On the Performance Effects of Unbiased Module Encapsulation

R. Paul Wiegand
Institute for Simulation & Training
University of Central Florida
Orlando, FL 32826
wiegand@ist.ucf.edu

Gautham Anil
School of Electrical Eng. & Computer Science
University of Central Florida
Orlando, FL 32826
ganil@cs.ucf.edu

Ivan I. Garibay
Office of Research & Commercialization
University of Central Florida
Orlando, FL 32826
igaribay@cs.ucf.edu

Ozlem O. Garibay
Office of Research & Commercialization
University of Central Florida
Orlando, FL 32826
ozlem@cs.ucf.edu

Annie S. Wu
School of Electrical Eng. & Computer Science
University of Central Florida
Orlando, FL 32826
aswu@cs.ucf.edu

ABSTRACT

A recent theoretical investigation of modular representations shows that certain modularizations can introduce a distance bias into a landscape. This was a static analysis, and empirical investigations were used to connect formal results to performance. Here we replace this experimentation with an introductory runtime analysis of performance. We study a base-line, unbiased modularization that makes use of a *complete module set* (CMS), with special focus on strings that grow logarithmically with the problem size. We learn that even unbiased modularizations can have profound effects on problem performance. Our (1+1) CMS-EA optimizes a generalized OneMax problem in $\Omega(n^2)$ time, provably worse than a (1+1) EA. More generally, our (1+1) CMS-EA optimizes a particular class of concatenated functions in $O(2^{l_m} kn)$ time, where l_m is the length of module strings and k is the number of module positions, when the modularization is aligned with the problem separability. We compare our results to known results for traditional EAs, and develop new intuition about modular encapsulation. We observe that search in the CMS-EA is essentially conducted at two levels (intra- and extra-module) and use this observation to construct a *module trap*, requiring super-polynomial time for our CMS-EA and $O(n \ln n)$ for the analogous EA.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity;
G.1.6 [**Numerical Analysis**]: Optimization

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

General Terms

Theory, Algorithms, Performance

Keywords

module encapsulation, runtime analysis, search space bias

1. INTRODUCTION

Many researchers have begun to develop complex encodings that try to exploit known regularities in a problem using notions like *modularity* in the representation space [10, 8, 17]. Modular representations permit repeatable and reusable components of a candidate solution (modules) to be embedded into the encoding of candidate solutions, which can alter the structure of the search space and the performance of the EA operating on that space. Indeed, many engineering problems seem well-addressed by algorithms using representations that make use of such ideas (e.g., multiagent learning [3]). A useful survey of concepts and definitions for modular representations appears in [4].

Empirical investigations into modularity have primarily focused on understanding how regularity in the landscape affects such representations [10, 15, 8]. Algorithmic design choices include: how many modules exist, how they should be composed, whether they can be dynamically discovered or generated, or whether there is a static *a priori* list of possible modules [2, 6]. Proponents of modular representations highlight, among other advantages, the potential for these representations to provide greater *scalability* for increasingly complex problems.

Theoretical approaches to understanding modularity have been much more limited and mainly confined to general investigations of genotype-phenotype mapping [19, 18]. A more focused analytical study of the effects of module encapsulation appeared recently [9], where simple modular encapsulations are considered from the perspective of search-space bias. That is, modular representations can introduce a kind of *distance bias* by over-representing some portion of the search space and under-representing another. As a kind of baseline, they introduce a representation where solutions are encoded only by modules and the potential module set

consists of *all* possible substrings of a certain length (the *complete module set*, CMS). The paper proves that, via the application of a particular distance measure, this leaves the resulting search space *unbiased*. The authors focus their formalism on a *static* landscape analysis, using empirical studies to connect notions of representational bias to performance based on a well-known informal difficulty measure, and the experimental results suggest that unbiased representations will not impact performance when problem landscapes are well-suited to their distance measure.

Here we address the issue of scalability directly by formally examining the performance impact of such unbiased modular representations. The idea is to extend initial static investigations of landscape bias by replacing empirical study with a rigorous runtime analysis of performance. We study very simple algorithms, variants of the so-called (1+1) EA [5] in order to focus on our primary questions of representation, though at least one result generalizes to more sophisticated methods. Additionally, we confine ourselves to this first, simple modularization using a complete module set as an introductory step into performance analysis of modular representations.

We learn that even with a simple problem that appears quite compatible with the difficulty estimate used in existing experimental results, representations using unbiased modular encapsulation can still have provably different performance results from an analogous EA. Specifically, we show that a simple (1+1) EA will outperform our CMS variant on a generalized OneMax problem. We then generalize this bound for our CMS-EA to a particular class of *aligned concatenated* functions, providing upper and lower bounds on such functions. These analyses permit comparisons with traditional EAs on several problem classes and provide new intuition about modular encapsulation: When problems align well with the separability of the function, the module length relative to the problem size of an unbiased modular encoding can have a *bracketing* effect on the performance scaling of the EA, constraining what is possible for both the upper and lower bounds. Additionally, we see how search in the CMS-EA is essentially conducted at two levels (intra- and extra-module). This insight enables us to construct a *module trap*, which is super-polynomially difficult for the CMS-EA and quite straightforward for the EA.

The take-home message is that even simple modularization schemes for simple algorithms applied to simple problems generate surprising results and deserves further theoretical attention. Runtime analysis can be a useful way to extend existing efforts to understand the static effects of representational bias generated by a modularization scheme.

