Bridging the Gap Between Theory and Practice

Thomas Jansen[1] and R. Paul Wiegand[2,⋆]

[1] FB Informatik, LS 2, Universität Dortmund, Germany, Thomas.Jansen@udo.edu
[2] Department of Computer Science, George Mason University, Fairfax, VA 22030, USA
paul@tesseract.org

Abstract. While the gap between theory and practice is slowly closing, the evolutionary computation community needs to concentrate more heavily on the middle ground. This paper defends the position that contemporary analytical tools facilitate such a concentration. Empirical research can be improved by considering modern analytical techniques in experimental design. In addition, formal analytical extensions of empirical works are possible. We justify our position by way of a constructive example: we consider a recent empirically-based research paper and extend it using modern techniques of asymptotic analysis of run time performance of the algorithms and problems investigated in that paper. The result is a more general understanding of the performance of these algorithms for any size of input, as well as a better understanding of the underlying reasons for some of the previous results. Moreover, our example points out how important it is that empirical researchers motivate their parameter choices more clearly. We believe that providing theorists with empirical studies that are well-suited for formal analysis will help bridge the gap between theory and practice, benefitting the empiricist, the theorist, and the community at large.

1 Introduction

Historically, research in evolutionary computation (EC) has been replete with innovative refinements and augmentations to evolutionary algorithms. Analysis of improved evolutionary algorithms (EAs) has been sparse and primarily confined to empirical methods. There is a wide gap between EC theory and practice. Practitioners accuse theorists of analyzing simple algorithms and simple problems, impractical for real use and describing very little of value for real applications. Theorists are frustrated by seemingly arbitrarily engineered and poorly motivated algorithms that are difficult, if not impossible to analyze thoroughly.

Fortunately, the gap between these two sides can be made smaller. To bridge it, more attention on the middle ground between theoreticians and engineers is required. We need empiricists who are driven and guided by theory to understand the performance of these algorithms at a more fundamental level. As theorists strive to provide more advanced and applicable analytical methods, empiricists should strive to develop experimental frameworks that provide a more fertile ground for future theoretical development, motivate their algorithmic and problem choices more carefully and thoroughly, and make use of existing theoretical results to guide and inform their research.

One way to bridge the gap is to extend empirical analysis with theoretical results. Though experimental analysis can be carried out for any EA on almost any problem,

⋆ R. Paul Wiegand currently holds a postdoctoral position with the American Society for Engineering Education and conducts research at the Naval Research Laboratory.

only a limited number of parameter settings can be considered. This makes it impossible to say how the performance of the algorithm scales with the size of the problem beyond the experiments actually carried out. Extending empirical research with theoretical analysis can help fill in these gaps, and provide a much stronger grasp of the "how's" and "why's" of EA performance.

Recent advances in asymptotic analyses of evolutionary algorithms has made the idea of bridging the theory-practice gap more feasible. Droste, Jansen, and Wegener [1] presented various analytical tools developed for the extremely simple (1+1) EA, a kind of randomized mutation hill-climber. Some of these methods can be extended to the use of a larger offspring population size [3]. There are also results for mutation-based evolutionary algorithms with a population size $\mu > 1$ [2, 4, 14]. Witt [15] even presented a method for proving lower bounds on the expected optimization time, which is often much harder. Additionally, though EAs using crossover are considerably more difficult to analyze [11], results on the expected running time are known for some steady-state GAs [5, 6, 12]. These results are encouraging, though admittedly contain only upper bounds; there are currently no known methods for proving lower bounds on the expected optimization time of EAs with crossover.

This paper states a firm position: not only can theory be useful for practical research, but well-designed empirical studies can both make use of theoretical results, and be extended by theoretical research. We justify this position using a constructive example taken directly from recent literature. We consider an empirical paper by Wiegand, Liles, and De Jong [13] that concentrates on the performance of a particular design aspect of a specific cooperative coevolutionary algorithm (CCEA). We provide theoretical results for the expected optimization time of this real algorithm. Our analysis is incomplete in the sense that we do not consider every experiment discussed in [13]; however, we are able to provide a surprising amount of information with contemporary analytical techniques. This shows how experimental results can be validated and generalized for larger search spaces. More generally, we uncover some interesting difficulties that are based on the specifics of empirical research and suggest changes in the presentation of empirical studies that eases the continuation of empirical research by theoretical studies.

