Modular Variable-length Representations from Pareto-Coevolution

Edwin D. de Jong

institute of information and computing sciences, utrecht university

technical report UU-CS-2003-009

www.cs.uu.nl

Modular Variable-length Representations from Pareto-Coevolution

Edwin D. de Jong

March 20, 2003

Abstract

Genetic algorithms generally use a fixed problem representation that maps variables of the search space to variables of the problem, and operators of variation that are fixed over time. As a result, the potential to explore combinations of large partial solutions is limited. This is problematic when the search space of a problem is large, and scalability is required.

To address this issue, several researchers have proposed methods that form modules during the search. An open question is how modules in the resulting coevolutionary setup should be evaluated. Recently, a theoretical basis for evaluation in coevolution has been provided by Pareto-coevolution. We define the notion of functional modularity, and define objectives for module evaluation based on Pareto-Coevolution. It is shown that optimization of these objectives maximizes functional modularity.

The resulting evaluation method is developed into an algorithm for variable length, open ended representation development called *DevRep*. DevRep simultaneously forms modules and searches the space of module combinations. It successfully identifies large partial solutions and greatly outperforms fixed length and variable length genetic algorithms on several test problems, including the 1024-bit Hierarchical-XOR problem.

Keywords: Development of Representations, hierarchical modularity, bias learning, coevolution, Pareto-coevolution, Evolutionary Multi-Objective Optimization

1 Introduction

Most genetic algorithms employ a single, fixed representation that is given as part of the problem specification. To apply a genetic algorithm to a problem, a mapping has to be chosen between *genotypes* (sequences of binary or other variables) and the actual individuals they represent, *phenotypes*. We will call this mapping the *representation* of the problem. For most genetic algorithms, the representation is chosen once, and does not change during the algorithm's operation. The operators of variation are typically constant too. Thus, the search space and the operators specifying the possible moves in this space cannot be adapted by the algorithm.

In combination, the use of a fixed representation and fixed operators of variation make it unlikely that different combinations of large partial solutions[1] will be explored, see (Thierens, 1999). Both forming large partial solutions and mixing existing genotypes can be achieved in isolation. However, the standard genetic algorithm makes it unlikely for *overlapping* partial

[1] We will use the term partial solution in the sense of a schema, i.e. a partial specification of a genotype.

solutions to persist, since no mechanism for protecting alternative partial solutions is present. Combination of different large partial solutions is only likely if there is no overlap between these partial solutions, which implies a restriction to separable problems[2].

Even when niching is used and different partial solutions for the same variables can be maintained, partial solutions are not explicitly represented. Thus, crossover is unlikely to respect the boundaries of the partial solutions in both individuals.

We are interested in large search problems that have structure, as will be made more precise. Such problems may require the maintenance and combination of different large partial solutions. Furthermore, the required length for correct solutions may not be known in advance. Therefore, the question we study is how combinations of large partial solutions may be explored in variable length settings.

Recombination of intact partial solutions can be made more likely by explicitly representing modules, where a module specifies a set of variables and their setting. Several authors have studied setups where modules evolve in the context of solution candidates that may employ them, see GLiB (Angeline & Pollack, 1992), MIL (Juille, 1999; Juille & Pollack, 1996), ADF's (Koza, 1994), ARL (Rosca & Ballard, 1996), ADSN (Gruau, 1994), and SEAM (Watson & Pollack, 2000). The development of modules allows such algorithms to start searching in terms of *combinations* of variables. Thus, these algorithms can be viewed as adapting the *representation* of the problem during search (Angeline & Pollack, 1994; Rosca & Ballard, 1996; De Jong & Oates, 2002b). An alternative approach that appears very different in approach is that taken by estimation of distribution algorithms, e.g. (Bosman & Thierens, 1999; Pelikan, Goldberg, & Cantu-Paz, 1999). However, in order to address hierarchical problems in a scalable manner, these methods too need to maintain partial solutions; in H-BOA for instance, this role is performed by decision trees (Pelikan & Goldberg, 2001).

So far, there has been a lack of theory to guide the choices in algorithms that form modules. In particular, an open question has been how modules should be evaluated. Here, we will apply a recently developed notion called *Pareto-coevolution* to address this issue. This theory provides a basis for evaluation in coevolution in general, of which coevolution of modules and contexts is a special case.

We are interested in methods that simultaneously evolve *modules* and candidate solutions consisting of such modules, which we will call *contexts*. Such methods are instances of coevolution. We take coevolution to refer to setups where the evaluation of individuals is influenced by or based on interactions with other evolving individuals. This substantially changes the standard evolutionary setup, since the *ranking* of individuals may be reversed as a result of changes in the population. This distinguishes coevolution from conventional evolution, where other individuals can only affect the *scaling* of fitnesses. When modules are evaluated based on their role in contexts, their evaluation critically depends on the available contexts. Thus, the evolution of modules and contexts is a form of coevolution.

Recently, the paradigm of Pareto-coevolution has provided a theoretical basis for evaluation in coevolution, see (Ficici & Pollack, 2000; Watson & Pollack, 2000; Bucci & Pollack, 2002; De Jong & Pollack, 2003b). In standard coevolution, individuals are typically evaluated based on the *average* outcome of interactions with other individuals. In Pareto-coevolution, the outcomes of these interactions are viewed as separate *objectives*, in the sense employed by multi-objective optimization (Fonseca & Fleming, 1993, 1995; Van Veldhuizen, 1999; Deb,

[2]A problem is *separable* if each variable has a single optimal setting, independent of the other variables (Watson, 2002).

2001).

The first algorithm to use Pareto-coevolution for module evaluation is Watson's SEAM algorithm, see (2000). While SEAM is designed for fixed length problems, here we will investigate relations between modularity and Pareto-coevolution to arrive at an algorithm that is suited for use in variable length problems.

The structure of the article is as follows. First the application domain is described in section 2. Next, the method is gradually introduced, by discussing structural and functional modularity (3), coevolution of modules and contexts (4), Pareto-coevolution 5, evaluation of modules based on Pareto-coevolution 6, and finally the algorithm following from this principle 7. In section 8, a number of variations of the standard algorithm are described. Section 9 describes a hierarchical test problem for variable length evolution methods called H-XOR. Furthermore, a pattern recognition task is described. Experimental results are reported in section 10, followed by discussion and conclusions.

2 Application Domain: Problems with Structure

A large search problem is a problem that requires a long solution, when specified in terms of the original variables of the problem. We are interested in large search problems for which information from part of the search space can be used to predict (better than random) information about other parts of the search space. By exploiting such information, algorithms can learn from experience. We will denote such problems as *having structure*.