The next section provides the technical background by formalizing our notion of modular encapsulation and describing the runtime analysis framework we employ. Section 3 describes the two algorithms we consider in this paper and motivates the rest of the discussion by providing a lower-bound for our CMS variant on generalized OneMax. Section 4 provides the analysis of the more general class of aligned concatenated problems and provides bounds for examples both in and out of that class. The final section discusses what we conclude from these results.

2. BACKGROUND

2.1 Modular Encapsulation

The formalism for mapping from a modular encoding to the true problem encoding discussed in [9] is quite general, and we refer the reader there for that framework. Here we focus only on a relevant subset of modularization schemes on which their analysis was conducted and simplify our formalism for the purposes of clarity of exposition. Where possible, our notation is consistent with both this work and the traditional runtime analysis literature; however, where this is confusing, we employ the latter.

We apply our algorithms to pseudo-Boolean problems, where the domain space consists of binary strings of length n, $f : \{0,1\}^n \mapsto \mathbb{R}$. We consider a *module* to be a symbol that represents some binary substring of fixed length, $M_i \in \{0,1\}^{l_m}$ and use a *complete module set* (CMS), $\mathcal{M}$, consisting of symbols representing each copy of all such strings exactly once. That is, $\mathcal{M} := \{M_0, \ldots, M_{2^{l_m}-1}\}$, where M_j is the j^{th} binary string of length l_m indicated in $\mathcal{M}$.

A potential solution is encoded as a string of module symbols, which is translated into a binary string by simple substitution. Before substitution, we write such a solution $m := m_1 m_2 \cdots m_k$, where k is the length of the modular string and $n := k \cdot l_m$. We write the final solution string, after substitution, $x := x_1 x_2 \ldots x_n$. When it is useful, we occasionally use the notation $x^{(i)}$ to refer to the substring corresponding with the i^{th} module of the string, $m_i = x^{(i)} := x_{(i-1)l_m+1} \cdots x_{i \cdot l_m}$.

Examining both an un-encoded binary representation in conjunction with the above CMS representation, it is easy to see that there is no difference in the underlying points of the space being represented. That is, in the final search space, algorithms employing either representation will have access to all 2^n search points, and none are any more or less represented than another. A slightly subtler question is whether or not there are topological changes to the space as a result of the encoding, and such a question can only be answered when some kind of distance measure is imposed.

[9] examine the effects of modularization on the space. They employ *Hamming distance* to the global optimum to show that average Hamming distance between the pre- and post-encoded spaces do not differ when using a complete module set. In that sense, the encapsulation is *unbiased* because the aggregate relationship of underlying search points to the optimum is unchanged. Biases can be introduced with more sophisticated choices about module set composition.

That analysis is static, considering only the effects on the representation space given a distance measure. To say something about performance of algorithms employing these schemes, the static descriptions of bias are connected to a well-known "difficulty" measure in the EC literature, *fitness distance correlation* [14], which relates distance to the optimum to solution quality. This difficulty measure is used to help make predictions about how these bias properties will perform experimentally.

2.2 Performance & Runtime Analysis

Though it may be informative in certain circumstances, fitness distance correlation is a poor general predictor for EA performance [1]. Moreover, the measure is meant to describe a problem's "difficulty" and, whether or not it achieves this, it is clear that what is difficult for an EA using one rep-

resentation may not be difficult for another. It's instructive to consider what one wants out of performance comparison.

If performance and scale of performance are of interest, it is productive to replace the empirical studies following static analysis of modular representation with runtime analysis of the first-hitting time of an algorithm to the global optimum. Here we try to elicit bounds for the expected number of function evaluations before an EA first evaluates the optimum, as well as provide bounds on the success probabilities [16, 20]. Since these bounds are typically expressed as function classes dependent on the size of the problem, and since one of the advantages modular representations are believed to have is scalability [4], determining such bounds as a function of problem size is particularly useful for further study the performance of modular representations.

A challenge for those conducting runtime analysis as a follow-on to existing work is the tendency for researchers to neglect to specify the functional relationship between certain algorithmic parameters and the problem size [12]. In this case the trouble comes from l_m: Is it constant? Does it grow linearly with n? For our analysis, we primarily focus on $l_m := \lg n$; our reasoning follows.

If the length of the module string is constant in n then there is no real difference between a modular encapsulation and a representation that uses a non-binary alphabet. Of course either can impact performance and may be worthy of study, but our view is that true "modular" representation is one in which the modules have some hope of incorporating useful sub-portions of (some) problems, and that suggests some non-constant relationship to the size of the problem. If the length of the module string is linear in n then the size of $\mathcal{M}$ grows exponentially as n increases, which is impractical if an implementation explicitly stores $\mathcal{M}$ (though often it isn't necessary to do so). With our choice, the size of the module set grows linearly with n, which is a useful property since it leads to a kind of apples-to-apples compatibility with the (1+1) EA, which we will discuss later.

Other scaling function classes for l_m are possible and interesting, of course,$l_m := \sqrt{n}$ in particular. More generally, one could imagine a class of scaling functions $l_m := n^\epsilon$, where $\epsilon \in [0, 1]$. When ϵ is 0, our CMS-EA is equivalent to an analogous EA, and when it is 1, our CMS-EA is random search. A thorough analysis of the effects of the range in between is beyond the scope of this paper; however, we will keep these extremes in mind in our discussion.