2 An Empirical Analysis of Cooperative Coevolution

A common extension to evolutionary algorithms are coevolutionary algorithms in which fitness is a function of interactions between individuals. One common subset of such algorithms are so-called "cooperative" coevolutionary algorithms, where individuals collaborate to form complete solutions that are mutually beneficial. A very general framework for applying cooperative coevolutionary algorithms to many types of optimization problems was provided by Potter and De Jong [10]. In this framework there are multiple populations, representing problem components, and each population is evolved more or less independently with the exception of evaluation. The framework is appealing since it places no demands on the underlying EAs that are used for each population, nor on any particular kind of representation.

During evaluation, representatives (so called collaborators) from the collaborating populations must be selected to form complete solutions in order to obtain fitness. How these collaborators are chosen, and what problem properties affect this choice is a challenging issue. It has lead to a variety of work, almost all of which has been purely empirical in nature. Two representative such works are work by Bull [16] and Wiegand, Liles and De Jong [13]. We consider the latter, more recent empirical research for our example. Before we can begin, we will need to define the problems under study, as well as the specific CCEA researched.

2.1 Problem Domains for Analysis

Wiegand, Liles, and De Jong [13] consider three maximization problems in their analysis. These problems can be desribed by pseudo-Boolean functions that map bit strings of length n map to real numbers, $f\colon \{0,1\}^n \to \mathbb{R}$. All three functions are based on the well-known LEADINGONES and ONEMAX problems, investigating the influence of decomposition of the bit string on collaboration. We consider exactly the same three functions, though use a different notation that fits better within the framework of asymptotic analysis. The relation to the original notation from [13] is made explicit.

Definition 1. *For $n' \in \mathbb{N}$ let $n = 4n'$. The function $f_1\colon \{0,1\}^n \to \mathbb{R}$ is defined by*

$$f_1(x) := \left(\sum_{i=1}^{n/2} \prod_{j=1}^{i} x_j\right) + \left(\sum_{i=(n/2)+1}^{n} \prod_{j=1}^{i} x_j\right).$$

The function $f_2\colon \{0,1\}^n \to \mathbb{R}$ is defined by $f_2(x) := n \cdot f_1(x) - \text{ONEMAX}(x)$. The function $f_3\colon \{0,1\}^n \to \mathbb{R}$ is defined by

$$f_3(x) := n \cdot \left(\left(\sum_{i=1}^{n/4} \prod_{j=1}^{i} x_{2j-1}x_{2j}\right) + \left(\sum_{i=(n/4)+1}^{n/2} \prod_{j=1}^{i} x_{2j-1}x_{2j}\right)\right) - \text{ONEMAX}(x).$$

In [13] f_1 is called concatenated LEADINGONES. It can be described as $\text{LEADINGONES}(x_1 \cdots x_{n/2}) + \text{LEADINGONES}(x_{(n/2)+1} \cdots x_n)$. The function f_2 is called LEADINGONES − ONEMAX. The definition of f_3 corrects a misprint in [13] and is in accordance with the function used for experiments there. It is identical to the function $\text{CLOB}_{2,2}$ from [9].

2.2 A Cooperative Coevolutionary Algorithm

Wiegand, Liles, and De Jong [13] present an instantiation that uses steady-state GAs with rank-based selection, replacement of the worst, standard bit-flip mutation, and parameterized uniform crossover. In the coevolutionary framework, the strings are partitioned into components and each population is assigned the task of representing one such component. This suggests two choices: how many components (populations) will be used, and how will the bit string ($x = x_1x_2 \cdots x_n \in \{0,1\}^n$) be decomposed among these? We use the notation k to denote the number of components (the previous study used p), and l to denote the length of a component. The pseudo-Boolean nature of the fitness functions suggests some obvious decompositions, but without intimate knowledge of the function itself engineers cannot know the most appropriate decomposition *a priori*. Two decompositions explored by the previous study are defined below.

