The Cooperative Coevolutionary (1+1) EA

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

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

Abstract

Coevolutionary algorithms are variants of traditional evolutionary algorithms and are often considered more suitable for certain kinds of complex tasks than non-coevolutionary methods. One example is a general cooperative coevolutionary framework for function optimization. This paper presents a thorough and rigorous introductory analysis of the optimization potential of cooperative coevolution. Using the cooperative coevolutionary framework as a starting point, the CC (1+1) EA is defined and investigated from the perspective of the expected optimization time. The research concentrates on separability, a key property of objective functions. We show 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 its 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 overwhelming probability, 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 may even outperform it on certain inseparable functions.

Keywords

cooperative coevolution, run time analysis, separability, global optimization.

1 Introduction

An increasingly more common extension to traditional evolutionary algorithms is coevolution. In so-called coevolutionary algorithms (CEAs) individuals obtain fitness values corresponding to how well they behave in conjunction with other individuals. These algorithms seem to offer a great deal of promise, providing many potential advantages over traditional evolution. For example, there is reason to believe that they may be well-suited for problems with very large (perhaps infinite) search spaces. Additionally, coevolution is often attractive for problems that have no intrinsic objective measure. Also, in the case of cooperative coevolution, there is reason to believe they may be well-suited for problems with certain kinds of complex structural characteristics. The driving appeal for coevolution is the idea that an *arms race* can be established, where steady progress is made by mutual and reciprocal adaptations between different groups of individuals.

*The author was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531). Paul Wiegand currently holds a postdoctoral position with the American Society for Engineering Education and conducts research at the Naval Research Laboratory.

While it is clear that one can apply coevolution to tasks such as optimization, 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 has 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 focused on aspects of coevolutionary dynamics, representation, and progress measurement, while very little effort has been placed on 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.

1.1 The CCEA

Cooperative CEAs seem particularly well-suited for optimization problems having certain kinds of structural characteristics that can be exploited by its decompositional nature. The idea is that problems may benefit from partitioning potential solutions into smaller components that can be solved separately. One way of looking at this is that coevolutionary algorithms adaptively search *projections* of a complete problem space in parallel, and therefore allow for smaller search spaces in a given generation. Advantages such as this are leading researchers to apply CCEAs ever more frequently, often quite successfully. For example, cooperative coevolution has been successfully applied to problems such as function optimization (Potter and De Jong, 1994; Leung, Wong, and King 1998; Iorio and Li, 2002), job-shop scheduling (Eriksson and Olsson, 1997), concept learning for classification tasks (Potter and De Jong, 1998; Potter and De Jong, 2000), and behavioral learning for autonomous agents (Moriarty and Miikkulainen, 1997; Potter, Meeden, and Schultz, 2001).

In order to focus on the question of optimization, we adopt the well-known cooperative coevolutionary framework provided by Potter and De Jong (1994), which has several advantages for our interests. First, questions of objective progress measurement are greatly simplified. Also, the framework provides a very general architecture for optimization applications. Finally, the framework allows any evolutionary algorithm to be used as a part 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.

1.2 Problem Decomposition

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. Historically, problem separability has been considered the de-

mon of CCEAs. The intuition behind this is that breaking up strongly-related problem components will damage a CCEA's adaptive search capabilities. However, recently questions regarding the effects of interactivity among problem components have become a major focus of attention in empirical studies (Bull, 1997; Wiegand, Liles, and De Jong 2001; Bull, 2001; Wiegand, Liles, and De Jong, 2002a). While most of these studies suggest that problem separability is somehow related to problem difficulty, all of them state that the role of problem separability in predicting CCEA performance is far from clear. What *is* clear is that researchers applying these algorithms want to understand how to decompose problems for CCEA representational purposes and what effects such choices in conjunction with inherent problem properties have on CCEA performance. We believe that the best way to answer this is with rigorous run time analysis.

Specifically, our goal is to simplify the algorithm and investigate the effects that decompositional properties of certain problem classes have on run time performance in a context that is as clear and intuitive as possible. As a result, 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 is divided into k disjoint components. 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 decomposition is how run time performance is affected by the degree to which such representational choices match the true separability of the problem. Since the individual components are in some sense treated almost independently of one another, it is plausible to believe that a 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 focused 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 populations represent only components of a complete solution, and are therefore 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.