3. ALGORITHMS ANALYZED

For several reasons, we concentrate our analysis on very simple evolutionary algorithms, variants of the so-called (1+1) EA. First, we wish to focus on the question of representation rather than other algorithmic details, so it is sound to consider the simplest algorithm that demonstrates our points. Second, there are now many theoretical runtime results for this simple EA, so concentrating on this provides the most opportunity for comparison. Still, at least one of our bounds is quite general, and we will discuss the extent to which it will include other algorithms.

3.1 Pseudocode

Without loss of generality, our algorithms are described for maximization. We describe them without stopping criteria since our interest is in the expected time until the algorithm first evaluates a global optimum (first-hitting time).

The (1+1) EA begins with a single random parent, produces a child by the standard bit-flip mutation operator, and replaces the parent with the child if it is at least as good.

Algorithm 1 $(1 + 1)$ EvolutionaryAlgorithm

1. Choose $x \in \{0, 1\}^n$, uniformly
2. $t := 0$
3. **Mutation:** Create $x' \in \{0, 1\}^n$ by copying x and independently flipping each bit with probability $\min\{1/n, 1/2\}$
4. **Selection:** If $f(x') \geq f(x)$ then set $x := x'$
5. $t := t + 1$
6. Continue at line 3

Our CMS variant functions similarly, except that it operates on the modules directly and only decodes module strings for evaluation. It maintains an individual that has k modules, produces a new individual by applying mutation, then decodes the two module strings into binary strings and evaluates their fitness. If the child is at least as good as the parent, the child's module string replaces the parent's.

The important observation here is that the *mutation operator must change* to accommodate the modular representation. We use the same operator described by [9, 7]: Each position has an independent probability of having mutation applied to it, $1/k$. However, now we choose a new module from the module set uniformly at random. This means that there is some (low) probability that the very same module will be selected, which essentially implies that the effective mutation rate is somewhat smaller than $1/k$.

Algorithm 2 $(1 + 1)$ CMS-EvolutionaryAlgorithm

1. Choose $m \in \mathcal{M}^k$, uniformly
2. $t := 0$
3. **Mutation:** Create $m' \in \mathcal{M}^k$ by copying m, then
   for Each m'_i **do**
   Decide if mutating with probability $\min\{1/k, 1/2\}$
   If mutating, choose a new $m'_i \in \mathcal{M}$, uniformly
   end for
4. **Mapping:** Decode $m \to x$, $m' \to x'$ by substitution
5. **Selection:** If $f(x') \geq f(x)$ then set $m := m'$
6. $t := t + 1$
7. Continue at line 3

3.2 Why it Matters: A Tale of Generalized OneMax

Since both modular bias and fitness distance correlation are measures of search point distance to the global optimum, and functions from the simple generalized OneMax class are defined in terms of the very same distance measure, an easy inference to draw is that unbiased modular encodings will not impact the performance of the algorithm on such problems; however, this is not true.

Consider the generalized OneMax problem:

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

where $\hat{x}$ specifies a target string (global optimum).

THEOREM 1. *The expected first-hitting time for the (1+1) CMS-EA on generalized* ONEMAX *is* $\Omega\left(2^{l_m} k \ln k\right)$ *if it is initialized with no modules solved.*

PROOF. We examine the event when mutation produces a specific module symbol that decodes to the correct substring in the target string, referring to such an event as having *considered* the solution to that module. For this to happen for a module position, we must both have a mutation event and that event must produce the correct value, the probability for which is $\frac{1}{k} \cdot \frac{1}{2^{l_m}}$. The probability that this event *does not happen* in $t-1$ steps is:

$$\left(1 - \frac{1}{2^{l_m} k}\right)^{(t-1)}$$

The probability all k modules have been considered is

$$\left(1 - \left(1 - \frac{1}{2^{l_m} k}\right)^{(t-1)}\right)^{k}$$

Let $(t-1) := \left(2^{l_m} k - 1\right)(\ln k - \ln\ln n)$. Using the well-known inequality $(1-1/n)^{(n-1)} \geq 1/e$, we have:

$$\left(1 - \frac{1}{2^{l_m} k}\right)^{\left(2^{l_m} k - 1\right)(\ln k - \ln\ln n)} \geq e^{-(\ln k - \ln\ln n)} = \frac{\ln n}{k}$$

So the probability that the algorithm will have considered all module symbols needed in the solution string within $t-1$ steps is at most $\left(1 - \frac{\ln n}{k}\right)^k \leq e^{-\ln n} = 1/n$; thus, the probability that it takes at least t steps is $1 - 1/n$. Noting that

$$E\{T\} = \sum_{t \in [1,\infty]} Pr\{T \geq t\} = \sum_{t \in [1,\infty]} t \cdot Pr\{T = t\},$$

we have

$$\begin{aligned} E\{T\} &\geq \left(2^{l_m} k - 1\right)(\ln k - \ln\ln n)\left(1 - \frac{1}{n}\right) \\ &= \Omega(2^{l_m} k \ln k) \qquad \square \end{aligned}$$

The proof shown above is a special case of the work in [5] and follows their method. The (1+1) EA solves this problem class in $\Theta(n \lg n)$ time while our CMS variant is $\Omega\left(\frac{n^2}{\lg n} \ln(n/\lg n)\right) = \Omega(n^2)$ when $l_m = \lg n$, provably worse. Obviously this difference is relatively minor; however, it serves to motivate the analysis below. The Hamming distance correlation of search points to the optimum is an ideal distance measure for this landscape. Yet in spite of the fact that the CMS is unbiased according to that measure, these two algorithms scale by different function classes. At issue is the effect of the growth in the module length on the mutation operator, as we shall see.

