Exploring the Explorative Advantage of the Cooperative Coevolutionary (1+1) EA

Thomas Jansen[1]* and R. Paul Wiegand[2]

[1] FB 4, LS2, Univ. Dortmund, 44221 Dortmund, Germany
Thomas.Jansen@udo.edu
[2] Krasnow Institute, George Mason University, Fairfax, VA 22030
paul@tesseract.org

Abstract. Using a well-known cooperative coevolutionary function optimization framework, a very simple cooperative coevolutionary (1+1) EA is defined. This algorithm is investigated in the context of expected optimization time. The focus is on the impact the cooperative coevolutionary approach has and on the possible advantage it may have over more traditional evolutionary approaches. Therefore, a systematic comparison between the expected optimization times of this coevolutionary algorithm and the ordinary (1+1) EA is presented. The main result is that separability of the objective function alone is is not sufficient to make the cooperative coevolutionary approach beneficial. By presenting a clear structured example function and analyzing the algorithms' performance, it is shown that the cooperative coevolutionary approach comes with new explorative possibilities. This can lead to an immense speed-up of the optimization.

1 Introduction

Coevolutionary algorithms are known to have even more complex dynamics than ordinary evolutionary algorithms. This makes theoretical investigations even more challenging. One possible application common to both evolutionary and coevolutionary algorithms is optimization. In such applications, the question of the optimization efficiency is of obvious high interest. This is true from a theoretical, as well as from a practical point of view. While for evolutionary algorithms such run time analyses are known, we present results of this type for a coevolutionary algorithm for the first time. Coevolutionary algorithms may be designed for function optimization applications in a wide variety of ways. The well-known cooperative coevolutionary optimization framework provided by Potter and De Jong (7) is quite general and has proven to be advantageous in different applications (e.g., Iorio and Li (4)). An attractive advantage of this framework is that any evolutionary algorithm (EA) can be used as a component of the framework.

* The research was partly conducted during a visit to George Mason University. This was supported by a fellowship within the post-doctoral program of the German Academic Exchange Service (DAAD).

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 310–321, 2003.

However, since these cooperative coevolutionary algorithms involve several EAs working almost independently on separate *pieces* of a problem, one of the key issues with the framework is the question of how a problem representation can be decomposed in productive ways. Since we concentrate our attention on the maximization of pseudo-Boolean functions $f: \{0,1\}^n \to \mathbb{R}$, there are very natural and obvious ways we can make such representation choices. A bit string $x \in \{0,1\}^n$ of length n is divided into k separate components $x^{(1)}, \ldots, x^{(k)}$. Given such a decomposition, there are then k EAs, each operating on one of these components. When a function value has to be computed, a bit string of length n is reconstructed from the individual components by picking representative individuals from the other EAs. Obviously, the choice of the EA that serves as underlying search heuristic has great impact on the performance of this cooperative coevolutionary algorithm (CCEA). We use the well-known (1+1) EA for this purpose because we feel that it is perhaps the simplest EA that still shares many important properties with more complex EAs, which makes it an attractive candidate for analysis.

Whether this mechanism of dividing the optimization problem f into k subproblems and treating them almost independently of one another is an advantage strongly depends on properties of the function f. In applications, *a priori* knowledge about f is required in order to define an appropriate division. We neglect this problem here and investigate only problems where the division in sub-problems matches the objective function f. The investigation of the impact of the separation of inseparable parts is beyond the scope of this paper. Intuitively, separability of f seems to be necessary for the CCEA to have advantages over that EA this is used as underlying search heuristic. After all, we could solve linearly separable blocks with completely independent algorithms and then concatenate the solutions, if we like. Moreover, one expects that such an advantage should grow with the degree of separability of the objective function f. Indeed, in the extreme we could imagine a lot of algorithms simultaneously solving lots of little problems, then aggregating the solutions. Linear functions like the well-known OneMax problem have a maximal degree of separability. This makes them natural candidates for our investigations. Regardless of our intuition, however, it will turn out that separability alone is not sufficient to make the CCEA superior to the "stand-alone EA."