1.3 Paper Organization

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. Furthermore, we give an overview of the analytical tools and proof methods used throughout the paper.

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 can lead 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 which the 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 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 Fundamentals

2.1 Definitions

We choose to instantiate the cooperative coevolutionary function optimization framework of Potter and De Jong (1994) with the (1+1) EA as the 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 the 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 Rudolph (1997), Garnier, Kallel, and Schoenauer (1999), Droste, Jansen, 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 the 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 $y \in \{0,1\}^n$ *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 $f(x_t)$ can be assumed to be known in line four since it will have been

computed in the previous generation. Thus, the number of function evaluations is larger than the number of generations by exactly one. Definition 2 below formally describes the focus of attention in our analysis.

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.*

In this work, the (1+1) EA is used as the underlying search heuristic and we 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, considering this to be the most natural implementation for a sequential computing environment. We always use an even division of a bit string $x = x_1 \dots x_n \in \{0,1\}^n$ into k pieces $x^{(1)}, \dots, x^{(k)}$ of equal length l with $x^{(i)} = x_{(i-1)l+1} \cdots x_{il}$ and assume $l := (n/k) \in \mathbb{N}$.

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

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

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 (see Definition 6).

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 so that the separate components do not interfere with each other. These functions are called separable. Since separability is a key issue that must be discussed when analyzing the performance of the CC (1+1) EA, we give a precise formal definition.

Definition 4. *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 exists 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 related. 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 a 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 an 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.

Since we concentrate on the division of a bit string $x \in \{0,1\}^n$ into k separate components of equal size n/k, $(k,n/k)$-separable functions are of special interest to us. 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 a polynomial

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

with $w_I \in \mathbb{R}$, where $\mathcal{P}(\{1,\ldots,n\}$ denotes the set of all sub-sets of $\{1,\ldots,n\}$. 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.

Definition 5. *Let a function* $f : \{0,1\}^n \mapsto \mathbb{R}$ *be exactly* (r,s)*-separable as in Definition 4. 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, each EA operating on the bits contained exactly in 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.

2.2 Analytical Methods

Concentrating on the expected optimization time of evolutionary algorithms is similar to concentrating on the expected run time of randomized algorithms. Therefore, it is not surprising that the tools and proof methods applied throughout this paper are similar to proofs on the expected run times of randomized algorithms. Since they are not standard in all parts of the evolutionary algorithm community, we give a short overview on the main tools applied. Almost all these tools are implicit or explicit in the excellent text book by Motwani and Raghavan (1995). For the sake of completeness, we begin by providing definitions of the well-known notions we use to describe the asymptotic growth of functions.

Definition 6. *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 $\lim_{n\to\infty} f(n)/g(n) = 0$ holds. We say $f = \omega(g)$, if $g = o(f)$ holds.

The optimization time $T = T(A, f, n)$ of an evolutionary algorithm A on a problem f on a search space with size parameter n is a random variable. We are mostly interested in its mean value $\mathrm{E}(T)$. Since T cannot take negative values, Markov's inequality holds and we have $\mathrm{Prob}(T \geq t) \leq \mathrm{E}(T)/t$ for all $t \in \mathbb{R}^+$. For any random variable T we have $\mathrm{E}(T) = \sum_A \mathrm{Prob}(A) \cdot \mathrm{E}(T \mid A)$. The immediate consequence $\mathrm{E}(T) \geq \mathrm{Prob}(A) \cdot \mathrm{E}(T \mid A)$ is often helpful in obtaining a lower bound on the expected optimization time: The expected optimization time may be easy to estimate given that some (typical) event A occurs. Thus, we have a lower bound on $\mathrm{E}(T \mid A)$ and get a good lower bound on $\mathrm{E}(T)$ if we can prove that $\mathrm{Prob}(A)$ is not too small.