Definition 2. *Let $l, k \in \mathbb{N}, n = lk$ be given. A bit string $x = x_1 x_2 \cdots x_n \in \{0,1\}^n$ is divided into k components $x^{(1)}$, $x^{(2)}$, …, $x^{(k)}$ of equal length $l = n/k$. Regardless of the specific decomposition we write $x^{(1)} x^{(2)} \cdots x^{(k)}$ to denote the complete bit string x in its original bit ordering.*

We call this a direct decomposition *if $x^{(i)} = x_{(i-1)l+1} x_{(i-1)l+2} \cdots x_{il}$ holds for all $i \in \{1, \ldots, k\}$. We call this a* distributed decomposition *if $x^{(i)} = x_i x_{i+k} \cdots x_{i+(l-1)k}$ holds for all $i \in \{1, \ldots, k\}$.*

More precisely, we always consider the case of a direct or distributed decomposition of x into k components. There are k identical EAs, each operating on one component $x^{(i)}$. The initial population is chosen uniformly at random. The main algorithm is working in rounds, where in each round each of these k EAs is active once, i. e., each does one generation. The ordering is always $x^{(1)}$, $x^{(2)}$, …, $x^{(k)}$. We consider the algorithm without stopping criterion and are interested in the first point of time when some complete bit string x is optimal under the fitness function f. This point is subtle: An optimal bit string may be constructible from the components in the k populations. However, we ask more: We require the CCEA to realize this by actually assembling such an optimal bit string for a function evaluation. We call the total number of function evaluations at this point of time the optimization time T and derive upper bounds on $\mathrm{E}(T)$.

The steady-state GA used as underlying search heuristic in [13] has population size μ. In each generation, first two parents are selected according to the same probability distribution. Selection is done rank-based: the individuals are sorted with respect to increasing fitness values and the position within this sorted list becomes the rank. For each individual, the probability to be selected is proportional to its rank. With some probability p_c we apply parameterized uniform crossover with probability 0.2. If no crossover is performed, we copy the first parent to the offspring. In any case, standard bit-flip mutation with mutation probability $1/l$ is applied. Finally, randomly one of the individuals of the current population with minimal fitness is replaced by the offspring.

We assign a fitness value to some component $x^{(i)}$ in the following way. We select one member of the population from each EA. This can be either done uniformly at random from the whole population ($b = 0$) or uniformly at random from the individuals with maximal fitness ($b = 1$). We then assemble the complete bit string and compute the fitness value. After doing this for c times, the maximum value of the objective function values obtained is assigned as fitness to the component at hand.

We denote one such GA as GA(μ, p_c, c, b). For b we consider both values 0 and 1. Like [13] we study $c \in \{1, 2, 3\}$. For the crossover probability p_c, we are interested in the cases $p_c = 1$ (as used in [13]) and $p_c \in [\varepsilon; 1 - \varepsilon]$, where $0 < \varepsilon \leq 1/2$ is a constant. We do not fix a certain value for μ, but do our analyses for arbitrary population sizes. The reason for this is discussed below.

2.3 Poorly Motivated Empirical Choices

Empirical researchers have a variety of reasons for selecting parameter values. These include experience and intuition, but also include underlying demonstrative reasons. Unfortunately, when researchers do not state their reasoning further analysis can become difficult. The choice for μ in [13] is exactly such a parameter.

The previous study uses a constant $\mu = 10$, regardless of n and k. In that case, they fix $n = 128$ as a constant, but we are now interested in generalizing their results by looking at the asymptotic behavior of the expected optimization time $\mathrm{E}\,(T)$ for growing n. We might speculate that perhaps $\mu = \sqrt{n}$ is close to the value intended; but with different values of k ($k \in \{2, 4, 8, 16\}$ being used in [13]) one may argue that even $\mu = l$ would be plausible. Regardless, it is doubtful that choosing μ independent of n, i.e. $\mu = O(1)$, would be appropriate in our asymptotic setting. We avoid this by choosing not to restrict the size of μ.