Another aspect that comes with the CCEA are increased explorative possibilities. Important EA parameters, like the mutation probability, are often defined depending on the string length, i.e., the dimension of the search space. For binary mutations, $1/n$ is most often recommended for strings of length n. Since the components have shorter length, an increased mutation probability is the consequence. This differs from increased mutation probabilities in a "stand-alone" EA in two ways. First, one can have different mutation probabilities for different components of the string with a CCEA in a natural way. Second, since mutation is done in the components separately, the CCEA can search in these components more efficiently, while the partitioning mechanism may afford the algorithm some added protection from the increased disruption. The components that are not

"active" are guaranteed not to be changed in that step. We present a class of example functions where this becomes very clear.

In the next section we give precise formal definitions of the (1+1) EA, the CC (1+1) EA, the notion of separability, and the notion of expected optimization time. In Section 3 we analyze the expected optimization time of the CC (1+1) EA on the class of linear functions and compare it with the expected optimization time of the (1+1) EA. Surprisingly, we will see that in spite of the total separability of linear functions the CC (1+1) EA has no advantage over the (1+1) EA. This leads us to concentrate on the effects of the increased mutation probability. In Section 4, we define a class of example functions, CLOB, and analyze the performance of the (1+1) EA and the CC (1+1) EA. We will see that the cooperative coevolutionary function optimization approach can reduce the expected optimization time from super-polynomial to polynomial or from polynomial to a polynomial of much smaller degree. In Section 5, we conclude with a short summary and a brief discussion of possible directions of future research.

2 Definitions and Framework

The (1+1) EA is an extremely simple evolutionary algorithm with population size 1, no crossover, standard bit-wise mutations, and plus-selection known from evolution strategies. Due to its simplicity it is an ideal subject for theoretical research. In fact, there is a wealth of known results regarding its expected optimization time on many different problems (Mühlenbein (6), Rudolph (9), Garnier, Kallel, Schoenauer (3), Droste, Jansen, and Wegener (2)). Since we are interested in a comparison of the performance of the EA alone as opposed to its use in the CCEA, known results and, even more importantly, known analytical tools and methods (Droste et. al. 1) are important aspects that make the (1+1) EA the ideal choice for us.

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

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

We do not care about finding an appropriate stopping criterion and let the algorithm run forever. In our analysis we are interested in the first point of time when $f(x)$ is maximal, i.e., a global maximum is found. As a measure of time we count the number of function evaluations.

For the CC (1+1) EA, we have to divide x into k components. For the sake of simplicity, we assume that x can be divided into k components of equal length

l, i. e., $l = n/k \in \mathbb{N}$. The generalization of our results to the case $n/k \notin \mathbb{N}$ with $k-1$ components of equal length $\lfloor n/k \rfloor$ and one longer component of length $n-(k-1)\cdot\lfloor n/k \rfloor$ is trivial. The k components are denoted as $x^{(1)},\ldots,x^{(k)}$ and we have $x^{(i)} = x_{(i-1)\cdot l+1}\cdots x_{i\cdot l}$ for each $i \in \{1,\ldots,k\}$. For the functions considered here, this is an appropriate way of distributing the bits to the k components.

Algorithm 2. (Cooperative Coevolutionary (1+1) EA (CC (1+1) EA))

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

We use $\min\{1/l, 1/2\}$ as mutation probability instead of $1/l$ in order to deal with the case $k = n$, i. e., $l = 1$. We consider $1/2$ to be an appropriate upper bound on the mutation probability. The idea of mutation is to create small random changes. A mutation probability of $1/2$ is already equivalent to pure random search. Indeed, larger mutation probabilities are against this basic "small random changes" idea of mutation. This can be better for some functions and is in fact superior for the functions considered here. Since this introduces annoying special cases that have hardly any practical relevance, we exclude this extreme case.

The CC (1+1) EA works with k independent (1+1) EAs. The i-th (1+1) EA operates on $x^{(i)}$ and creates the offspring $y^{(i)}$. For the purpose of selection the k strings $x^{(i)}$ are concatenated and the function value of this string is compared to the function value of the string that is obtained by replacing $x^{(a)}$ by $y^{(a)}$. The (1+1) EA with number a is called *active*. Again, we do not care about a stopping criterion and analyze the first point of time until the function value of a global maximum is evaluated. Here we also use the number of function evaluations as time measure.