We consider pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$. Thus, bit strings $x \in \{0,1\}^n$ of length n are of special interest. Often, some property A holds independently for each bit with some probability p. Then, Chernoff bounds are extremely useful for obtaining sharp bounds on the probability that the collection of bits deviates from its expected behavior. In particular, let X be the number of bits that have the property A. The random variable X is binomially distributed with parameters n and p, thus we have $\mathrm{E}(X) = pn$. Chernoff bounds yield that $\mathrm{Prob}(X \geq (1+\delta)\mathrm{E}(X)) \leq \left(e^\delta/(1+\delta)^{1+\delta}\right)^{\mathrm{E}(X)}$ for any $\delta > 0$ and that $\mathrm{Prob}(X \leq (1-\delta)\mathrm{E}(X)) \leq e^{-\mathrm{E}(X)\delta^2/2}$ for any $0 < \delta < 1$.

Another useful tool in a similar context is the coupon collector's problem. Assume one is interested in getting n different coupons. The coupons are obtained one after the other and each coupon is obtained with equal probability $1/n$ at each step. We are interested in the number of coupons C one has to obtain until a collection of all n different coupons is complete. It is known that $\mathrm{E}(C) = n \ln n + O(n)$ and $\mathrm{Prob}(C > \beta n \ln n) \leq n^{1-\beta}$ for any $\beta \geq 1$ hold. When you think of the n bits of a bit string as coupons and of obtaining a coupon as mutating the corresponding bit, then the connection to evolutionary algorithms becomes apparent.

3 Exploiting separability

When using the CCEA framework for function optimization 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 sub-section 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 results of Sections 3.1 and 3.2 significantly extend work already published (Jansen and Wiegand, 2003a). 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

For linear functions, the expected optimization time for the (1+1) EA is $O(n \log n)$ and this upper bound matches the lower bounds for typical linear functions, as derived by Droste, Jansen, and Wegener (2002).

Theorem 7. *The expected optimization time of the (1+1) EA on a linear function $f\colon \{0,1\}^n \to \mathbb{R}$ 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 functions since they are exactly $(n, 1)$-separable. 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 and which have no properties of special interest. Since we are especially 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 $\Omega(n \log n)$.*

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, thus $\mathrm{E}(T) \geq p_t \cdot t \cdot k$ holds.

After random initialization with probability at least $1/2$, at least $\lfloor n/2 \rfloor$ of the bits

in the initial string differ from x^*. Each of these bits must be mutated at least once in order to find the global optimum x^*.

First, we assume that $k < n$ holds. This implies $l = (n/k) \geq 2$. We consider the situation after $(l-1)\ln n$ 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 that need to be mutated at least once there is at least one that is not within $(l-1)\ln n$ complete rounds equals $1-\left(1-(1-1/l)^{(l-1)\ln n}\right)^{\lfloor n/2 \rfloor}$. We see that we have

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

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

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

for $k < n$.

For $k = n$ we have n (1+1) EAs each operating on exactly one bit. Each bit has a unique optimal value. We are waiting for the first point of time when each bit has 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 halved in each round. It is easy to see 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; 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 $p_f(t,n)$ 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 $l := n/k$ bits each is $E(T) = O\left((tn\ln l) - tn(\ln l)(\ln k)/\ln p_f(t,l)\right)$.*

Proof. We consider periods of time that consist of $tl\ln l$ rounds ($tn\ln l$ generations) 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 $1-p_f(t,l)$. 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 decreases from s to $p_f(t,l) \cdot s$. Markov's inequality yields that the probability of having at least $2 \cdot p_f(t,l) \cdot s$ such (1+1) EAs is bounded above by $1/2$. We conclude that on average after at most $2tl \ln l$ rounds we decrease the number of (1+1) EAs that are not yet optimal from s to at most $2p_f(t,l)$. We call such a sequence of $2tl \ln l$ rounds a *phase*. After r such phases we have at most $(2p_f(t,l))^r \cdot s$ (1+1) EAs that are not yet optimal if we started with s such components. We have $s \leq k$ and are interested in a small value of r such that $(2p_f(t,l))^r \cdot k \leq 1$ holds. We see that $r \geq (\ln k)/\ln(1/(2p_f(t,l)))$ is sufficient. This proves the lemma. □

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 $1/2$. Due to the independence of the initial bit string we can consider the next $2cn \ln n$ generations as another run of length $2cn \ln n$. Thus, we have $p_f(2ct,n) \leq 2^{-t}$ for any linear function f and any $t \in \mathbb{N}$. Inserting $p_f(2ct,n) \leq 2^{-t}$ in the expression for $\mathrm{E}(T)$ from Lemma 9 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* $O(n \ln l(1+\ln k)) = O(n \log^2 n)$.