Ideally, an extension such as ours should get information directly from the empirical study, but this is not always possible. Since any empirical study is necessarily restricted to a limited number of concrete parameter settings, it is tempting just to report the actual settings used. However, it is reasonable to expect some justification for these. The example of μ here makes it clear that more is needed in order to support further theoretical research: some functional correlation between the different parameters of the algorithmic system is required of the empiricist.

3 Theoretical Analyses of the CCEA

It is helpful to know something about the performance of the EA used as the underlying search heuristic in our CCEA. Therefore, we consider GA(μ, p_c, c, b) an "ordinary" GA with no decomposition. We use the notation GA(μ, p_c) since the values of b and c are only used when the GA is part of the CCEA and have no meaning here.

Theorem 1. *Let ε with $0 < \varepsilon \leq 1/2$ be some constant, $p_c \in [\varepsilon; 1-\varepsilon]$. The expected optimization time of GA(μ, p_c) on f_1 and f_2 is $O(n^2 + \mu n \ln n)$. The expected optimization time of GA(μ, p_c) on f_3 is $O(n^3 + \mu n^2 \ln n)$.*

Proof. We use a generalization of the method of fitness-based partitions [1]. Let b be the fitness of the current best individual and let n_b be the number of individuals with that fitness. Let T_i be the random variable denoting the number of function evaluations until b increased its value from i to at least $i + 1$ for f_1 or from at least $n \cdot i$ to at least $n \cdot (i + 1)$ for f_2 (with $i \in \{0, 1, \ldots, n - 1\}$). If there is never a current best individual with fitness i, we say $T_i = 0$. Obviously, $\mathrm{E}\,(T_0) + \mathrm{E}\,(T_1) + \cdots + \mathrm{E}\,(T_{n-1})$ is the expected optimization time.

For f_1 and f_2 we can use the same argument. We distinguish two cases with respect to the population size μ. First, assume $\mu < n/\ln n$ holds. As long as $n_b < \mu$ holds, n_b can be increased if the GA selects a current best string as first parent (probability at least n_b/μ), does no crossover (probability $1 - p_c$), and does not flip any bit (probability $(1 - 1/n)^n$). This happens with probability at least $(n_b/\mu) \cdot (1 - p_c) \cdot (1 - 1/n)^n = \Omega(n_b/\mu)$. The expected waiting time for this event is bounded above by $O(\mu/n_b)$. Of course, n_b can take any value from $\{1, \ldots, \mu\}$ at most once. Thus, we have $O(\mu \ln \mu) = O(n)$ as upper bound on the expected waiting time until either $n_b = \mu$, or b has been increased. In the case $n_b = \mu$, we can increase b be selecting any string, doing no crossover and mutating exactly the left-most zero-bit. Such an event has probability $\Omega(1/n)$. Altogether, this yields $\mathrm{E}\,(T_i) = O(n)$ for all i. We have $O(n^2)$ as upper bound on the expected optimization time in this case. Considering the case $\mu \geq n/\ln n$, we repeat the same arguments, waiting only until $n_b \geq n/\ln n$ holds. The expected waiting time for this is $O(\mu \ln n)$. Then the probability to increase b is bounded below by $\Omega((n/(\mu \ln n)) \cdot (1/n)) = \Omega(1/(\mu \ln n))$. This implies $\mathrm{E}\,(T_i) = O(\mu \ln n)$ for all i, and we have $O(\mu n \ln n)$ as upper bound on the expected optimization time in this case.

For f_3 we can use the same arguments. We may need a mutation of two bits in order to increase b. This increases all time bounds by a factor of n. □

There is no asymptotic difference in the upper bounds for f_1 and f_2. We do not have lower bounds on $\mathrm{E}\,(T)$ since there is currently no lower bound technique known for EAs with crossover. We conjecture, however, that the similarities in the upper bound proofs reflect some common problem structure. In particular, we believe that f_2 will be slightly harder to optimize than f_1. Observe that for f_1 the bits to the right of the left-most zero-bit are all random [1]. Thus, we get about half of the bits “for free.” Therefore, we speculate the observed optimization time for f_2 will be a factor of 2 longer.