Consistent with existing terminology in the literature (Potter and De Jong 8), we call one iteration of the CC (1+1) EA where one mutation and one selection step take place a *generation*. Note, that it takes k generations until each (1+1) EA was active once. Since this is an event of interest, we call k consecutive generations a *round*.

Definition 1. *Let the random variable T denote the number of function evaluations until for some $x \in \{0,1\}^n$ with $f(x) = \max\{f(x') \mid x' \in \{0,1\}^n\}$ the*

function value $f(x)$ is computed by the considered evolutionary algorithm. The expectation $E(T)$ is called expected optimization time.

When analyzing the expected run time of randomized algorithms, one finds bounds of this expected run time depending on the input size (Motwani and Raghavan 5). Most often, asymptotic bounds for growing input lengths are given. We adopt this perspective and use the dimension of the search space n as measure for the "input size." We use the well-known O, Ω, and Θ notions to express upper, lower, and matching upper and lower bounds for the expected optimization time.

Definition 2. *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.

As discussed in Section 1, an important property of pseudo-Boolean functions is separability. For the sake of clarity, we give a precise definition.

Definition 3. *Let $f\colon \{0,1\}^n \to \mathbb{R}$ be any pseudo-Boolean function. We say that f is* s-separable *if there exists a partition of $\{1, \ldots, n\}$ into disjoint sets $I_1, \ldots, I_r$, where $1 \leq r \leq n$, 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*

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

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

We say that f is exactly s-separable*, if f is s-separable but not $(s-1)$-separable.*

If a function f is known to be s-separable, it is possible to use the sets I_j for a division of x for the CC (1+1) EA. Then each (1+1) EA operates on a function g_j and the function value f is the sum of the g_j-values. If the decomposition into sub-problems is expected to be beneficial, it should be so if s is small and the decomposition matches the sets I_j. Obviously, the extreme case $s = 1$ corresponds to linear functions, where the function value is the weighted sum of the bits, i. e., $f(x) = w_0 + w_1 \cdot x_1 + \cdots + w_n \cdot x_n$ with $w_0, \ldots, w_n \in \mathbb{R}$. Therefore, we investigate the performance of the CC (1+1) EA on linear functions first.

3 Linear Functions

Linear functions, or 1-separable functions, are very simple functions. They can be optimized bit-wise without any interaction between different bits. It is easy to see that this can be done in $O(n)$ steps. An especially simple linear function

is OneMax, where the function value equals the number of ones in the bit-string. It is long known that the (1+1) EA has expected optimization time $\Theta(n \log n)$ on OneMax (Mühlenbein 6). The same bound holds for any linear function without zero weights, and the upper bound $O(n \log n)$ holds for any linear function (Droste, Jansen, and Wegener 2). We want to compare this with the expected optimization time of the CC (1+1) EA.

Theorem 1. *The expected optimization time of the CC (1+1) EA for a linear function $f\colon \{0,1\}^n \to \mathbb{R}$ with all non-zero weights is $\Omega(n \log n)$ regardless of the number of components k.*

Proof. According to our discussion we have $k \in \{1, \ldots, n\}$ with $n/k \in \mathbb{N}$. We denote the length of each component by $l := n/k$. First, we assume $k < n$. We consider $(n-k) \ln n$ generations of the CC (1+1) EA and look at the first (1+1) EA operating on the component $x^{(1)}$. This EA is active in each k-th generation. Thus, it is active in $((n-k) \ln n)/k = (l-1) \ln n$ of those generations. With probability 1/2, at least half of the bits need to flip at least once after random initialization. This is true since we assume that all weights are different from 0. Therefore, each bit has an unique optimal value, 1 for positive weights and 0 for negative weights. The probability that among $l/2$ bits there is at least one that has not flipped at all is bounded below by

$$1 - \left(1 - \left(1 - \frac{1}{l}\right)^{(l-1)\ln n}\right)^{l/2} \geq 1 - \left(1 - e^{-\ln n}\right)^{l/2} = 1 - \left(1 - \frac{1}{n}\right)^{l/2}$$
$$\geq 1 - e^{-1/(2k)} \geq 1 - \frac{1}{1 + 1/(2k)} = \frac{1}{2k+1} \geq \frac{1}{3k}.$$