ONEMAX is a special linear function which can give us a better bound on $p_{\text{ONEMAX}}(t)$. The Hamming distance to the global optimum can never increase and the probability of increasing 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$\colon \{0,1\}^n \to \mathbb{R}$ *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 in general. For very large and very small values of k ($k = \Omega(n)$, $k = O(1)$), we already have an optimal bound from Theorem 10. For values in between (like $k = \sqrt{n}$) we only get the weaker bound $O(n \log^2 n)$ from Theorem 10. 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* $\Theta(n \log n)$.

3.2 Exploring the explorative advantage

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 of the CC (1+1) EA is the increased mutation probability. What is expected from an increased mutation probability is, of course, an increased number of mutations. To see this more clearly we classify a mutation by the number of mutated bits. A comparison is made between one round of the CC (1+1) EA, when each (1+1) EA was active once, and k generations of the traditional (1+1) EA. We concentrate on the first component only and calculate the expected

number of mutations of exactly b bits. For a single (1+1) EA operating on l bits, this number equals $\binom{l}{b}(1/l)^b(1-1/l)^{l-b}$. Thus, for the (1+1) EA operating on n bits for k generations, this number equals $k \cdot \binom{l}{b}(1/n)^b(1-1/n)^{l-b}$; the term $\binom{l}{b}$ stems from the fact that we only count mutations of exactly b bits in the first component. Considering the quotient of the two we get

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

The term is governed by k^{b-1}. 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 expect to see significant differences in the performance of the traditional (1+1) EA and its cooperative coevolutionary counterpart when such mutations are important. This motivates the definition of a family of functions where such mutations are crucial.

We start with a function that 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 introduced by van Nimwegen and Crutchfield (2001) in a different context. The function value equals the number of consecutive 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 $n \in \mathbb{N}$, $k \in \{1, \ldots, n\}$ 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*

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

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. 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.

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. We have k components $x^{(1)}, \ldots, x^{(k)}$ of length $l = n/k$ each. In each component the size of the blocks relevant for $\mathrm{CLOB}_{b,k}$ 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. In each of these generations, we have such a mutation with probability at least $1/(el^b)$ and call the actual, random number of such mutations M. We have $\mathrm{E}(M) \geq 10((l/b)+\ln k)$ and can apply Chernoff bounds. This yields $\mathrm{Prob}(M \leq (1-9/10)\mathrm{E}(M)) \leq \mathrm{Prob}(M \leq (l/b)+\ln k) \leq e^{-\mathrm{E}(M)(9/10)^2/2} \leq e^{-(81/20)((l/b)+\ln k)} \leq e^{-4((l/b)+\ln k)} \leq \min\{e^{-4}, k^{-4}\}$. In the case $k = 1$ this immediately implies the claimed upper bound. Otherwise, the probability that there is a component different from 1^l is bounded above by $k \cdot (1/k^4) = 1/k^3$, which 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 $\Omega(k)$ populations where $x^{(i)} = 0^l$ holds after $o\left(k \cdot l^2\right)$ generations. Second, we prove that for these populations $x^{(i)} \neq 1^l$ holds with a probability that is sufficiently large for the proof of the lower bound.

Each component equals 0^l with probability 2^{-l} after random initialization. If $l = O(1)$, this yields that each population is initialized 0^l with probability $\Omega(1)$. If 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 $1 - \left((3/4) + 2^{-5} + (10e\log l)/l\right) = \Omega(1)$. Application of Chernoff bounds yields in both cases that with probability close to 1 the number of 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 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 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 $\left(1-1/l^b\right)^{(l^b-1)\ln k} \geq e^{-\ln k} = 1/k$. 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 population has $x^{(i)} \neq 1^l$ is bounded below by $(1-1/k)^{ck} > \frac{e^{-c}}{2}$. 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 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 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. □

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 $\text{CLOB}_{b,k}$. 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* $\text{CLOB}_{b,k}\colon \{0,1\}^n \to \mathbb{R}$ *is* $\Theta\left(n^b\left(n/(bk) + \ln k\right)\right)$*, if* $2 \leq b \leq n/k$*,* $1 \leq k \leq n/4$*, and* $n/(bk) \in \mathbb{N}$ *hold.*

For the poof of Theorem 16 it is useful to have a tool similar to, and slightly more general than, the coupon collector's problem. We state and prove this tool in the more common language of balls and bins.