The proof of Theorem 1 concentrates on the generations without crossover. Due to the steady-state replacement, generations with crossover can do no harm. If we consider GA$(\mu, 1)$, this proof technique cannot work, leading us to upper bounds that are larger by the factor $n^{2/3}/\ln n$. We do not claim that these upper bounds are sharp.

Theorem 2. *The expected optimization time of GA$(\mu, 1)$ on f_1 and f_2 is $O(n^2 + \mu n^{5/3})$. The expected optimization time of GA$(\mu, 1)$ on f_3 is $O(n^3 + \mu n^{8/3})$.*

Proof. We use the same proof technique as above. However, since we have to cope with crossover here, we concentrate on different events. Observe that the result of crossover of two bit strings with m_1 and m_2 bits set to 1 yields a bit string with at least $\min\{m_1, m_2\}$ bits set to 1 with probability $\Omega(1)$. This motivates concentration on the improvement of the worst members of the population. We estimate the time it takes to increase the function value of up to μ worst members of the population by at least 1. Doing this at most n times leaves us with at least one optimal string in the population. We can assume that non-worst individuals in the current population are improvements from the set of worst members of a previous population.

We begin with f_1 and f_2. First, assume $\mu \leq n^{1/3}$ holds. If all members of the population have the same number of leading ones, then we can increase this number for one individual by first selecting two arbitrary parents and doing crossover. With probability $\Omega(1)$ we are left with a bit string with a number of leading ones that is at least as large as its parents. Then flipping exactly the left-most bit with value 0 increases the function value. Thus, such a generation has probability $\Omega(1/n)$ and the expected waiting time is $O(n)$. If we have at least one bit string with a larger function value than the current worst, we can produce copies of this individual in the following way. We select such a current best individual as two parents (with probability $\Omega(1/\mu^2)$) and produce an offspring with a number of leadings ones that is not smaller via crossover (with probability $\Omega(1)$). Flipping no bit in mutation (with probability $\Omega(1)$) leaves us with an individual that has at least as many leading ones. For f_2 the function value may be smaller but we do not care. On average, after $O(\mu^3)$ such generations the population consists of individuals that have all the same number of leading ones, and we start over. We have do this at most n times, leading to an upper bound of $O(n^2 + n \cdot \mu^3) = O(n^2)$ for the expected optimization time in this case. In the case $\mu > n^{1/3}$ our proof is similar.

Now we increase the number of leading ones in current worst individuals $\mu/n^{1/3}$ times. This increases the probability for copying one individual with a maximal number of leading ones to $\Omega(1/n^{2/3})$, leading to $O(n \cdot (n\mu)/n^{1/3} + n \cdot \mu \cdot n^{2/3}) = O(\mu n^{5/3})$.

For f_3, the proofs work in the same way but concentrate on pairs of leading ones. We distinguish the cases $\mu \leq n^{2/3}$ and $\mu > n^{2/3}$. In the latter case we increase the number of leading pairs of ones for a worst individual $\mu/n^{2/3}$ times. □

Concentrating on the worst members of the population leads to a larger upper bound that can be proven. It is not at all clear that our pessimistic analysis of the effects of crossover is realistic; however, we use the results from Theorem 1 and Theorem 2 to derive upper bounds on the expected optimization time of the cooperative coevolutionary algorithm and compare our findings with the empirical data from [13].

We begin the investigation for the case $b = 1$, i. e., a current best member of a population is selected as collaborator. This case is easier to analyze.

Theorem 3. *Let $l' \geq 1$ and $k > 1$ be two integers, $l := 2l'$, $n := l \cdot k$. We consider f_1, f_2, and f_3 on n bits with a direct decomposition into k components. Let ε with $0 < \varepsilon \leq 1/2$ be a constant, $p_c \in [\varepsilon; 1-\varepsilon]$, $c \in \{1,2,3\}$.*

The expected optimization time of GA$(\mu, p_c, c, 1)$ is $O(n^2 + \mu nk \log l)$ on f_1 and f_2. It is $O(n^2 l + \mu n^2 \log l)$ on f_3.