Large search problems that have no structure require sampling a significant fraction of the search space, depending on the fraction of the search space that represents acceptable solutions. Since the size of the search space is exponential in the length of the individuals that need to be considered, search problems without structure are typically infeasible, meaning that large instances cannot be addressed. An example of a problem without structure is the needle-in-a-haystack problem of searching for a single n-digit number with nonzero fitness among all possible n-digit numbers. Other examples of problems without structure are maximally epistatic problems, such as $n, k-$landscapes (Kauffman, 1993) for $n = k$.

Whether a search problem has structure depends on the predictive capacity of the search algorithms one is willing to consider. For example, a problem that assigns high fitness to the prime numbers only has structure to the extent that the search algorithm can distinguish between prime and non-prime numbers. As another example, if correct solutions in a problem contain above average fitness schemata, as is implicitly assumed in the application of the standard genetic algorithm, then algorithms favoring above average fitness schemata can exploit this structure.

Structure in a problem can be exploited by forming modules that represent the patterns present in the problem. For example, consider a problem for which a subset of ten of its binary variables has only twenty useful settings. If this pattern is detected and used, the search can be restricted to combinations with these 20 settings, rather than considering combinations with all 2^{10} settings.

The above makes clear that identification of patterns by finding useful modules can potentially greatly reduce the fraction of the search space that needs to be visited. Following this idea, an important consideration is which type of patterns an algorithm should be able to detect. Section 4 describes the general setup of a method that forms modules. The patterns it is meant to detect are described in section 11. First, in the next section, two different notions

of modularity and the relation between them are discussed.

3 Structural versus Functional Modularity

The current treatment focuses on variable length sequential problems, and we therefore choose to consider partial solutions that take the form of a sequence of primitives. Such a sequence will be called a module. Since a module is a sequence, the positions of the primitives it contains are relative to the context in which the module is employed.

We distinguish between two types of modularity in the context of search algorithms: structural and functional modularity. *Structural modularity* is a property of individuals, independent of a problem. It refers to the way individuals are represented by the search algorithm, as will be detailed below. *Functional modularity* is a property of the problem, and refers to the existence of subsets of the variables whose optimal setting is independent of other variables, or has a reduced dependency. The existence of functional modularity in a problem can render large problems feasible. When the structural modularity used by a search method corresponds to the functional modularity in the problem, the structure in the problem is exploited, and the problem can be addressed in a more efficient manner. Below, we will discuss these two forms of modularity in detail.

Watson (2002) defines notions of structural and functional modularity based on the physical structure of a dynamical system and its behavior. Both these and our definitions consider the dependencies between elements in a system; a distinction is that here we will distinguish between the modularity of the representation used by a method (structural) and the modularity present in the problem (functional).

Structural modularity relates to the origin or to the representation of an individual. An individual exhibits structural modularity if one or more subsets of its elements have been represented explicitly during the search process that led to its creation, or if its current specification contains entities that represent such subsets. For example, if the individual $AABB$ is represented by a search method as XBB where $X = AA$, its representation is said to feature structural modularity. The specifications XBB and $AABB$ are equivalent in that they refer to the same sequence of elements ($AABB$). However, the significance of a structurally modular representation follows from its use. For example, where a method searching for variants of $AABB$ may consider point-wise mutations such as $AACB$, a single mutation to the modular representation XBB might replace $X = AA$ by $Y = CC$, leading to $CCBB$. The potential benefit from modular representations is determined by the extent to which modules represent useful, non-random combinations of elements whose functionality is valuable in multiple contexts.

While any choice of elements grouped together into modules leads to structural modularity, the benefit of employing modularity will strongly depend on the particular modules that are chosen. Ideally, the modules constructed by an algorithm should correspond to functional modules present in the problem. We will now discuss the notion of functional modularity.

The primitives of a problem are the basic elements that may occur in a genotype. They can be numerical values, e.g. $\{0, 1\}$, or actions or operators. A **module** is a sequence of primitives, and has a unique identifier. We will assume that for every primitive there is a module containing only that primitive. Thus, without loss of generality, individuals can be viewed as sequences of modules. Since complete individuals function as **contexts** in which modules are evaluated, they will be referred to with this term.

The *compact representation* of a context describes the context in terms of the modules of which it consists. Its *expressed* form consists of the concatenation of the sequences of primitives represented by its modules. The sizes of the compact and expressed representations of a context are called its compact and expressed size. These may differ greatly, reflecting the encapsulation of large partial solutions into modules that can be invoked by a single element of the compact representation. Since a context is interpreted by concatenating the expressed forms of its modules, the position of a module's primitives is determined by the preceding modules in the context in which it occurs. Thus, a module encodes the *relative* position of its primitives.

The modularity of a module is considered with respect to some set of contexts, called the **context set**. Furthermore, we will use the operation of replacing the module at a given position within a context by another module. A position used in this way will be called the *insertion point*.

Using the above concepts, a definition for functional modularity in variable length search problems can now be stated. Let $\boldsymbol{S}$ be a context set, and let $\boldsymbol{C}$ be a set of comparison modules.

Definition (**Functional Modularity**): A module A is *functionally modular* with respect to $\boldsymbol{S}, i$, and $\boldsymbol{C}$ if:

$$\forall S \in \boldsymbol{S} : \forall C \in \boldsymbol{C} : f(S(A,i)) \geq f(S(C,i)) \tag{1}$$

where $S(A,i)$ specifies placing module A at the i^{th} position of S, and $f(S)$ returns the fitness of a context S. Modules that are functionally modular are called functional modules.

The generality of a module is determined by the set of contexts for which it is functionally modular. The larger this set, the wider the applicability of the module. Ideally, the setting of a module's variables would be independent of all remaining variables. In this case, the module is functionally modular for all possible contexts, and we will call this a **perfect module**.

Definition (**Perfect module**): A module A is a perfect module if it is functionally modular for the set of all contexts.

The existence of perfect modules in a problem greatly facilitates the search process; if individuals consist of perfect modules only, the problem is separable. In this case, once the required modules have been constructed a solution can be found in linear time, as each module can be optimized independently.