Since the k (1+1) EA are independent, the probability that there is one that has not reached the optimum is bounded below by $1 - (1 - 1/(3k))^k \geq 1 - e^{-1/3}$. Thus, the expected optimization time of the CC (1+1) EA with $k < n$ on a linear function without zero weights is $\Omega(n \log n)$.

For $k = n$ we have n (1+1) EA with mutation probability 1/2 operating on one bit each. Each bit has an 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. On average, the number of coins that never came up head is halved in each round. It is easy to see that on average this requires $\Omega(\log n)$ rounds with all together $\Omega(n \log n)$ coin tosses. □

We see that the CC (1+1) EA has no advantage over the (1+1) EA at all on linear functions in spite of their total separability. This holds regardless of the number of components k. We conjecture that the expected optimization time is $\Theta(n \log n)$, i. e., asymptotically equal to the (1+1) EA. Since this leads away from our line of argumentation we do not investigate this conjecture here.

4 A Function Class with Tunable Advantage for the CC (1+1) EA

Recall that there were two aspects of the CC (1+1) EA framework that could lead to potential advantage over a (1+1) EA: partitioning of the problem and increased focus of the variation operators on the smaller components created by the partitioning. However, as we have just discussed, we now know that separability alone is not sufficient to make the cooperative coevolutionary optimization framework advantageous. Now we turn our attention to the second piece of the puzzle: increased explorative attention on the smaller components. More specifically, dividing the problem to be solved by separate (1+1) EAs results in an increased mutation probability in our case.

Let us consider one round of the CC (1+1) EA and compare this with k generations of the (1+1) EA. Remember that we use the number of function evaluations as measure for the optimization time. Note that both algorithms make the same number of function evaluations in the considered time period. We concentrate on $l = n/k$ bits that form one component in the CC (1+1) EA, e. g., the first l bits. In the CC (1+1) EA the (1+1) EA operating on these bits is active once in this round. The expected number of b bit mutations, i. e., mutations where exactly b bits in the bits $x_1, \ldots, x_l$ flip, equals $\binom{l}{b}\left(\frac{1}{l}\right)^b\left(1-\frac{1}{l}\right)^{l-b}$. For the (1+1) EA in one generation the expected number of b bit mutations in the bits $x_1, \ldots, x_l$ equals $\binom{l}{b}\left(\frac{1}{n}\right)^b\left(1-\frac{1}{n}\right)^{l-b}$. Thus, in one round, or k generations, the expected number of such b bit mutations equals $k \cdot \binom{l}{b}\left(\frac{1}{n}\right)^b\left(1-\frac{1}{n}\right)^{l-b}$. For $b = 1$ we have $\left(1-\frac{1}{l}\right)^{l-1}$ for the CC (1+1) EA and $\left(1-\frac{1}{n}\right)^{l-1}$ for the (1+1) EA which are similar values. For $b = 2$ we have $((l-1)/(2l))\left(1-\frac{1}{l}\right)^{l-2}$ for the CC (1+1) EA and $((l-1)/(2n))\left(1-\frac{1}{n}\right)^{l-2}$ for the (1+1) EA, which is approximately a factor $1/k$ smaller. For small b, i. e., for the most relevant cases, the expected number of b bit mutations is approximately a factor of k^{b-1} larger for the CC (1+1) EA than for the (1+1) EA. This may result in an huge advantage for the CC (1+1) EA.

In order to investigate this, we define an objective function, which is separable and requires b bit mutations in order to be optimized. Since we want results for general values of b, we define a class of functions with parameter b. We use the well-known LeadingOnes problem as inspiration (Rudolph 9).