4. CONCATENATED FUNCTIONS

As in [5], the lower bound we just presented is much more general than ONEMAX. Indeed, it holds (perhaps loosely) *for any* function with a unique global optimum — one provably worse than that of the (1+1) EA when $l_m := \lg n$. The reason for this is that modularity forces a change in the mutation operator that induces a *different* topology on the underlying space than does bit-flip mutation. Since the length of the module string increases with n, the probability of mutating to the module containing the correct substring of the solution decreases. For any non-constant l_m, this lower bound must be higher than $\Omega(n \ln n)$, so the scale of l_m has a kind of "raising the bar" effect on what is possible for a lower bound.

Also, note that the ratio of the length of the modules to the length of the candidate solution string shrinks as n increases. Since the number of modules in the module set is linear in n, there is a kind of underlying compatibility with the (1+1) EA: given that a mutation event occurs, the correct module can be found with probability $1/n$. The difference is in what the algorithms do with the information inside a module — the (1+1) EA treats all information the same and ignores substring boundaries, while the (1+1) CMS-EA ignores most information inside the substring and concentrates search on assembling the useful modules. Indeed, the algorithm is performing two simultaneous searches: a mutation/section based search of useful modules and an essentially *random* search to find the correct module values.

This observation leads one to think naturally of functions in which this dual-search notion is exploited, where the problem itself is composed of pieces, so-called *concatenated* functions. We define a class of generalized concatenated functions below, using the same notion of generalization we did for the ONEMAX problem above:

Definition 1. Let $\hat{x} \in \{0,1\}^n$ be the unique global optimum of some function $f : \{0,1\}^n \mapsto \mathbb{R}$. We partition all search points, $x \in \{0,1\}^n$, into r pieces each of length l_r such that $x^{(i)} := x_{(i-1)l_r+1} \cdots x_{i \cdot l_r}$. Given some function $g : \{0,1\}^{l_r} \times \{0,1\}^{l_r} \mapsto \mathbb{R}$, we define a *generalized concatenated* pseudo-Boolean function by

$$f(x, \hat{x}) := \sum_{i=1}^{r} g\left(x^{(i)}, \hat{x}^{(i)}\right)$$

Further, when $l_r = \lg n$, we refer to such functions as *log-separable concatenated*.

Since we require f to have a unique global optimum and that it be composed of identical functions operating over different portions of the solution string, g must also have a unique global optimum. Also, since the g function might itself be a linear combination of subordinate functions, the *log-separable concatenated* stipulation does not restrict the class to only functions that are precisely separable on $\lg n$ boundaries, but to those that are at least separable along such boundaries (e.g., the generalized ONEMAX just discussed is such a function). In fact, there are many functions that can fall into this class (e.g., concatenated TRAP and CLOB, discussed below). Finally, while we have conveniently arranged our notation to match the definition for the module boundaries of the CMS-EA offered above, it's clear this needn't be so. The bit positions themselves might be presented in a different order and the underlying function would be the same (if the corresponding positions in $\hat{x}$ were also re-ordered in the same way).

This is important because the CMS-EA makes an implicit assumption about where good module boundaries may be, much as compositional coevolutionary algorithms do [13]. When the module boundary and the boundaries of the *pieces* of the log-separable concatenated function are the same, we consider the algorithm's modularization to be *aligned* with the problem. Below, we consider the general case of when modules are aligned with concatenated functions, as well as a counter example for CMS-EA that is outside this class.

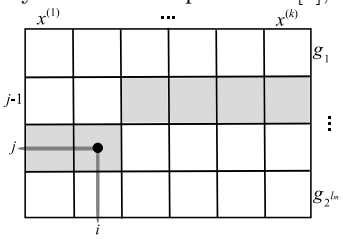
4.1 Module Alignment

The purpose here is to show how the bounds on the CMS-EA's runtime behavior applied to the class of generalized concatenated functions is affected when the modularization is aligned with the problem. Consequently, we intentionally ignore the details of the component function for the underlying pieces except to insist it have a unique global optimum.

THEOREM 2. *The expected first-hitting time for the (1+1) CMS-EA on a function from the class of generalized concatenated functions is $O\left(2^{l_m} kn\right)$ when the modularization is aligned with the problem.*

PROOF. By our definition, using a modularization that is aligned with the problem implies that $l_r \leq l_m$, so we use l_m in all cases below for simplicity.

We first consider the case where all $g(x, \hat{x})$ are unique. Without loss of generality, we consider the values ordered by their component function value, writing g_1 for the maximal value of g and $g_{2^{l_m}}$ for the minimal value. We proceed by fitness-based partitions [5], defining $2^{l_m} k + 1$ levels:



$$L_{ij} := \{x| \quad g_{j-1} \cdot (i+1) + g_j \cdot (k-i-1) > \sum_{d=1}^{k} g(x^{(d)}, \hat{x}^{(d)}) \geq g_{j-1} \cdot (i) + g_j \cdot (k-i)\},$$
$$\forall i \in [0, k-1], \forall j \in [2, 2^{l_m}]$$

$$L_{\text{OPT}} = L_{k1} := \left\{ x \left| \sum_{d=1}^{k} g(x^{(d)}, \hat{x}^{(d)}) = g_1 \cdot k \right. \right\}$$