While perfect modularity is rare, the concept of functional modularity is more generally applicable. The size of a module and the size of the context set define a *gradual* notion of functional modularity; the larger a module, and the context set for which it is functionally modular, the greater the utility of the module. This can be seen by considering the extreme cases: functional modules of size one for single contexts always exist, thereby answering the question whether modules can always be found. These do not facilitate the search however; they merely reflect the fact that when all but one variable of a context are defined, there exists a (not necessarily unique) best choice for the remaining variable. At the other extreme, modules that are functionally modular for all contexts are perfect modules, and a perfect module of maximal size is a complete correct solution to the problem. In between these two extremes are modules consisting of two or more primitives that are functionally modular for non-maximal context sets containing more than one element. The identification of modules that are functionally modular with respect to a large set of contexts can greatly benefit the search; for all such contexts, alternative settings for the variables of these modules can be safely disregarded.

4 Coevolution of Modules and Contexts

We study how the development of functional modules can be achieved in a setup where modules and contexts coevolve. A coevolutionary approach to module formation is obtained by using two populations, one containing modules and one containing contexts. Contexts are candidate solutions for the problem, and hence, their evaluation is given by the fitness function of the problem.

Modules are used in contexts, but since a context may contain multiple modules, the evaluation of modules is less straightforward; it involves a credit assignment (Minsky, 1963) problem. Intuitively, a module should be evaluated on the role it performs within a context. The following two sections describe how ideas from Pareto-Coevolution can be used to accomplish this.

5 Coevolution as Multi-Objective Optimization

In this section, we first describe the class of problems addressed by Evolutionary Multi-Objective Optimization (EMOO), a recently developed branch within evolutionary computation. Pareto-Coevolution is discussed in the next section, and uses EMOO to provide a theoretical basis for evaluation in coevolution.

5.1 Evolutionary Multi-Objective Optimization

Standard evolutionary algorithms use a fitness function that assigns a single scalar value to each individual. In general however, the quality of an individual may involve several aspects, called *objectives*; a construction may e.g. be evaluated on its strength and its costs. While a weighting function can be used to compress multiple evaluation values into a single value, this results in a loss of valuable information. A central tenet of EMOO is that comparing the values of distinct objectives is to be avoided. An individual can be safely discarded in favor of another if it is *dominated* by that individual.

Pareto-dominance is defined as follows: Let individual x have values x_i for the n objectives, and let individual y have objective values y_i. Then x dominates y if and only if:

$$\forall i \in [1..n] : x_i \geq y_i \quad \wedge \quad \exists i \in [1..n] : x_i > y_i$$

In general, there may not be a single solution combining the highest obtainable values for all objectives. Thus, the solution to an EMOO problem consists of a set of individuals that satisfy the different objectives to different degrees, and which are incomparable. The solution concept in EMOO therefore is the *Pareto-optimal set*, the set of all possible individuals that are not dominated by any other possible individuals. Several early papers describe the idea of optimizing for multiple objectives in evolutionary computation (Schaffer, 1985; Goldberg, 1989). For more recent work, see e.g. (Fonseca & Fleming, 1993; Srinivas & Deb, 1994; Fonseca & Fleming, 1995; Van Veldhuizen, 1999; Coello, 2000; Deb, 2001; Laumanns, Thiele, Deb, & Zitzler, 2002)

5.2 Pareto-Coevolution

While coevolution has produced a number of tantalizing results (Hillis, 1990; Sims, 1994; Juille & Pollack, 1996; Pollack & Blair, 1998; Juille & Pollack, 1998; Ficici & Pollack, 2001),

the dependence of fitness on evolving individuals can lead to instable results (Cliff & Miller, 1995). This instability may be the result of a lack of theory on which to base evaluation. Recently, a candidate for such a theory known as *Pareto-coevolution* has gained momentum (Ficici & Pollack, 2000; Watson & Pollack, 2000; Bucci & Pollack, 2002; De Jong & Pollack, 2002).

The central idea in Pareto-coevolution is that the outcomes of interactions with other evolving individuals should be viewed as *objectives*, in the sense employed by EMOO. Pareto-coevolution uses more specific information from individuals than a compounded average score; by treating interaction outcomes separately, better informed choices can be made regarding which individuals to keep and which to discard.

As an example, let us consider a chess-player that is evaluated by playing against ten individuals from a coevolving population. In standard coevolution, the average score in these ten games might be used as a fitness value. In Pareto-coevolution, the outcomes against the different opponents would be treated as separate objectives, resulting in a vector of objective values. A player winning its games against the first five opponents but losing against the second five would thus be very different from a player with the opposite outcomes under Pareto-coevolution; in standard coevolution they would be indistinguishable however as both would receive a fitness score of 50%. By employing the additional information contained in the assignment of the scores, the different nature of the players can be detected, and used to maintain a diverse population.

Here, contexts will provide objectives for modules. Thus, instead of considering the average fitness of the contexts in which a certain module is used, contexts are considered individually. The following section discusses in detail how contexts can be used to provide objectives for module evaluation.

6 Module Evaluation: Contexts Provide Objectives

This section shows how the Pareto-coevolution view leads to a principle for the evaluation of modules. A question is how this evaluation relates to functional modularity. It will be seen that optimization of the objectives corresponds to the optimization of functional modularity. This provides a connection between Pareto-coevolution and functional modularity.

6.1 Objectives from Pareto-Coevolution

Contexts provide situations in which modules can perform useful roles, and thereby implicitly define objectives. The maximal set of objectives that can be considered for a module therefore consists of the union of the objectives defined by the contexts in some set of contexts, e.g. the coevolving context population. For an individual context, the objective of a module in it is to contribute to the context's fitness by performing a useful role in it. Contexts define a number of positions at which modules can perform a useful role. Thus, in a context containing n modules, each of the n positions defines an objective. For a module A, the value of the i^{th} objective of context S is given by using i as an insertion point, and considering the fitness of the context resulting from placing A at the insertion point:

$$f(S(A, i))$$

The numerical value of this objective equals the fitness of the complete context, and is not

informative by itself. The logic behind this choice of objectives becomes clear when *comparing* the values it assigns to different modules. Intuitively, a module A is more valuable than another module B for a given context if using A instead of B has a positive effect on the overall fitness of the context. This is precisely what is measured when the objective values of two modules A and B are compared; A has a higher objective value than B for position i of a context S if

$$f(S(A,i)) > f(S(B,i))$$

This comparison turns out positive for A if A, when replacing B at the i^{th} position of S, results in a higher overall fitness for the context.

Using individual contexts as objectives allows for the identification of many different specialized roles or tasks. A module can in principle be valuable even if it is only used by a small number of contexts, or if only part of the contexts employing it have high fitness. This distinguishes this method from earlier approaches where modules are evaluated based on the average fitness of contexts in which they occur or on the frequency with which they occur in contexts.