Definition 4. *For $n \in \mathbb{N}$ and $b \in \{1, \ldots, n\}$ with $n/b \in \mathbb{N}$ we define the function $LOB_b\colon \{0,1\}^n \to \mathbb{R}$ (short for LeadingOnesBlocks) by*

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

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

LOB_b is identical to the so-called Royal Staircase function (van Nimwegen and Crutchfield 10) which was defined and used in a different context. Obviously,

the function value $\mathrm{LOB}_b(x)$ equals the number of consecutive blocks of length b with all bits set to one (scanning x from left to right). Consider the (1+1) EA operating on LOB_b. After random initialization the bits have random values and all bits right of the left most bit with value 0 remain random (see Droste, Jansen, and Wegener 2 for a thorough discussion). Therefore, it is not at all clear that b bit mutations are needed. Moreover, LOB_b is not separable, i. e., it is exactly n-separable. We resolve both issues by embedding LOB_b in another function definition. The difficulty with respect to the random bits is resolved by taking a leading ones block of a higher value and subtracting OneMax in order to force the bits right of the left most zero bit to become zero bits. We achieve separability by concatenating k independent copies of such functions, which is a well-known technique to generate functions with a controllable degree of separability.

Definition 5. *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 $CLOB_{b,k}\colon \{0,1\}^n \to \mathbb{R}$ (short for Concatenated LOB) by*

$$CLOB_{b,k}(x) := \left(\sum_{h=1}^{k} n \cdot LOB_b \left(x_{(h-1)\cdot l+1} \cdots x_{h\cdot l} \right) \right) - OneMax(x)$$

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

We have k independent functions, the i-th function operates on the bits $x_{(i-1)\cdot l+1} \cdots x_{i\cdot l}$. For each of these functions the function value equals n times the number of consecutive leading ones blocks (where b is the size of each block) minus the number of one bits in all its bit positions. The function value $\mathrm{CLOB}_{b,k}$ is simply the sum of all these function values.

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 $b > 1$. We always consider the case where the CC (1+1) EA makes complete use of the separability of $\mathrm{CLOB}_{b,k}$. Therefore, the number of components or sub-populations equals the function parameter k. In order to avoid technical difficulties we restrict ourselves to values of k with $k \leq n/4$. This excludes the case $k = n/2$ only, since $k = n$ is only possible with $b = 1$. We start our investigations with an upper bound on the expected optimization time of the CC (1+1) EA.

Theorem 2. *The expected optimization time of the CC (1+1) EA on the function $CLOB_{b,k}\colon \{0,1\}^n \to \mathbb{R}$ is $O\left(kl^b \left(\frac{l}{b} + \ln k\right)\right)$ with $l := n/k$, where the number of components of the CC (1+1) EA is k, and $2 \leq b \leq n/k$, $1 \leq k \leq n/4$, and $n/(bk) \in \mathbb{N}$ hold.*

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

We 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 function

value by at least $n-b$. After at most l/b such mutations $x^{(1)} = 1^l$ holds. The probability of such a mutation is bounded below by $(1/l)^b(1-1/l)^{l-b} \geq 1/(el^b)$. We consider $k \cdot 10e \cdot l^b((l/b) + \ln k)$ generations. The first (1+1) EA is active in $10e \cdot l^b((l/b) + \ln k)$ generations. The expected number of such mutations is bounded below by $10((l/b)+\ln k)$. Chernoff bounds yield that the probability not to have at least $(l/b) + \ln k$ such mutations is bounded above by $e^{-4((l/b)+\ln k)} \leq \min\{e^{-4}, k^{-4}\}$. In the case $k = 1$, this immediately implies the claimed bound on the expected optimization time. Otherwise, the probability that there is a component different from 1^l is bounded above by $k \cdot (1/k^4) = 1/k^3$. This again implies the claimed upper bound and completes the proof. □

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

Theorem 3. *The expected optimization time of the (1+1) EA on the function* $CLOB_{b,k}\colon \{0,1\}^n \to \mathbb{R}$ *is* $\Omega\left(n^b\left(n/(bk) + \ln k\right)\right)$, *if* $2 \leq b \leq n/k$, $1 \leq k \leq n/4$, *and* $n/(bk) \in \mathbb{N}$ *holds.*

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

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

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

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

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

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

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

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

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

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

$$\frac{\Omega\left(n^b \cdot \left(\frac{n}{bk} + \log n \right) \right)}{O\left(l^b \left((l/b) + \log k \right) \right)} = \Omega\left(k^{b-1} \right)$$

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

It should be clear that simply increasing the mutation probability in the (1+1) EA will not resolve the difference. Increased mutation probabilities lead to a larger number of steps where the offspring y does not replace its parents x, since the number of leading ones blocks is decreased due to mutations. 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.