To increase a level, the algorithm must mutate a module to a value that receives a larger payoff than the current value without generating deleterious mutations. Though there can in principle be a large number of combinations of component function values that result in a particular fitness level, for a given level L_{ij}, there must be *at least one* module value at or below the g_{j-1} level. We pessimistically assume that the we must mutate only that module to some larger value. This occurs with probability at least $\frac{1}{k}\left(1-\frac{1}{k}\right)^{k-1} \frac{j-1}{2^{l_m}} \geq \frac{j-1}{e 2^{l_m} k}$. We must make at most $2^{l_m} k$ such level advances and we estimate the total time by summing the expected time for each of such level increases:

$$E\{T\} \leq e 2^{l_m} k + \sum_{j=2}^{2^{l_m} k} \sum_{i=0}^{k-1} \frac{e 2^{l_m} k}{j-1}$$
$$= O\left(2^{l_m} k^2 \ln 2^{l_m}\right) = O\left(2^{l_m} kn\right)$$

Second, we consider the case where there exists j such that $g_{j-1} = g_j$. If the levels are defined as above, the fitness partition size is now zero, and it is possible to move backwards. Instead, one can define the fitness partitions so as to merge levels such that no fitness partition is of size zero. If this is done in such a way that rows comprising levels of size zero are repeatedly merged *upward* into rows *above* them until they contain valid points, one can ensure that module substrings decoding to the largest possible component fitness value from g appear only once in a given partition for each module position, and that all other values are the same and smaller. Thus, a module mutating between two points with equal fitness values does not lead to a transition in fitness partition. The probability of making a transition is still governed by the rank of the larger value in a partition for a given module, there are simply fewer rows and hence fewer fitness partitions than before. Since the probabilities are no worse and there are fewer steps to make, this case cannot be worse than the case where all g values are unique. □

There is a similar limiting pattern to the upper bound as we noted for the lower bound: Sub-linear scaling for l_m and l_r leads to sub-exponential bounds on this function class, and there is a kind of "lowering the bar" effect on what is possible for an upper bound that occurs when this scale is reduced. Taken together, there is a *bracketing* on the bounds for these kinds of concatenated problems determined by the size of modules when they are no larger than the problem pieces and the modularization is aligned. This follows naturally from the observation that the algorithm is essentially ignoring the values within the pieces and simply performing a traditional search at the module level. What the (1+1) EA does with bits, the (1+1) CMS-EA does with modules, and when the additional gradient information provided in those pieces is helpful for search, the CMS-EA cannot make use if it. But when that information is harmful or misleading, the CMS-EA is undeterred. In essence, bracketing results from the fact that the CMS-EA doesn't care about g.

Considering the (1+1) EA as an extreme case of the CMS-EA on linear functions, $l_r = l_m := 1, k = n$, our lower bound result is consistent with the known $\Theta(n \ln n)$ result for the 1-separable concatenated function class (generalized ONEMAX). At the other extreme, when $l_r = l_m := n, k = 1$, the CMS-EA performs random search, and the our bounds loosely corroborates this, as well. When $l_m := \lg n$, our lower bound is quadratic and the upper bound is *nearly* quadratic, $O\left(n^3 / \lg n\right)$.

Our result is general enough to compare with known results for several function classes. The concatenated TRAP function, where g decreases with the Hamming distance to its optimum at every point except that optimum, requires $n^{\lg n}$ time for the (1+1) EA if the length of the pieces grow as $\lg n$. The CLOB (concatenated LEADINGONESBLOCK) problem, where g is the sum of the left-to-right consecutive b 1-bit blocks, is $\Theta(n^b(n/(br) + \ln r))$ and so depends heavily on the size of the block [13]. The CMS-EA can solve instances of both problem classes faster when $l_r = \lg n$ and $b \geq 2$, as well as many others (of course).

The BUILDINGBLOCK function class from [11] is also a concatenated function, though the analysis from that paper uses $l_m := \sqrt{n}$. The case of $\sqrt{n}$ is particularly interesting since a simultaneous mutation of $\lg n$ bits is not so improbable, whereas this cannot be said for $\sqrt{n}$. Here, the running time of the CMS-EA with $l_m := \sqrt{n}$ on functions in this class could be quite unpleasant, $\Omega(2^{\sqrt{n}} n^{3/2} \lg n^{3/2})$.

But for the stated problem classes, the CMS-EA will solve such problems in nearly quadratic time. We do not suggest that this aligned log-separable concatenation property is sufficient or necessary for a CMS-EA advantage — there

are sub-classes in this class that are (somewhat) easier or no harder for a simple EA or other method to solve, and there very well may be functions on which the CMS-EA is advantageous that are *not* log-separable concatenated. But one clear effect that using such a module scheme has is to reduce the possible range of the bounds, and while the bounds will not be the same for more sophisticated variants of the algorithm (those with a population, for instance), we speculate that there will be a similar bracketing effect on what kind of runtime performance is possible for such problems.

4.2 A Module Trap

Obviously there are many functions for which the CMS-EA will perform very poorly — the full TRAP function, for instance. Such examples have their place, but a more interesting counter example here can be constructed using the same observation that helped us understand one class of functions for which our CMS-EA is reasonably efficient: the CMS-EA cannot make effective use of the information *inside* the modules and has no effective means of searching the relationships between the internal content of the modules.