Evaluation by replacing a module with other modules and comparing the overall fitness is a form of *differential fitness comparison*. This principle has been used in various forms in Cooperative Coevolution (Potter & De Jong, 2000), Baum's Artificial Economy (2000), the Wonderful Life Utility in COIN (Tumer & Wolpert, 2000), and SEAM (Watson & Pollack, 2000).

6.2 Relation between Pareto-Coevolution and Function Modularity

The previous subsection has shown how using Pareto-coevolution, contexts can provide objectives for modules. An important question is how the resulting objectives relate to the earlier notion of functional modularity. Below, we show that there is a direct correspondance between these, by demonstrating that maximizing functional modularity is equivalent to maximizing all objectives for a population containing all possible contexts.

Theorem (**Correspondence between Functional Modularity and Objectives from Pareto-Coevolution**): Let A be a candidate module, chosen from a set of all possible candidate modules $\boldsymbol{C}$. Let $\boldsymbol{S}$ a set of contexts, and i an insertion point. Then A is functionally modular with respect to $\boldsymbol{S}, i$, and $\boldsymbol{C}$ if and only if A simultaneously maximizes over $\boldsymbol{C}$ all objectives specified by $\boldsymbol{S}$ and i.

Proof (): The proof follows directly from the earlier definitions. Let A be a module that is functionally modular with respect to $\boldsymbol{S}$ and i. Then according to definition 3, A is functionally modular with respect to $\boldsymbol{S}, i$, and if and only if:

$$\forall S \in \boldsymbol{S} : \forall C \in \boldsymbol{C} : f(S(A,i)) \geq f(S(C,i)) \tag{2}$$

Now consider the objectives specified by $\boldsymbol{S}$ and i. As defined in the previous section, these are given by:

$$f(S(A,i))$$

for all $S \in \boldsymbol{S}$. A maximizes these objectives simultaneously over $\boldsymbol{C}$ if and only if:

$$\forall S \in \boldsymbol{S} : \forall C \in \boldsymbol{C} : f(S(A,i)) \geq f(S(C,i)) \tag{3}$$

Equation 2 and 3 are identical, demonstrating the equivalence.

6.3 Practical Issues in Module Evaluation

We now consider the set of objectives that play a role in the evaluation of candidate modules. This set depends on the current population of contexts. For each context, the set of objectives defined by it is obtained by taking the compact form of the context, and leaving out each of the $\frac{n^2+n}{2}$ subsequences of this sequence in turn.

To reduce the computational requirements of evaluating a candidate module on every context and insertion point, we will only consider contexts in which the candidate module actually occurs, as these are most likely to be relevant in evaluating the module. Candidate modules are identified by considering consecutive pairs of modules that occur frequently in contexts. By selecting one of the contexts in which such a candidate module occurs, a context is obtained and thereby an objective value for the candidate module. For example, if the combination AB is frequent, this candidate module can be evaluated by comparing it to other modules when placed at the insertion point $(\cdot)$ in $C \cdot C$.

The candidate module is evaluated by comparing its value for the objective to that of other possible candidate modules in a comparison set $\boldsymbol{C}$. The candidate module is only accepted as a new module if its value for the objective is equal or greater than all alternatives, and strictly greater than some alternatives. Thus, for given S, i, and $\boldsymbol{C}$:

$$\forall C \in \boldsymbol{C} : f(S(A,i)) \geq f(S(C,i))$$
$$\exists C \in \boldsymbol{C} : f(S(A,i)) > f(S(C,i))$$

This condition ensures that no better candidate is available, and that the candidate is an improvement over alternative combinations of modules.

To approximate the Pareto-optimal set, it is important to evaluate individuals on multiple objectives, typically using Pareto-dominance as a criterion; other schemes such as a weighting of objectives cannot achieve this set in general (Deb, 2001). Here in contrast, our aim is to identify functional modules, i.e. modules that maximize the fitness of a contexts for one or more contexts, where each such fitness is an objective. These modules are at the extreme of one or more dimensions in the feasible region of the objective space. Thus, modules that achieve suboptimal performance in some of the objectives need not be considered. The algorithm considers single contexts (objectives) at a time, and adds modules to the module population only if they maximize the objective. Thus, for any context objective, individuals that maximize it can in principle be found, and evaluation based on a single objective is sufficient for our purposes here. In summary, while the idea from Pareto-coevolution of considering performance on individual objectives is essential in module evaluation, it is sufficient to evaluate modules on single objectives at a time.

While maximizing the fitness of a single context is sufficient to achieve functional modularity for that context, additional benefit is gained when a module is functionally modular for several contexts. A possible approach to encourage this would again be to evaluate modules on multiple contexts, as considered above. This would greatly increases the computational costs however, as many comparisons may have to be performed for a single context already. A simple technique used by the algorithm to promote modules that function in different contexts, is to consider module combinations that occur frequently in contexts; since contexts are evolved on fitness, such modules are likely to play a role in achieving high fitness. It is important that this criterion is not *sufficient* for module acceptance however; it only serves to

select candidates for new modules, and a strict test must be satisfied in order for the module to be accepted, as will now be described.

A question is to what alternatives a candidate module should be compared during evaluation. In principle, an infinite number of possible sequences of primitives can be inserted into the context within which a candidate module is evaluated. First, to avoid unnecessary growth of genetic material, additional genetic material should be motivated by additional functionality. This relates to the issue of bloat in genetic programming (Smith, 1980; Blickle & Thiele, 1994; McPhee & Miller, 1995; Langdon & Poli, 1998; De Jong & Pollack, 2003a). Thus, a candidate module should be compared to alternatives of the same or smaller expressed length. This choice for the comparison set corresponds to requiring that modules may not contain unnecessary material.

The resulting scheme would not yet result in a scalable algorithm, as the number of required comparisons would grow exponentially with the size of modules. However, candidate modules can themselves only be combinations of existing modules. Thus, alternative combinations of *existing modules* provide a reasonable basis for comparison. The maximal set of comparison modules $\boldsymbol{C}$ then consists of all combinations AB of two existing modules, where $A, B \in \boldsymbol{M}$, with the requirement that the length of a comparison module may not exceed that of the candidate module.

A further efficiency improvement can be made by viewing the constituents of a module AB as modules; thus, A must be modular in the given context including B, and B must be modular in the given context including A. This final reduction makes the number of required comparisons linear in the number of existing modules $|M|$. We thus compare a candidate module AB to all modules A* and *B, observing the above length requirement, and only accept it if it obtains at least as high fitness as all of these and higher fitness than at least one of these. A final requirement is that the fitness of the context is at least as high as when either or both a module's contituent modules are left out; that is, the module is also compared to A, B, and [].