Lemma 17. *Consider balls that 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.*

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

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 $\mathrm{E}(M)$. 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 $\mathrm{E}(M) \geq (1-1/e)\cdot(k-1)\ln n$. Together, we have $\mathrm{E}(M) = \Omega(kh + k\ln k)$.

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

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

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 $\mathrm{E}(M) \leq \left(1 - 1/\sqrt{k}\right)^{-1} \cdot (4kh + 4k\ln k) =$ $O(kh + k\ln k)$. □

Proof of Theorem 16. 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 $\text{CLOB}_{b,k}$. 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 $\text{CLOB}_{b,k}(y) < \text{CLOB}_{b,k}(x)$ in this case and 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 $\text{CLOB}_{b,k}(y) < \text{CLOB}_{b,k}(x)$ can be ignored. Thus, we are only concerned with the case $\text{CLOB}_{b,k}(y) \geq \text{CLOB}_{b,k}(x)$. 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 $\text{CLOB}_{b,k}$.

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 of any one becoming a 1-block in y is bounded above by $1/n^2$. We know that the $-$ONEMAX part of $\text{CLOB}_{b,k}$ 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

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

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 that 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 $x^{(1)}, \dots, x^{(k)}$ 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 $i \in \{1, \dots, k\}$, $x \in \{0,1\}^n$ 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 that 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 that is not the all one string is given by

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

Note that $z_i(x)$ 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, for all x, Prob $(A(x)) \geq \binom{k}{1} n^{-b} (1 - 1/n)^{n-b} \geq k/(en^b)$ 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 $O\left(n^b/k\right)$ generations for each advancing step, the total expected waiting time until the global optimum is reached is $O(n^b \ln k)$. □

We have argued above that it may be beneficial to apply the increased mutation probability in a directed way due to the cooperative coevolutionary approach. We have analyzed the CC (1+1) EA and the (1+1) EA on the same problem to investigate this idea. Thus, our interest is not specifically concentrated on the concrete expected optimization times. Rather, we are much more interested in a comparison between the two algorithms. 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 $\text{CLOB}_{b,k}$. 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 $\text{CLOB}_{b,k}$ 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 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 $\text{CLOB}_{b,k}$ 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. However, separability by itself is not sufficient to make the cooperative coevolutionary approach beneficial. Could there even be disadvantages due to this approach 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. 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 exponential expected optimization time. We saw that the superior performance of the CC (1+1) EA on $\text{CLOB}_{b,k}$ 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 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, the possible advantage of the (1+1) EA is limited. 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 $I_1, \ldots I_r$ and functions $g_1, \ldots, g_r$ such that

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

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

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

for all $j \in \{1, \ldots, r\}$. 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 $j \in \{1, \ldots, r\}$. 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 that g_l is exactly $(1, l)$-separable, thus $f_{k,l}$ 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.

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 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 $\mathrm{E}(T \mid F) = O(n^4)$ hold. We have $\mathrm{E}(T) \leq \mathrm{E}\left(T \mid \overline{F}\right) + \mathrm{Prob}(F) \cdot \mathrm{E}(T \mid F)$. Therefore, it is sufficient to prove $\mathrm{Prob}(F) = O(1/n^2)$.

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 $n - \textsc{OneMax}(x)$. Consider $B_i := \{x \in \{0,1\}^n \mid \textsc{OneMax}(x) = i\}$. For symmetry reasons, each point $y \in B_i$ has equal probability of becoming the 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 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

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

□

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 of having this in at least 1 of the k independent components is $p(l,k) := 1 - \left(1 - \Theta(1/l^3)\right)^k$. 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 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 of reaching $0^{n-4}1111$. 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 of entering 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 of having $x = 0^{n-4}1111$ 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 could get a larger relative performance advantage if we chose a more complicated example function.