5 Conclusion

We investigated a quite general cooperative coevolutionary function optimization framework that was introduced by Potter and De Jong (7). One feature of this framework is that it can be instantiated using any evolutionary algorithm as underlying search heuristic. We used the well-known (1+1) EA and presented the CC (1+1) EA, an extremely simple cooperative coevolutionary algorithm. The main advantage of the (1+1) EA is the multitude of known results and powerful analytical tools. This enabled us to present the run time or optimization time analysis for a coevolutionary algorithm. To our knowledge, this is the first such analysis of coevolution published. The focus of our investigation was on separability. Indeed, when applying the Potter and De Jong 7 cooperative coevolutionary approach, practitioners make implicit assumptions about the separability of the function in order to come up with appropriate divisions of the problem space. Given such a static partition of a string into components, the CCEA is expected to exploit the separability of the problem and to gain an advantage over the employed EA when used alone. We were able to prove that separability alone is not sufficient to give the CCEA any advantage. We compared the expected optimization time of the (1+1) EA with that of the CC (1+1) EA on linear functions that are of maximal separability. We found that the CC (1+1) EA is not faster. Motivated by this finding we discussed the expected frequency of mutations for both algorithms. The main point is that b bit mutations occur noticeably more often for the CC (1+1) EA for $b > 1$ only. The expected frequency of mutations changing only one single bit is asymptotically the same for both algorithms. This leads to the definition of $\text{CLOB}_{b,k}$, a family of separable functions where b bit mutations are needed for successful optimization. For this family of functions we were able to prove that the cooperative coevolutionary approach leads to an immense speed-up. The advantage of the CC (1+1) EA over the (1+1) EA can be of super-polynomial order. Moreover, this advantage stems not only from the ability of the CC (1+1) EA to partition the problem, but because coevolution can use this partitioning to concentrate increased variation on smaller parts of the problem.

Our results are a first and important step towards a clearer understanding of coevolutionary algorithms. But there are a lot of open problems. An upper bound for the expected optimization time of the CC (1+1) EA on linear functions

needs to be proven. Using standard arguments the bound $O(n \log^2 n)$ is easy to show; however, we conjecture that the actual expected optimization time is $O(n \log n)$ for any linear function and $\Theta(n \log n)$ for linear functions without zero weights. For $\text{CLOB}_{b,k}$ we provided neither a lower bound proof of the expected optimization time of the CC (1+1) EA nor an upper bound proof of the expected optimization time of the (1+1) EA. A lower bound for the CC (1+1) EA that is asymptotically tight is not difficult to prove. A good upper bound for the (1+1) EA is slightly more difficult.

Furthermore, it is obviously desirable to have more comparisons for more general parameter settings and other objective functions. The systematic investigation of the effects of running the CC (1+1) EA with partitions into components that do not match the separability of the objective function is also the subject of future research. A main point of interest is the analysis of other cooperative coevolutionary algorithms where more complex EAs that use a population and crossover are employed as underlying search heuristics. The investigation of such CCEAs that are more realistic leads to new, interesting, and much more challenging problems for future research.

References

S. Droste, T. Jansen, G. Rudolph, H.-P. Schwefel, K. Tinnefeld, and I. Wegener (2003). Theory of evolutionary algorithms and genetic programming. In H.-P. Schwefel, I. Wegener, and K. Weinert (Eds.), *Advances in Computational Intelligence*, Berlin, Germany, 107–144. Springer.

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

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

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

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

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

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

M. A. Potter and K. A. De Jong (2002). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation* 8(1), 1–29.

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

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