7 The DevRep Algorithm

DevRep()

```
1.  modules:=primitives ;
2.  contexts:=generate_random_sequences(modules);
3.  while(¬stop_criterion)
4.        create_modules(contexts, modules);
5.        for i=1:interval
6.              evolve_contexts(contexts, modules);
7.  end
```

Figure 1: Basic cycle of the algorithm

The choices that have been made regarding module construction and evaluation lead to an algorithm that develops a representation for the problem as part of the search, and which we will therefore call DevRep. An earlier version of this algorithm was discussed in (De Jong &

Oates, 2002a). The basic cycle of the algorithm is as follows, see figure 1. First, the population of modules is initialized to the set of primitives in the problem. The context population is initialized to random sequences of these initially available modules of a given length. Next, the following loop is repeated until a stop criterion is reached, e.g. solutions of satisfactory performance have been found: pairs of existing modules occurring consecutively in the contexts are considered for consolidation into new modules, and a generation of evolving the contexts is performed. The two steps will now be discussed in detail.

Create_modules does the following. Let AB be the pair of existing modules occurring most frequently in the context population. One context in which AB occurs is selected randomly. Let us write this context as $XABY$, where X and Y represent sequences of modules[3]. We now consider all contexts $XA \cdot Y$ and $X \cdot BY$ in which either A or B has been replaced by some other module, whose expressed length does not exceed that of $XABY$. Then the fitness $f(XABY)$ must be at least as high as that of all of these modified contexts, and higher than that of at least one of the modified contexts:

$$\forall Z: \ f(XABY) \geq f(XAZY) \wedge$$

$$f(XABY) \geq f(XZBY)$$

and

$$\exists Z: \ f(XABY) > f(XAZY) \vee$$

$$f(XABY) > f(XZBY)$$

If this is the case, we furthermore require that the compact representation of AB contains no unnecessary elements:

$$f(XABY) > f(XAY) \wedge$$

$$f(XABY) > f(XBY) \wedge$$

$$f(XABY) > f(XY)$$

If these requirements are met, the new module is given a unique ID, and added to the module population. Furthermore, all occurrences of AB in the current contexts are replaced by the new module, followed by a null module to maintain the same context length. If not, the next `max-modules-to-consider` most frequent pairs of modules are considered in order for consolidation until at most `max-modules-per-gen` new modules are found.

Evolve_contexts follows Mahfoud's deterministic crowding scheme (1995), and works by repeating the following cycle as many times as the context population size. Two contexts are selected randomly to function as parents. Offspring are produced by crossover with probability `pcross`, and by copying otherwise. The resulting contexts are mutated at each element with probability `pmut`. Mutation consists of replacing a module by a randomly selected module from the module population. Next, the parents are paired up with the offspring, such that the sum of the hamming distances between the compact representations of the parent-offspring pairs is minimized. Each offspring replaces its matched parent if its fitness is greater than or equal to that of its parent.

[3]The module AB may occur more than once in the context, in which case the replacements will concern all occurrences.

8 Variations on the Standard Method

To assess the necessity of different components of the algorithm, several variations and control experiments to the basic setup for context evaluation will be compared. These are described below.

8.1 Variation 1: Genetic Algorithm (No module formation)

The first control experiment is to see whether module formation as investigated here is at all helpful; if large search problems with structure can be solved with a conventional genetic algorithm, then there is no need to devise more involved methods. Therefore, in the first control experiment, we compare the performance of the method to a version where no additional modules can be formed. Thus, contexts are restricted to sequences of the available primitives. Contexts are simply evaluated based on fitness, as in the standard method, but are replaced when the offspring's fitness is greater than that of the parent; this was found to work better in preliminary experiments, and follows Mahfoud's deterministic crowding scheme. The resulting setup is a genetic algorithm that resembles the setup, except that it lacks module formation. An additional aspect that has to be changed is the length of contexts, since the maximum expressed size now equals the maximum compact size. In the experiments, sizes 100 and 200 were used, in additional to a variable length variant.

8.2 Variation 2: Random Contexts

The mechanism for evaluation and construction of modules is based on their role in contexts, and is therefore based on the problem-dependent fitness. Thus, an important test is to see whether fitness-based context selection is actually necessary. This will be investigated in a control experiment where module-construction is based on random contexts of modules, rather tha contexts evolved based on fitness. While one may wonder whether random contexts provide sufficient information for module evaluation, this principle can been used successfully in fixed length settings, as demonstrated by the SEAM algorithm (Watson & Pollack, 2000).

8.3 Variation 3: No Length Test

As described, candidate modules are are only compared to alternative modules of the same or smaller size. A question is whether this restriction is necessary. To test this, we use a variant where this length restriction is not applied.

8.4 Variation 4: Module Comparison Set

In the standard algorithm, a candidate module AB is only compared against $A*$ and $*B$, but not against $**$. To test whether making the additional comparisons is useful, we investigate a variant where these comparisons are made.

9 Test Problems

9.1 Hierarchical Test Problems

Several authors have recently studied the scalability of evolutionary algorithms (Thierens, 1999; Watson & Pollack, 2000; Watson, 2001; Pelikan & Goldberg, 2001). As part of this, several test problems have been designed that are difficult, yet contain structure that allows them to be addressed in principle. One such problem is the Hierarchical IF-and-only-iF (H-IFF) problem (Watson, Hornby, & Pollack, 1998).

The H-IFF problem can be described as follows. Let an individual be a bitstring of length n: $x_0, x_1, \ldots x_{n-1}$. Then we can view the individual at a number of different levels $l = 0, 1, \ldots {}^2log(n)$, where at each level l the individual consists of a sequence of modules of size 2^l. Formally, at level 0, we consider a set of 'modules' $x_0^0 \ldots x_{n-1}^0$ where $x_i^0 = x_i$, i.e. each module is a bit in the individual. For each level $1 \leq l \leq {}^2log(n)$, we define modules that each consist of a pair of consecutive modules of the previous level: $x_i^l = x_{2i}^{l-1} x_{2i+1}^{l-1}$ for $0 \leq i \leq n2^{-l}$.