While knowing the asymptotic results of the theory is extremely useful, without knowing the specific constants involved, we cannot know how large the problem space must be before we see a noticeable difference in performance. In fact, in this case the problem need not be particularly large at all. To justify this assertion, we construct experiments that apply both algorithms to the $f_{k,l}$ problem for varying parameter values of $l \in \{2, \ldots, 8\}$, where $k = l^4$, resulting in 14 experiments (7 per algorithm). There are

30 independent trials run for each group, where each trial reports the number of function evaluations needed until the first time the optimum is reached. Figure 1 below plots the mean results for each algorithm as a function of $n = l^5$.

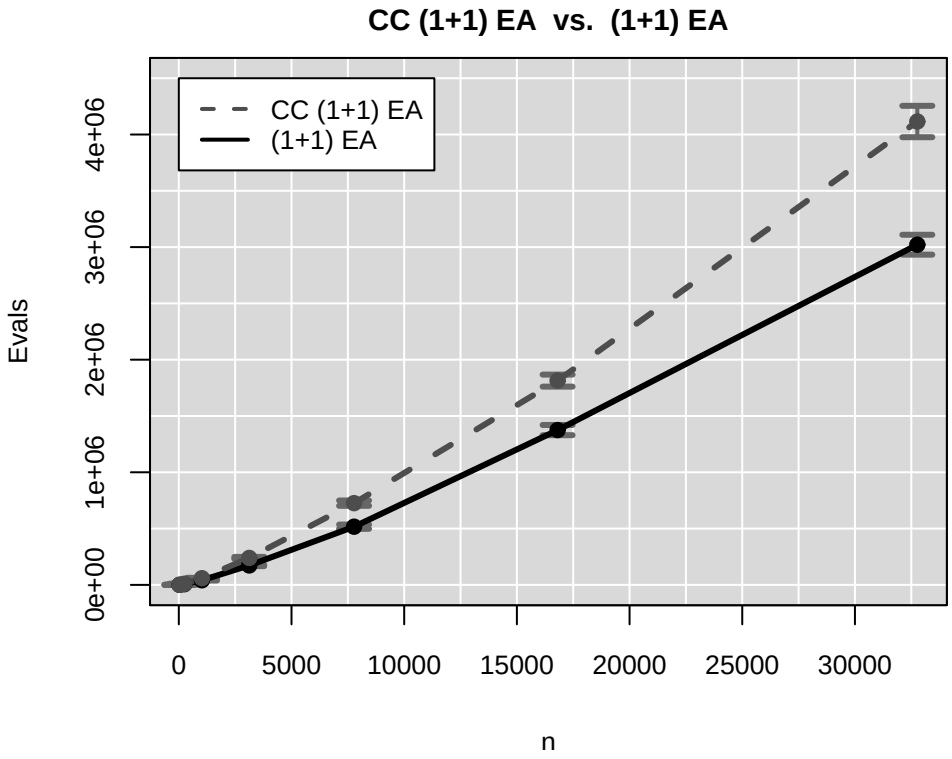

Figure 1: Results from experimental comparison of (1+1) EA and CC (1+1) EA for the $f_{k,l}$ problem, where $l \in \{2, \ldots, 8\}$ and $n = l^5$. Points on the graph represent mean number of evaluations needed to obtain the global max for 30 independent runs.

In all cases of this experiment ($n \in \{32, \ldots, 32768\}$), the performance of the (1+1) EA is statistically significantly better than the CC (1+1) EA within a 99% level of confidence. These statistical tests were performed by making 7 pair-wise Welch t-test comparisons (one for each value of n), then using the Bonferroni adjustment to allow for valid multiple comparison testing (Hancock and Klockars, 1996).

We now turn our attention to a discussion of how larger relative performance differences can be achieved. 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 significantly less likely. Since the path 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, 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 the cross-population nonlinearities of a problem 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 is commensurate 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 disadvantage to the CC (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 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 depends 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.

This is the case for some inseparable functions. Definition 21 below describes the very well-known, exactly $(1, n)$-separable TRAP function. There we can not only justify our intuition, but also give it a little more clarity.

