.

R. Eriksson and B. Olsson (1997). Cooperative coevolution in inventory control optimisation. In G. Smith, N. Steele, and R. Albrecht (Eds.), *Proceedings of the Third International Conference on Artificial Neural Networks and Genetic Algorithms*, University of East Anglia, Norwich, UK. Springer.

S. Ficici and J. Pollack (1998). Challenges in coevolutionary learning: Arms–race dynamics, open–endedness, and mediocre stable states. In A. et al (Ed.), *Proceedings of the Sixth International Conference on Artificial Life*, Cambridge, MA, 238–247. MIT Press.

S. Ficici and J. Pollack (2000). A game-theoretic approach to the simple coevolutionary algorithm. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H.-P. Schwefel (Eds.), *Proceedings from the Sixth Conference on Parallel Problem Solving from Nature)*, 467–476. Springer-Verlag.

J. Garnier, L. Kallel, and M. Schoenauer (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation 7*(2), 173–203.

J. Horn, D. E. Goldberg, and K. Deb (1994). Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer (Eds.), *Parallel Problem Solving From Nature (PPSN III)*, Berlin, Germany, 149–158. Springer.

A. Iorio and X. Li (2002). Parameter control within a co-operative coevolutionary genetic algorithm. In J. J. Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel (Eds.), *Proceedings of the Seventh Conference on Parallel Problem Solving From Nature (PPSN VII)*, Berlin, Germany, 247–256. Springer.

T. Jansen and I. Wegener (2000). On the choice of the mutation probability for the (1+1) EA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel (Eds.), *Proceedings of the Sixth Conference on Parallel Problem Solving From Nature (PPSN VI)*, Berlin, Germany, 89–98. Springer.

T. Jansen and R. P. Wiegand (2003). Exploring the explorative advantage of the copoperative coevolutionary (1+1) EA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*. To appear.

K. Leung, T. Wong, and I. King (1998). Probabilistic cooperative-competitive hierarchical modeling for global optimization. In *Proceedings of the Fifth*

International Conference on Soft Computing and Information/Intelligent Systems, 748–751. World Scientific.

R. Motwani and P. Raghavan (1995). *Randomized Algorithms.* Cambridge: Cambridge University Press.

H. Mühlenbein (1992). How genetic algorithms really work. Mutation and hillclimbing. In R. Männer and R. Manderick (Eds.), *Proceedings of the Second Conference on Parallel Problem Solving from Nature (PPSN II)*, Amsterdam, The Netherlands, 15–25. North-Holland.

M. A. Potter and K. A. De Jong (1994). A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, and R. Männer (Eds.), *Proceedings of the Third Conference on Parallel Problem Solving From Nature (PPSN III)*, Berlin, Germany, 249–257. Springer.

G. Rudolph (1997). *Convergence Properties of Evolutionary Algorithms.* Hamburg, Germany: Dr. Kovač.

J. Scharnow, K. Tinnefeld, and I. Wegener (2002). Fitness landscapes based on sorting and shortest paths problems. In J. J. M. Guervos, P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacanas, and H.-P. Schwefel (Eds.), *Proceedings of the Seventh Conference on Parallel Problem Solving From Nature (PPSN VII)*, Berlin, Germany, 54–63. Springer.

K. O. Stanley and R. Miikkulainen (2002). The dominance tournament method of monitoring progress in coevolution. In *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2002.*

E. van Nimwegen and J. P. Crutchfield (2001). Optimizing epochal evolutionary search: Population-size dependent theory. *Machine Learning 45*(1), 77–114.

I. Wegener (2002). Methods for the analysis of evolutionary algorithms on pseudo-boolean functions. In R. Sarker, X. Yao, and M. Mohammadian (Eds.), *Evolutionary Optimization*, 349–369. Kluwer.

R. P. Wiegand, W. Liles, and K. De Jong (2001). An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In L. *et al.* Spector (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference 2001*, 1235–1242. Morgan Kaufmann.

R. P. Wiegand, W. Liles, and K. De Jong (2002a). The effects of representational bias on collaboration methods in cooperative coevolution. In J. J.

Merelo Guervós, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel (Eds.), *Proceedings of the Seventh Conference on Parallel Problem Solving From Nature (PPSN VII)*, Berlin, Germany, 257–268. Springer.

R. P. Wiegand, W. Liles, and K. De Jong (2002b). Modeling variation in cooperative coevolution using evolutionary game theory. In R. Poli, J. Rowe, and K. D. Jong (Eds.), *Foundations of Genetic Algorithms VII*, 231–248. Morgan Kaufmann.

C. Witt (2003). Personal communication.