An obvious way to exploit this search methodology is to create a trap for the modules, though the underlying information in the modules contains sufficient information to solve the problem. We construct such a function class as follows. First, let NEEDLE be a function that returns 1 when two given strings match and 0 when they do not. Interpreting NEEDLE as our g function in the above framework allows us to specify the class of concatenated functions:

Definition 2. Let $\hat{x} \in \{0,1\}^n$ be the unique global optimum of the function CNEEDLE : $\{0,1\}^n$. We partition search points, $x \in \{0,1\}^n$ into r pieces each of length l_r such that $x^{(i)} := x_{(i-1)l_r+1} \cdots x_{i \cdot l_r}$. We define this by

$$\text{CNEEDLE}(x, \hat{x}) := \sum_{i=1}^{r} \text{NEEDLE}(x^{(i)}, \hat{x}^{(i)})$$

We use the following notational short-cuts:

- $\text{OM}(x) = \text{ONEMAX}(x, \hat{x})$
- $\text{ZM}(x) = \text{ONEMAX}(x, \bar{\hat{x}})$
- $\text{CN}(x) = \text{CNEEDLE}(x, \bar{\hat{x}})$
- $\text{MINOM}(x)$ is short for $\min_{i \in [1,k]}\{\text{OM}(x^{(i)}, \hat{x}^{(i)})\}$

Definition 3. Let $\hat{x} \in \{0,1\}^n$ be the unique global optimum of the function MODTRAP : $\{0,1\}^n$, $\bar{\hat{x}}$ be the complement of $\hat{x}$, and $x \in \{0,1\}^n$ be a search point. We define

$$\text{MODTRAP}(x, \hat{x}) := \begin{cases} n^4 & \text{if } x = \hat{x} \\ n^3 \cdot \text{ZM}(x) & \text{if } \text{CN}(x) > 0 \\ \text{OM}(x) + n^2 & \text{if } \text{MINOM}(x) \geq \ln \lg n \ \wedge \\ & \quad \text{CN}(x) = 0 \\ n \cdot \text{MINOM}(x) + \text{OM}(x) & \text{otherwise} \end{cases}$$

Intuitively, each of the two underlying functions, CNEEDLE and ONEMAX, are concatenated functions and could be solved efficiently (if the log-separable concatenated and properly aligned); however, for the combined function, once a NEEDLE is *activated*, the function provides information leading away from the optimum. We also add a minor contrivance that prevents backward drift when very close, but outside of the trap region. It's useful to see that the ratio of NEEDLE to non-NEEDLE points in the landscape is quite small and shrinks as n grows, so the problem class is not that different from ONEMAX at the bit level when n is large. Indeed, as we show below, the (1+1) EA will solve this problem in $O(n \lg n)$ time, while the problem is quite difficult for the CMS-EA.

THEOREM 3. *The expected first-hitting time for the (1+1) EA on* MODTRAP *is* $O(n \lg n)$ *if a* NEEDLE *is never activated, which occurs with probability* $1 - o(n^{-1/3})$.

PROOF. The proof is separated into three parts. First, we show that after initialization the EA will have at least $\ln \lg n$ bits in each piece. We follow by bounding the probability of activating a NEEDLE before the EA has completed $n^{\ln \lg n}$ steps. Finally we compute the expected number of steps to the optimum given that no NEEDLE is activated. Without loss of generality, we assume $\hat{x} = 1^n$ and $\bar{\hat{x}} = 0^n$.

Given a bit-flip mutation probability of $1/n$, applying Chernoff bounds we know that the probability of having at least $n/4$ one-bits in the complete string is $1 - e^{-\Omega(n)}$, so we pessimistically assume we have only $n/4$ 1's to distribute among the $n/\lg n$. First, we re-write $n/4$:

$$\begin{aligned} \frac{n}{4} &= \frac{n}{4 \lg n}\left(\frac{\ln n}{\lg 2}\right) \\ &= \frac{1}{4 \lg 2} \cdot \frac{n}{\lg n}(\ln n - \ln \lg n + \ln \lg n) \\ &= \frac{1}{4 \lg 2}\left(\frac{n}{\lg n} \ln\left(\frac{n}{\lg n}\right) + \frac{n}{\lg n} \ln \lg n\right) \end{aligned}$$

Lemma 17 from [13] provides a generalized form of the well-known *coupon collector's problem*, the expected number of throws necessary to ensure at least h balls in each of k bins is $\Theta(k \ln k + kh)$ with probability at least $1 - 1/\sqrt{k}$. For our result, this implies that with a probability at least $1 - 1/\sqrt{n/\lg n} = 1 - o(n^{-1/3})$ there are at least $\ln \lg n$ one-bits in each piece.

As long as no NEEDLE has been activated in initialization, there will never be fewer ones after initialization. Moreover, due to the MINOM condition, the only way to obtain fewer than $\ln \lg n$ 1-bits in a given module is to simultaneously flip all 1-bits to zero. So in the easiest case, the EA must flip $\ln \lg n$ bits to activate a NEEDLE, which occurs with probability at most $1/n^{\ln \lg n}$. Since there are $r = n/\lg n$ pieces of the problem, there are $n/\lg n$ opportunities to do so in each step, so considering n^2 steps gives us $\frac{n^3}{\lg n}$ such opportunities. The probability that there is never a NEEDLE activation event in that time is $\left(1 - \frac{1}{n^{\ln \lg n}}\right)^{\frac{n^3}{\lg n}}$.