Definition 21. *For $n \in \mathbb{N}$, we define* TRAP$\colon \{0,1\}^n \to \mathbb{R}$ *by* TRAP$(x) := n - \text{ONEMAX}(x) + (n+1) \cdot \prod_{i=1}^{n} x_i$ *for all $x = x_1 \cdots x_n \in \{0,1\}^n$.*

Theorem 22. *Let* 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* 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 populations 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, the algorithm cannot accept mutations leading it to the global optimum.

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 populations equals $k/2^l$. Chernoff bounds yield that the probability that more than half of the k populations are solved initially is at most

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

So the probability that at least half of the k populations are unsolved is $1 - 2^{-\Omega(n)}$. With $k \geq 4$ this means at least two unsolved components.

Now, let us suppose that, after initialization, all the populations contain the all one string except for two of them, one that is the current active population 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 TRAP$(x^{(1)} \cdots y^{(a)} \cdots x^{(b)} \cdots x^{(k)}) \geq$ 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 $x^{(b)}$. Having more than two unsolved components cannot resolve the problem since an 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, for other fitness functions 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 $\text{CLOB}_{b,k}$ problem discussed in the last section.

Though our proof demonstrates only that a particular CCEA cannot find the global optimum for certain kinds of problems, it is not difficult to see that the this difficulty affects a much larger class of such algorithms. For example, modifying our essentially sequential selection and update mechanism of populations to one that is more parallel will not resolve the difficulty (Jansen and Wiegand, 2003b). Simply adding a population will also not resolve the difficulty since retaining poorly performing components near the global optimum would imply populations of exponential size. Likewise, for populations of constant or polynomial sizes, adding crossover will not help since the populations will not retain components having genetic material that can be productively recombined. Of course, one could come up with a CCEA tailored towards this example function, but that is not the point. It should come as no surprise that the advanced exploratory capability of the cooperative coevolutionary framework has its price.

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 section, it is not.

Consider the well-known LEADINGONES problem, which is identical to LOB_1. This problem is exactly $(1, n)$-separable, so there exists no way to partition it into smaller, separable components. 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). But the CC (1+1) EA's expected optimization time is $\Theta(n^2)$, too.

Theorem 23. *The expected optimization time for the CC (1+1) EA on the function* LEADINGONES *is* $\Theta(n^2)$.

Proof. For the upper bound, 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 the successive solutions of the leading ones problems for each component. The lower bound for a given component can be obtained from Droste, Jansen, and Wegene (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 $l > 3$. Since the string is random until the component becomes progressive, a component that becomes progressive has r leading ones with probability $2^{-(r+1)}$ for $r < l$ and 2^{-r} for $r = l$. Thus, with probability $1 - 2^{-(2/3)l}$ the progressive component starts with less than $2/3$ leading ones. From Droste, Jansen, and Wegener (2002) we know that then 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 that 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.

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.

Definition 24. *For any $c > 0$, the function* NEEDLEOM$_c$*: $\{0,1\}^n \to \mathbb{R}$ is defined by NeedleOM$_c(x) := c \cdot$ ONEMAX$(x) + n \cdot \prod_{i=1}^{n} x_i$ for all $x = x_1 \cdots x_n \in \{0,1\}^n$.*

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 the CC (1+1) EA.

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.

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

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

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. For large Δ 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 affecting both, the front and the rear part, may cause unwanted effects. This may make changes to this general setup necessary.

This technique can obviously be applied to separate the CC (1+1) EA and the (1+1) EA. The two functions used here are LOB_2 and LEADINGONES. We know that the CC (1+1) EA clearly outperforms the (1+1) EA on LOB_2. We also know that the advantage is larger on $\text{CLOB}_{b,l}$ with larger b, but that is not necessary for this technique. Therefore, this simpler function is our preference.

Definition 25. *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*

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

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

Theorem 26. *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.

C_1**:** After random initialization, 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 $\{0,1\}^{n-m}$. Chernoff bounds yield 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 of mutating 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 $2^{-\Omega(n^\delta)}$.