The expected optimization time of GA$(\mu, 1, c, 1)$ is $O(n^2 + \mu n^2/l^{1/3})$ on f_1 and f_2. It is $O(n^2 l + \mu n^2 l^{2/3})$ on f_3.

Proof. We start with the proof for GA$(\mu, p_c, c, 1)$ on f_1 and f_2. We consider the algorithm in rounds, where in each round each GA performs exactly one generation. Due to the steady-state selection it suffices to concentrate on improving steps for an upper bound. After an initial phase with $\Theta(\mu k)$ function evaluations, in each GA at least one sub-string that can currently be assembled into a string where it is part of the leading ones is identified if it does exist in the population. Then we concentrate in each round on a GA that contains a left-most 0 in such an assembled string. Since we have $b = 1$, the “correct” string is assembled in each round. We know from Theorem 1 that on average after $O(l^2 + \mu l \log l)$ generations this string has become all ones. We can conclude that on average after $O(kl^2 + \mu kl \log l) = O(nl + \mu n \log l)$ function evaluations this happens. This happens with probability very close to 1, and we have to wait for k GAs. This yields $O(n^2 + \mu nk \log l)$ as upper bound.

The proofs for the other statements can be done in the same way. □

Theorem 4. *Let $l' \geq 1$ and $k > 1$ be two integers, $l := 2l'$, $n := l \cdot k$. We consider f_1, f_2, and f_3 on n bits with a distributed decomposition into k components. Let ε with $0 < \varepsilon \leq 1/2$ be a constant, $p_c \in [\varepsilon; 1-\varepsilon]$, $c \in \{1,2,3\}$, $b \in \{0,1\}$.*

The expected optimization time of GA$(\mu, p_c, c, 1)$ is $O(n^2 + \mu nk \log l)$ on f_1 and f_2. The GA$(\mu, p_c, c, 1)$ has no finite expected optimization time on f_3.

The expected optimization time of GA$(\mu, 1, c, 1)$ is $O(n^2 + \mu n^2/l^{1/3})$ on f_1 and f_2. The GA$(\mu, 1, c, 1)$ has no finite expected optimization time on f_3.

Proof. For f_1 and f_2, in the proof of Theorem 3 we concentrated on mutations of single bits. With respect to these mutations, the kind of decomposition used has no effect. Thus the same proofs are valid and the same upper bounds hold.

This is not true for f_3. There mutations of two bits may be needed: If we have $1^{2i}00\dots$ at some point of time, changing the $(2i+1)$-th or $(2i+2)$-th bit to 1 alone will decrease the function value. Such an offspring will be inserted into the population. However, it cannot have the best function value and therefore will not be selected in the following generations as collaborator (since we have $b=1$). Since this is already true after random initialization with positive probability, there is a non-zero probability that the global optimum will never be reached. Thus there is no finite expected optimization time. This holds regardless of the crossover probability p_c. $\square$

When the collaborators are chosen randomly ($b=0$), the performance is more difficult to analyze. We present much weaker upper bounds. Since we have no lower bounds, this does not prove that choosing collaborators randomly is worse for f_1, f_2, and f_3. In fact, in some ways it is even better for f_3.

Theorem 5. *Let $l' \geq 1$ and $k > 1$ be two integers, $l := 2l'$, $n := l \cdot k$. We consider f_1, f_2, and f_3 on n bits with a direct decomposition into k components. Let ε with $0 < \varepsilon \leq 1/2$ be a constant, $p_c \in [\varepsilon; 1-\varepsilon]$, $c \in \{1,2,3\}$.*

The expected optimization time of GA$(\mu, p_c, c, 0)$ is $O(\mu^{k-1}n^2 + \mu^k nk \log l)$ on f_1 and f_2. It is $O(\mu^{k-1}n^2 l + \mu^k n^2 \log l)$ on f_3.

The expected optimization time of GA$(\mu, 1, c, 0)$ is $O(\mu^{k-1}n^2 + \mu^k n^2/l^{1/3})$ on f_1 and f_2. It is $O(\mu^{k-1}n^2 l + \mu^k n^2 l^{2/3})$ on f_3.