Next, we define *target modules* for each level, see table 1. For level 0, we define two target modules A^0 and B^0:
$A^0 = 0$
$B^0 = 1$
For level $l + 1$:
$A^{l+1} = A^l A^l$
$B^{l+1} = B^l B^l$
Thus, at each level l, modules of length 2^l are defined consisting of all zeroes (A) or all ones(B). Using these definitions, we can define the fitness function for H-IFF. The fitness contribution at each level l is the number of target modules (A or B) present at that level, multiplied by a scaling factor of 2^l. The total fitness is obtained by summing the contributions of all levels:

$$f(x) = \sum_{l=0}^{l={}^2log(n)} 2^l |\{0 \leq i \leq n2^{-l} | x_i^l = A^l \vee x_i^l = B^l\}|$$

What makes the H-IFF problem difficult for genetic algorithms is that for each level multiple target modules exist, while a globally optimal solution requires coordinating the choices between these. For example, at level 1, a bit pair 01 can increase its contribution by changing to either 00 or 11, and so different bit pairs may converge to different choices. Only at the next level, which becomes relevant once modules at the current level have formed, correspondence between neighboring pairs becomes important. However, if the modules have formed by convergence, changing them become less likely, since this would require a number of intermediate changes that would temporarily decrease fitness. Without crossover and special mechanisms for diversity maintenance, it is unlikely for alternative module settings to be maintained, so that the modules at the next level can often not be formed. Given the dependence of high level contributions on finding the modules at previous levels, this can bar any further progress.

The difficulty of H-IFF changes greatly when modules found for some variables can be reused for other variables. For example, when the algorithm that has been described forms a module A=00, it can use this module multiple times in a single context, e.g. AAAA, to obtain fitness contributions at subsequent levels. Moreover, by recursively consolidating pairs of such modules, long strings of zeroes or ones can be formed very quickly. Thus, when primitives can be reused at different locations, the problem is no longer difficult. We have validated this

Level	H-IFF		H-XOR	
	A	B	A	B
3	00000000	11111111	01101001	10010110
2	0000	1111	0110	1001
1	00	11	01	10
0	0	1	0	1

Table 1: Target modules for the H-IFF and H-XOR problems. As in H-IFF, the two target modules at each level are composed of those at the previous level, and are each other's inverse. By using XOR instead of IFF, multiple modules are required at each level, making the problem more challenging for variable length methods.

observation by applying the method that has been described to the 64-bit H-IFF problem, which was solved within a few generations, or in a fraction of a second.

9.2 The Hierarchical Exclusive-OR Problem: H-XOR

To test scalable methods for variable length evolution, we describe a test problem that is analogous to H-IFF, but cannot be solved by the repetition of a single type of module. H-XOR was first mentioned in (Watson et al., 1998), and is based on the XOR function, rather than If-and-Only-If. While in a fixed-length setting H-IFF and H-XOR are of equal difficulty, in variable length settings H-XOR is much more difficult due to its reduced potential for exploiting repetitivenes.

Hierarchical eXclusive OR (H-XOR) problem is defined as above, but with the following target modules:
$A^0 = 0$
$B^0 = 1$
For level $l + 1$:
$A^{l+1} = A^l B^l$
$B^{l+1} = B^l A^l$
At the first level, 01 and 10 contribute to fitness, while 00 and 11 do not, in correspondence with the XOR function. Again, at each subsequent level, the constituent modules are those that contributed to the fitness at the previous level. Thus at the first level, A=01 and B=10, leading to 0110 and 1001 at the second level.

Both H-IFF and H-XOR have the interesting property that at each level, the first half of each module is the exact inverse of the second half, and these problems are the only hierarchically consistent problems with this property. However, whereas in HIFF correct modules can be created by repeating a single module indefinitely (e.g. AA, $AAAA$), in H-XOR multiple modules are required at each level. This makes the problem substantially more difficult for variable length problems. Thus, we believe H-XOR to be an interesting test problem for variable length methods, where primitives may be employed in different positions or for different variables. The maximum fitness for an n-bit version of $H - XOR$, where $n = 2^i$, equals $n(1 +^2 log n)$.

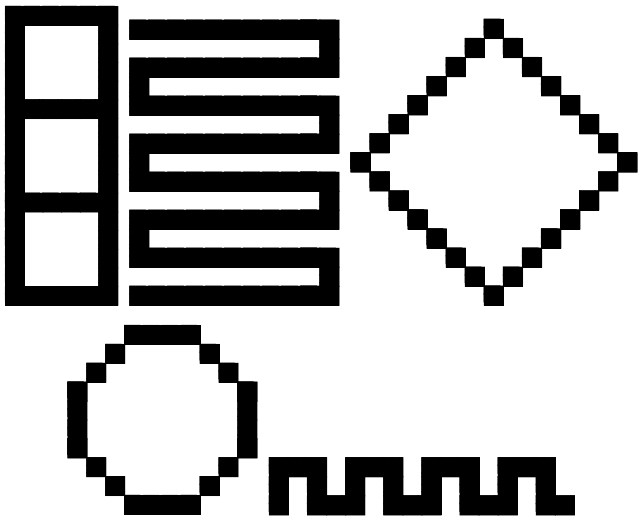

Figure 2: Target images used in the experiments: squares, path, diagonal square, octagon, and blockwave.

9.3 Pattern Recognition

In this second test problem, the goal is to generate a picutre using turtle graphics on a toroidal grid. The primitives for this problem are the following commands: TURN LEFT, TURN RIGHT, MOVE, and PUT PIXEL. The expressed form of a context is a sequence of these primitives. The interpretation of a sequence of primitives produces a bitmap, representing the concept specified by the context. The starting point for the interpretation of a sequence is the point from which the figure is drawn; thus, locating the target figure on the grid is not part of the task.

The target images are 16x16 bitmaps containing simple line drawings, see figure 2. The objectives are the number of black and white pixels correctly produced. As noted in section 2, a primary motivation for this work is to investigate methods for problems that require long solutions. Perfect solutions for these four problems require between 70 and 160 primitive operators, and thus satisfy this criterion.

10 Experimental Results

The settings of the experiments are as follows. Contexts are of length 2 (H-XOR) or 10 (pattern generation). The parameters used in the experiments (see algorithm description in section 7) are as follows: the context population size is 100, `max-modules-to-consider` = 5, `max-modules-per-gen` = 2, `interval` = 50, `pcross` = .9, `pmut` = .1.

Figures 3 through 4 show the performance on the H-XOR problem of size 64, 128, and 1024 bits. Since variable size evolution is used, it is important to take into account the size of individuals in measuring computational expense (De Jong & Pollack, 2003a). Therefore, we measure computational expense as the number of bits contained in the individuals that have been evaluated so far, rather than the number of individuals that have been evaluated.