If the NEEDLE is not activated, the algorithm proceeds much like it would for ONEMAX. The upper bound can be derived using the common f-based partitions approach for ONEMAX since in all cases where a NEEDLE is not activated it is sufficient to count only steps where precisely one 0 is flipped to a 1. Using ONEMAX as a proxy, we have $L_i := \{x | \sum_{j=1}^{n} x_j = i\}, 0 \leq i < n$. The probability of advancing a level is at least $(n-i)/ne$, so $E\{T\} = \sum_{i=0}^{n-1} \frac{ne}{n-i} = O(n \lg n)$.

The probability that no NEEDLEs were activated during initialization and none were activated before the optimum

was otherwise found, then, is at least:

$$(1 - o(n^{-1/3}) \cdot \left(1 - \frac{1}{n^{\ln \lg n}}\right)^{\frac{n^3}{\lg n}} = (1 - o(n^{-1/3})) \quad \square$$

THEOREM 4. *The expected first-hitting time for the (1+1) CMS-EA on* MODTRAP *when the modularization is aligned with the problem is* $\Omega\left(\left(2^{l_m} k\right)^{k/4}\right)$ *if a* NEEDLE *is activated before at most* $3k/4$ *modules have been solved, which occurs with probability* $1 - e^{-\Omega(k)}$.

PROOF. Again, we assume without loss of generality that the solution is found at the all-1 string, the complement at the all-0 string. Also, the fact that the modularization is aligned with the problem implies that $l_r \leq l_m$, so we use l_m in all cases below for simplicity.

We begin by considering the probability that at least one NEEDLE will have activated in $t := 2^{l_m} k/4$ steps. The probability that a NEEDLE is activated by a mutation event in a given position is $\frac{1}{2^{l_m} k}$, and there are $2^{l_m} k^2/4$ opportunities for such a mutation in t steps. So the probability of NEEDLE activation is at least:

$$1 - \left(1 - \frac{1}{2^{l_m} k}\right)^{2^{l_m} k \cdot \frac{k}{4}} \geq 1 - e^{-k/4}$$

Next, we bound the probability of having more than $3k/4$ solved modules in t steps. The probability of finding a solution substring via a given module mutation event is also $\frac{1}{2^{l_m} k}$, and given that there are k opportunities in each step to create find such a substring, the expected number of such solving events in time t is $2^{l_m} k^2/4 \cdot \frac{1}{2^{l_m} k} = k/4$. Applying Chernoff bounds, we see that the probability that there are more than $3k/4$ solving events in that time is $\left(e^2/9\right)^{k/4}$. Of course, not all of those events will occur in distinct module positions, but the number of solved modules cannot be more than the number "solving events" encountered in time t. Thus, the probability that at most $3k/4$ solved modules is at least $1 - \left(e^2/9\right)^{k/4}$.

We consider the first phase of the run to be these first $2^{l_m} k/4$ steps and bound the probability that by such time at least one NEEDLE will have been activated and no more than $3k/4$ modules can have been solved:

$$\left(1 - \left(e^2/9\right)^{k/4}\right) \cdot \left(1 - e^{-k/4}\right) = 1 - e^{-\Omega(k)}$$

We consider the second phase of the run to be the remaining steps required to solve the problem once a NEEDLE has been activated. Once this occurs, the algorithm cannot select an individual with fewer (or no) NEEDLES unless it simultaneously flips all remaining unsolved modules. In the best case, there are at least $k/4$ of these modules. The probability of mutating all $k/4$ of these modules to the solution substring at the same time is at most $\Omega(2^{l_m} k)^{-k/4}$ so the expected waiting time for such an event is $\left(2^{l_m} k\right)^{k/4}$.

Adding the time for the first phase to that of the second phase, we get $2^{l_m} k/4 + \left(2^{l_m} k\right)^{k/4} = \Omega\left(\left(2^{l_m} k\right)^{k/4}\right)$. $\square$

The (1+1) EA performs reasonably because it is able to use the intra-piece information to scale the Hamming levels to the solution before it has an opportunity to fall for the trap. The (1+1) CMS-EA is not so lucky. When $l_r = l_m = \lg n$, the lower bound is $\Omega\left(\left(n^2/\lg n\right)^{n^2/4 \lg n}\right)$. Because it cannot make use of the intra-block information, it must wait to assemble solved blocks. Unfortunately, it is very unlikely to solve all the modules before falling for the trap — and once in, it cannot easily escape.

5. CONCLUSIONS

In this paper, we provide a deeper look at the performance effects of modular representations described in [9] that use a *complete module set*. That work lays out a rigorous and careful analysis of how certain kinds of modularization impact search space bias in terms of their distance to the global optimum, but their discussion of the actual performance impact of such representations is largely empirical. Moreover, the empirical difficulty measure used implies that *unbiased* modular encodings, like those that include all substrings of a certain length (e.g., the CMS), will not typically impact search performance.

The goal of our paper is to extend existing analysis of modular representations to include formal investigations into their effects on EA performance. We have shown that even this very simple modularization can alter search performance, both positively and negatively. This can be true even when the problem is well-suited for the distance measure used to measure bias. Indeed, the *reason* fitness distance correlation is a poor predictor of CMS-EA performance is because it is attuned to a landscape topology induced by something like bit-flip mutation, whereas we now see that mutation in the CMS-EA leads to a quite different search mechanism.