Proof. To yield useful information, function values that are computed must be based on components carrying leading ones. Those are the best of their population and are guaranteed to become collaborators for $b=1$. With $b=0$ they are only selected with a small probability. We make a rough and pessimistic approximation by assuming that in each component there is only one "good" collaborator chosen with probability $1/\mu$. This can be improved to almost c/μ. It introduces an additional factor μ^{k-1} to the expected waiting time. The bounds follow directly from Theorem 3. $\square$

Theorem 6. *Let $l' \geq 1$ and $k > 1$ be two integers, $l := 2l'$, $n := l \cdot k$. We consider f_1, f_2, and f_3 on n bits with a distributed decomposition into k components. Let ε with $0 < \varepsilon \leq 1/2$ be a constant, $p_c \in [\varepsilon; 1-\varepsilon]$, $c \in \{1,2,3\}$, $b \in \{0,1\}$.*

The expected optimization time of GA$(\mu, p_c, c, 0)$ is $O(\mu^{k-1}n^2 + \mu^k nk \log l)$ on f_1 and f_2. The expected optimization time of GA$(\mu, p_c, c, 0)$ is $O(\mu^{k-1}n^2 l + \mu^k n^2 \log l)$ on f_3. The expected optimization time of GA$(\mu, 1, c, 0)$ is $O(\mu^{k-1}n^2 + \mu^k n^2/l^{1/3})$ on f_1 and f_2 and it is $O(\mu^{k-1}n^2 l + \mu^k n^2 l^{2/3})$ on f_3.

Proof. The proof for f_1 and f_2 follow from the Theorem 4 proof in the same way as the Theorem 5 proof follows from the Theorem 3 proof, except that things now change for f_3. As explained above, setting single bits of a pair to 1 decreases the fitness. However, such strings are generated and inserted with a positive and not too small probability. After doing so, a bit string with such a pair of ones may be assembled due to the random choice of the collaborators. Then the increase in fitness is realized and the new pair of ones is guaranteed to remain. However, the two "lucky one bit mutations" must occur within a short time span. Otherwise, a component with a single one may be removed. We model this by concentrating on the case that these two mutation of single bits happen in a direct sequence. This has asymptotically the same probability as the joined mutation of a pair. Thus we again obtain the bounds from Theorem 5. □

Our asymptotic results are obviously not strong enough to cover all aspects of the research presented in [13], but the main observations are the same. Note that the lack of lower bounds leaves room for speculations that the true asymptotic expected optimization times may be different. But this is not supported by the empirical data in [12]. Our asymptotic analysis suggests no performance difference between f_1 and f_2, and we argue that the expected optimization time most likely differs by a factor of at most 2. This is confirmed by the experimental data from [13]. They saw no differences due to the number of collaborators. This had to be expected since the number of collaborators is restricted to a small constant. We conjecture that a number of collaborators that grows with n will have an effect that is visible within our asymptotic framework. This is supported by the data in [13]. For f_3 we can prove that with the distributed decomposition and $b = 1$ the algorithms fail to locate the global optimum. However, for the direct decomposition or $b = 0$ we can prove a polynomial upper bound on the expected optimization time. This appears to contradict that findings in [13], where f_3 seems to be very hard in any case. However, the upper bounds on the expected optimization time for f_3 are significantly larger than for f_1 and f_2. We believe that given more time the algorithms with $b = 0$ or using the direct decomposition would succeed on f_3.

4 Conclusions

Analytical extensions to experimental studies are possible. The mathematical methods for formal studies of algorithmic performance continue to improve so that increasingly more realistic algorithms may be considered. Bridging the gap between theory and practice is more attainable now than ever, but some work on the middle ground is needed by both theoreticians and empiricists. As theoretical work seeks to provide ever more flexible analytical methods, experimental scientists should begin to use existing theoretical results to guide experimental design, and offer studies that facilitate analysis by recognizing the current state of theory and motivating parameter values more clearly.

As a case in point, this paper presents an example of a theoretical extension of existing empirical study on the very same problems and algorithms used in that study [13]. Our results not only validate those discovered by experimentation, but also *generalize* them for increased search space size. Moreover, the proofs provided here offer more insights and help us understand why certain effects can be observed. The theoretical results are less concrete than the experimental results, but together they offer a much more comprehensive picture of the run time performance.