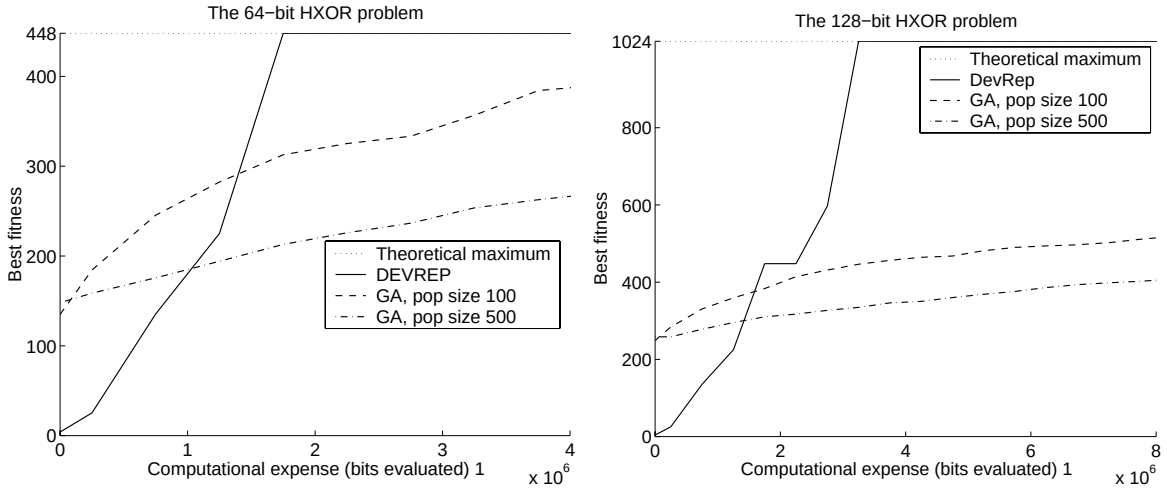

Figure 3: Performance on the 64-bit (left) and 128-bit (right) H-XOR problem

The algorithm that has been described achieves maximal performance on all three HXOR problems. In contrast, a genetic algorithm employing deterministic crowding achieves some progress, but is not able to find correct solutions for the population sizes used (100 and 500), and within the time given.

Next, we consider the results for the pattern generation tasks, see fig. 5-7. On all problems, the genetic algorithm variants are able to make *some* progress, but none of the algorithms were successful at solving any of the tasks completely. Apparently, the biases of the genetic algorithm do not correspond well to those required for these problems, which are characterized by long range dependencies and by sequential structure. The DevRep algorithm greatly surpasses the genetic algorithm methods on all of these problems. Average performance is substantially improved, and for all of the problems perfect solutions are found.

We now discuss the variations on the method that has been described. Variation 2, using random contexts, reaches a substantially lower level of performance. Apparently, using fitness to guide the exploration of different contexts is helpful in restricting the search. Indeed, since the number of possible primitive sequences is very large, and many contexts are possible in which modules can result in slightly better performance than others, without contributing to a solution of the problem. An example of this is a context in which an unnecessary or harmful action is taken, e.g. moving right. A module that cancels this action outperforms one that doesn't cancel the action. Thus, in the context of the harmful action, it would correctly be viewed as a useful module. This illustrates the need for goal-directedness in searching contexts for module-evaluation.

Random exploration of contexts has been used successfully in SEAM (Watson & Pollack, 2000). An explanation of the difference between these results, is that modules here encode *relative* position information rather than absolute position information. Thus, modules do not specify where they should be employed. This means that random combinations of modules will result in many contexts where potentially useful modules are wrongly arranged, making it difficult to identify new useful modules. The ordering information contained in the context population apparently is valuable to the evolutionary process.

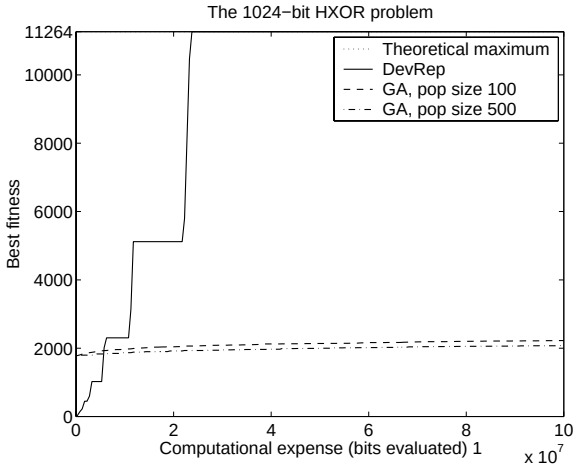

Figure 4: Performance on the 1024-bit H-XOR problem

Variation 3, not using the length test, achieves good performance on the problem, but not as good as the standard method. Thus, the length criterion plays a useful role in the algorithm.

11 Discussion

The formation of explicit modules may be seen as a way to make recombination respect useful combinations of variable settings that have been identified so far; this distinguishes the approach from conventional crossover, which will continue to explore disruptive combinations of alternative schemata indefinitely (Watson & Pollack, 2000).The usage of a short context length points to the role of contexts in representation development; rather than having to compare different combinations for all variables of a problem simultaneously, it is sufficient if good settings can be found for only a few variables at each point in time. Thus, contexts perform a sort of lookahead search (Juille, 1999) that identifies effective combinations of modules, which in turn can become new modules.

By reducing the fraction of the search space that will be visited, larger search problems can be addressed. Naturally, such reductions are beneficial only to the extent that a module does not require any further changes. Thus, the method may work well on certain problems, but not on all possible problems. Specifically, the approach is expected to be beneficial for large problems that can be decomposed into modules. The prevalence of modularity in both nature and design (Simon, 1968; Lipson, Pollack, & Suh, 2002; Ravasz, Somera, Mongru, Oltvai, & Barabási, 2002) suggests that this a large and important class of problems.