The main technical difference between their work and ours centers around the relationship between the size of the modules and the size of the problem itself. We take the view that representations that presume constant-sized modules are really just changing the alphabet, while representations that use modules in the truest sense must admit some non-constant relationship between the module length and the size of the problem. We provide bounds for a general class of separable concatenated functions to which a properly aligned simple (1+1) CMS-EA can be applied, focusing on the subclass where the piece size scales with $\lg n$. This class includes functions that require arbitrarily large polynomial time, as well as those that require super-polynomial time for the (1+1) EA. The CMS-EA solves such problems in $\Omega(n^2)$, $O\left(n^3/\lg n\right)$ time when it's module size is also $\lg n$. Moreover, the general bounds relate in an informative way: $\Omega(2^{l_m} k \ln k)$ and $O(2^{l_m} kn)$. The relationship between l_m and n determines a kind of *bracketing* effect on the bounds and helps highlight a how the CMS-EA trades off searches inside and outside modules. We use this intuition to construct a counter example where the CMS-EA requires super-polynomial time to solve to help clarify this point.

More specifically, the CMS-EA is attempting to conduct two concurrent but different search processes: one in which individual genotypic values are being manipulated (intra-module search) and one in which assemblies of large components of the solution string are being explored (extra-module search). We note the relationship between EAs employing crossover, EAs employing modular representations, and cooperative coevolutionary EAs — all of which attempt to conduct this kind of two-level search. In cooperative coevolution, intra-module search occurs via genetic operators and extra-module search occurs via the collaboration mechanism [13]. With crossover, intra-module search employs mutation and extra-module search is performed via recombination.

In the case of our (1+1) CMS-EA, extra-module search uses an essentially mutation-based method, while intra-module search is blind, random search.

Still, the CMS-EA is merely a baseline. Understanding this aspect of the algorithm helps inform our notion of modular bias. Conceptually, one can imagine a transition matrix, for example, where transitioning from one module substring to another occurs according to some specified probability. With this notion, an intra-module search that is equivalent to bit-flip mutation can be produced by simply eliciting the probabilities from the binomial distribution, while one that is identical to our CMS-EA would be based on a uniform distribution. Now, though, we can imagine other distributions, and doing so offers an opportunity to think very generally about modular bias.

We believe this analysis, though quite coarse, improved our intuition about CMS-EAs, and we hope there will be increased interest in examining modularization from this perspective. Further runtime analysis of modular representation is certainly needed. In the future, we would like to consider population-based algorithms, as well as those that use more sophisticated operators. More important is the consideration of increasingly complex modular encodings, biased encodings in particular.

6. REFERENCES

[1] L. Altenberg. Fitness distance correlation analysis: An instructive counterexample. In *In Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 57–64. Morgan Kaufmann, 1997.

[2] P. Angeline and J. Pollack. Evolutionary induction of subroutines. In *Proceedings for the 14th Annual Cognitive Science Conference*, pages 236–241, 1992.

[3] D. D'Abrisio and K. Stanley. Generative encoding for multiagent learning. In *Proceedings of the 2008 Genetic and Evolutionary Computation Conference*. ACM Press, 2008.

[4] E. De Jong, D. Thierens, and R. Watson. Defining modularity, hierarchy, and repetition. In *Proceedings for the 2004 GECCO Workshop on Modularity, regularity and hierarchy in open-ended evolutionary computation*, pages 2–6. Springer, 2004.

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

[6] I. Garibay, O. Garibay, and A. Wu. Effects of module encapsulation in repetitively modular genotypes on the search space. In *Proceedings for the 2004 Genetic and Evolutionary Computation Conference*, pages 1125–1137. Springer, 2004.

[7] O. Garibay. *Analyzing the Effects of Modularity on Search Spaces*. PhD thesis, University of Central Florida, Orlando, FL USA, 2009.

[8] O. Garibay, I. Garibay, and A. Wu. The modular genetic algorithm: exploiting regularities in the problem space. In *Proceedings for the 2003 International Symposium on Computer and Information Systems*, pages 584–591. Springer, 2003.

[9] O. Garibay and A. Wu. Analyzing the effects of module encapsulation on search spaces. In *Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, pages 1234–1241. ACM Press, 2007.

[10] G. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In *Proceedings for the 2005 Genetic and Evolutionary Computation Conference*, pages 1729–1736. Springer, 2005.

[11] T. Jansen and R. Watson. A building-block royal road where crossover is provably essential. In *Proceedings from the 2007 Genetic and Evolutionary Computation Conference*, pages 1452–1459. ACM Press, 2007.

[12] T. Jansen and R. Wiegand. Bridging the gap between theory and practice. In *Proceedings from the 8th Parallel and Problem Solving from Nature*, pages 61–71. Springer, 2004.

[13] T. Jansen and R. Wiegand. The cooperative coevolutionary (1+1) EA. *Evolutionary Computation*, 12(4):405–434, 2004.

[14] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquergue, NM USA, 1995.

[15] H. Lipson. Principles of modularity, regularity, and hierarchy for scalable systems. In *Proceedings for the 2004 GECCO Workshop on Modularity, regularity and hierarchy in open-ended evolutionary computation*. Springer, 2004.

[16] P. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293, 2007.

[17] K. Stanley, D. D'Abrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life Journal*, (to appear), 2009.

[18] M. Toussaint. *The evolution of genetic representations and modular adaptation*. PhD thesis, Institut für Neuroinformatik, Ruhr-Universiät Bochum, Germany, 2003.

[19] M. Toussaint. Compact genetic codes as a search strategy of evolutionary processes. In *Foundations of Genetic Algorithms VIII*, pages 75–94. Springer, 2005.

[20] I. Wegener. Theoretical aspects of evolutionary algorithms. In *28th International Colloquium of Automata, Languages and Programming (ICALP 2001)*, pages 64–78. Springer, 2001.