We encountered a number of difficulties when transferring the experimental setup to our analytical framework. In an asymptotic analysis one investigates the expected optimization time of an evolutionary algorithm for growing search space dimension. Meaningful results can be obtained when parameters of the algorithm are expressed as functions of the size of the search space, but like almost all empirical studies, this is not done in [13]. This requires a careful interpretation of the chosen setup and translation into such functions. High quality empirical research should clearly present justifications for specific algorithm design choices and problem properties. They should be designed as simply as possible, while still demonstrating their points effectively, and they should consider existing theoretical results and methods while designing experiments. Only then can the work between theory and practice truly begin.

Experimental results are intrinsically restricted to the problem sizes and parameter settings that have been actually used. In order to come to results that allow meaningful statements about the expected behavior for increasing problem sizes, a theoretical analysis is needed. Asymptotic analyses are most useful in this situation. They allow for some simplifications during in the proof that makes the analysis tractable. But they still deliver rigorously proven results without unproven assumptions or simplifications with unknown consequences. Theoretical analysis can and should become an increasingly important part of research in the field of evolutionary computation.

References

1. S. Droste, T. Jansen, I. Wegener (2002): On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science 276:51–81.
2. J. He, X. Yao (2002): From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms. IEEE Trans. Evolutionary Computation 6(5):495-511.
3. T. Jansen, K. A. De Jong (2002): An analysis of the role of offspring population size in EAs. *Genetic and Evolutionary Computation Conf.* (*GECCO 2002*), Morgan Kaufmann. 238–246.
4. T. Jansen, I. Wegener (2001): On the utility of populations. *Genetic and Evolutionary Computation Conf.* (*GECCO 2001*), Morgan Kaufmann. 1034–1041.
5. T. Jansen, I. Wegener (2002): On the analysis of evolutionary algorithms — a proof that crossover really can help. Algorithmica 34(1):47–66.
6. T. Jansen, I. Wegener (2004): Real royal road — where crossover provably is essential. To appear in Discrete Applied Mathematics.
7. T. Jansen, R. P. Wiegand (2003): Exploring the explorative advantage of the cooperative coevolutionary (1+1) EA. *Genetic and Evolutionary Computation Conf.* (*GECCO 2003*), LNCS 2724, Springer. 310–321.
8. T. Jansen, R. P. Wiegand (2003): Sequential versus parallel cooperative coevolutionary (1+1) EAs. In *Congress on Evolutionary Computation* (*CEC 2003*), IEEE Press. 30–37.
9. T. Jansen, R. P. Wiegand (2004): The cooperative coevolutionary (1+1) EA. Accepted for *Evolutionary Computation*.
10. M. A. Potter, K. A. De Jong (1994): A cooperative coevolutionary approach to function optimization. *Parallel Problem Solving from Nature* (*PPSN III*), LNCS 866, Springer. 249–257.
11. Y. Rabani, Y. Rabinovich, A. Sinclair (1998): A computational view of population genetics. Random Structures and Algorithms 12(4):313–334.
12. T. Storch, I. Wegener (2003): Real royal road functions for constant population size. *Genetic and Evolutionary Computation Conf.* (*GECCO 2003*), LNCS 2724, Springer. 1406–1417.
13. R. P. Wiegand, B. Liles, K. A. De Jong (2002): The effects of cross-population epistasis on collaboration methods in cooperative coevolution. *Parallel Problem Solving from Nature* (*PPSN VII*). LNCS 2439, Springer. 257–270.
14. C. Witt (2003): Population size vs. runtime of a simple EA. *Congress on Evolutionary Computation* (*CEC 2003*), IEEE Press. 1996–2003.
15. C. Witt (2004): An analysis of the $(\mu+1)$ EA on simple pseudo-boolean functions. Accepted for GECCO 2004.
16. L. Bull (1997): Evolutionary Computing in Multi-agent Environments: Partners. *Int'l Conf. on Genetic Algorithms* (*ICGA 1997*), Morgan Kaufmann. 370–377.