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 27. *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 $\textsc{OneMax}\left(x^{(1)}\right) \leq 2m/3$ 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 that is not all 1 are random at all times. Obviously, in $n^\varepsilon\mathrm{LOB}_2 - \textsc{OneMax}$ 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 that 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^δ} steps is bounded above by $2^{-\Omega(n^\varepsilon)}$. □

5 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). Such work 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. 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 the property of separability

is neither 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 for linear functions there is no advantage to the CC (1+1) EA in spite of full separation of the problem.

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. However, it should be noted that the inseparability property is not totally irrelevant. One can only guarantee that a global optimum can be found when independent optimization of the components results in global optimization.

The coevolutionary advantage does not arise from the presence of the separability property, but arises when problems benefit from *both* the partitioning *and* the increased exploratory focus of the 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 $\text{CLOB}_{b,k}$ problem, the CC (1+1) EA is advantageous because a b-bit mutation is far more likely for it than for its EA analog. Increasing the mutation rate for the (1+1) EA will not help since there will be significantly more disruption in the total string. For problems with such properties, the partitioning of coevolution can act as a protection against disruption.

With this in mind, we constructed a function with cross-population nonlinearities that intentionally separates these two algorithms. There we prove that the CC (1+1) EA outperforms the (1+1) EA exponentially in time in spite of the presence of inseparable problem pieces.

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 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.

We have laid the groundwork for future analysis by applying run time analysis tools to these questions of coevolution. With these tools, we would like to continue looking at the CCEA further, 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

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

Bull, L. (2001). On coevolutionary genetic algorithms. *Soft Computing*, 5:201–207.

Cliff, D. and Miller, G. F. (1995). Tracking the red queen: Measurements of adaptive

progress in co–evolutionary sumulations. In Moran, A., Merelo, J., and Chacon, P., editors, *Proceedings of the Third European Conference on Artificial Life*, pages 200–218. Springer.

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

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

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

Ficici, S. and Pollack, J. (1998). Challenges in coevolutionary learning: Arms–race dynamics, open–endedness, and mediocre stable states. In Adami, C., Belew, R., Kitano, H., and Taylor, C., editors, *Proceedings of the Sixth International Conference on Artificial Life*, pages 238–247. MIT Press.

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

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

Hancock, G. R. and Klockars, A. J. (1996). The question for alpha; developments in multiple comparison procedures in the quarter century since games (1971). *Review of Educational Research*, 66:269–306.

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

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

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

Jansen, T. and Wiegand, R. P. (2003a). Exploring the explorative advantage of the cooperative coevolutionary (1+1) EA. In Cantu-Paz, E., Foster, J. A., Deb, K., Davis, L. D., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schulz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 310–321. Springer.

Jansen, T. and Wiegand, R. P. (2003b). Sequential versus parallel cooperative coevolutionary (1+1) eas. In McKay, B., editor, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 30–37. IEEE Press.

Leung, K., Wong, T., and King, I. (1998). Probabilistic cooperative-competitive hierarchical modeling for global optimization. In *Proceedings of the Fifth International Conference on Soft Computing and Information/Intelligent Systems*, pages 748–751. World Scientific.

Moriarty, D. and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399.

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

Potter, M. and De Jong, K. (1998). The coevolution of antibodies for concept learning. In Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors, *Proceedings from the Fifth International Conference on Parallel Problem Solving from Nature*, pages 530–539. Springer.

Potter, M. and De Jong, K. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Potter, M., Meeden, L., and Schultz, A. (2001). Heterogeneity in the coevolved behaviors of mobile robots. In Nebel, B., editor, *Proceedings of the Seventeenth International Conference on Artificial Intelligence*, pages 1337–1343. Morgan Kaufmann.

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

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

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

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

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

Wiegand, R. P., Liles, W., and De Jong, K. (2001). An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voight, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference 2001*, pages 1235–1242. Morgan Kaufmann.

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

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

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

Witt, C. (2003). Population size vs. runtime of a simple EA. In Sarker, R., Reynolds, R., Abbass, H., Tan, K. C., McKay, B., Essam, D., and Gedeon, T., editors, *Proceedings of the Congress on Evolutionary Computation (CEC 2003)*, pages 1996–2003. IEEE Press.