The Devrep algorithm shares several features with SEAM (Watson & Pollack, 2000), most importantly the use of recursive module formation, leading to hierarchy, and the use of Pareto-coevolution for module evaluation; the latter distinguishes these algorithms from other algorithms for representation development. Compared to SEAM, the main new contribution of DevRep is the application of this evaluation principle in a variable length setting. This has several implications. First, the set of modules from which contexts are built is not fixed,

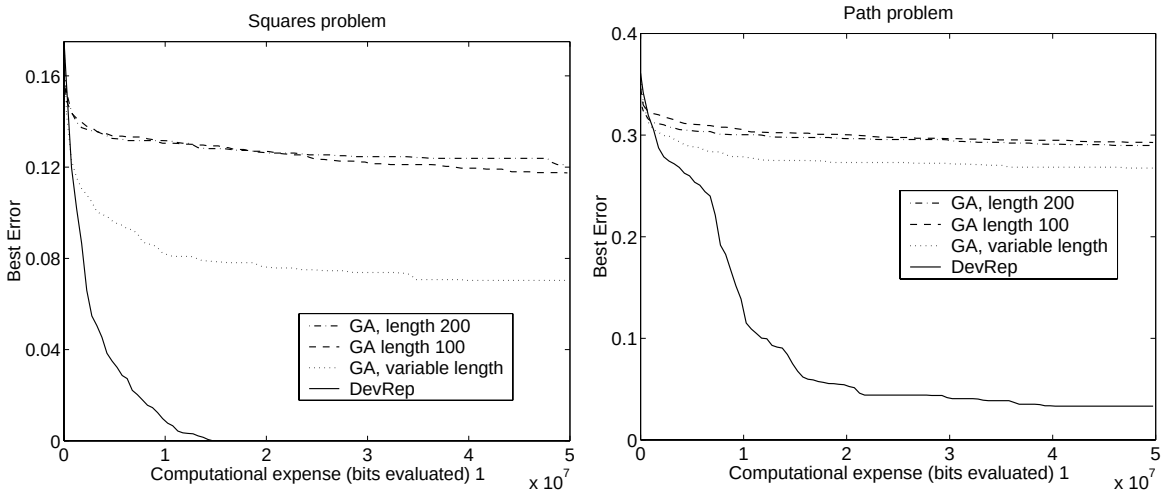

Figure 5: Performance on the *squares* (left) and *path* (right) problems as a function of the number of bits evaluated.

but open-ended; while module construction is subject to strict test to avoid the inclusion of unnecessary material, there are no limits on the size of modules when large modules afford a higher fitness. Likewise, and as a result, there is no limit on the expressed length of contexts. Furthermore, while the use of variable length representations can be combined with location-specific modules as demonstrated by the messy GA (Goldberg et al., 1989), in DevRep modules are sequences of other modules. Thus, while in SEAM and the messy GA the primitives contain a value and a position, in DevRep the primitives themselves are sufficient to initialize the module population. As a result, solutions can benefit from repetitive modularity. Due to relative positioning, a module has no control over its position within a context. Thus, random combinations of modules may use modules in uninformative ways. The use of a coevolving context population evaluated on fitness therefore appears a critical factor in the succes of the method, as confirmed by experiments.

A crucial idea in defining modularity for variable length problems was to consider modularity relative to specific subsets of all possible contexts. Without the possibility to consider such subsets, only separable modules can be detected, and problems such as H-IFF and H-XOR could not be addressed.

Search algorithms can be characterized by the types of patterns they are able to discover. In this section, we consider what type of patterns the algorithm that has been described may detect. These include the following:

- **Modularity** By maximizing the module objectives, the algorithm searches for modules that are functionally modular, as defined in section 3.
- **Hierarchical modularity** The principle of looking for useful modules is applied recursively, leading the algorithm to search for modules of hierarchical structure.
- **Repetitive modularity** Modules can be used repeatedly, i.e. a combination of two consecutive primitives or operators can be used multiple times within a single individual,

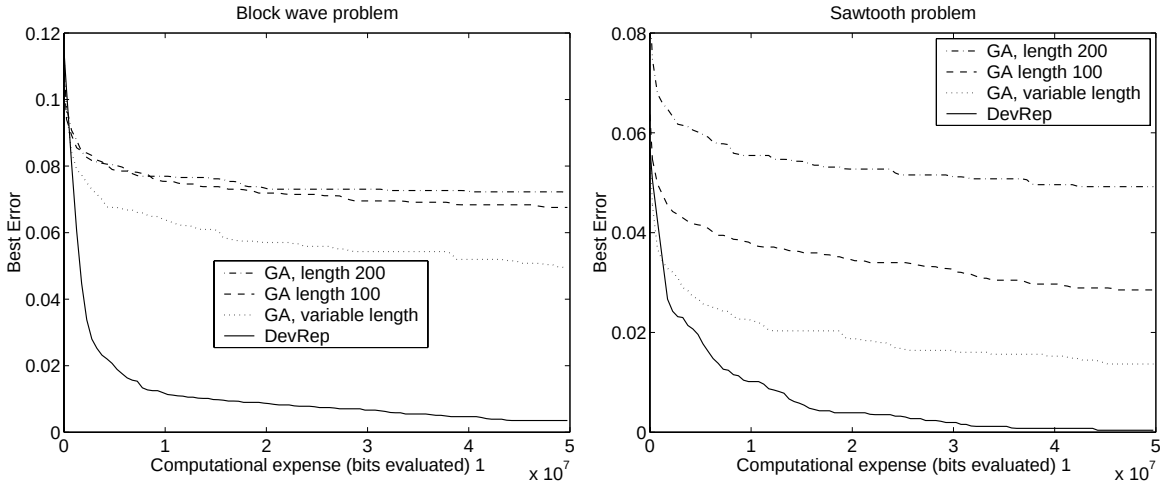

Figure 6: Performance on the *blockwave* (left) and *sawtooth* (right) problems as a function of the number of bits evaluated.

due to the use of position-independent coding.

The recursive combination of pairs of modules into new and larger modules provides a potential for increasingly long individuals, even though the compact length of contexts remains constant. Thus, a principle of Increasing Description Length (IDL) is used; if useful modules can be detected in small contexts, this greatly reduces the amount of computation required to find such modules. By incrementally considering larger contexts, the fraction of the search space that is visited while considering long individuals is highly biased compared to a randomly initialized search in the space of such long individuals. This idea of biasing the search using an incremental description length was used in SAGA (Harvey, 1992a, 1992b).It relates to incremental commitment (Watson & Pollack, 1999), which employs partial specifications in a fixed length setting, and to other forms of bias learning (Baxter, 2000; Gordon & (eds.), 1995; Baxter, 1995; Caruana, 1997; De Jong & Pollack, 2001; Peshkin & De Jong, 2002). Depending on the structure present in a problem, the recursive combination of modules can allow algorithms to quickly identify large, highly non-random individuals.

The limitations of the method are determined by patterns it cannot detect. At first sight, it would seem that only schemata involving consecutive variables can be consolidated into modules. However, since modules are formed recursively and often consist of other modules, a module involving non-adjacent variables can be constructed by repetitive module formation. For example, if A and C are related but occur as ABC in an individual, AB can first be consolidated into a new module D, after which DC can be combined into a module comprising both A and C. Nonetheless, while this presents some capacity to exploit long-distance dependencies, the restriction of candidates in module formation to consecutive variables clearly favors local dependencies.

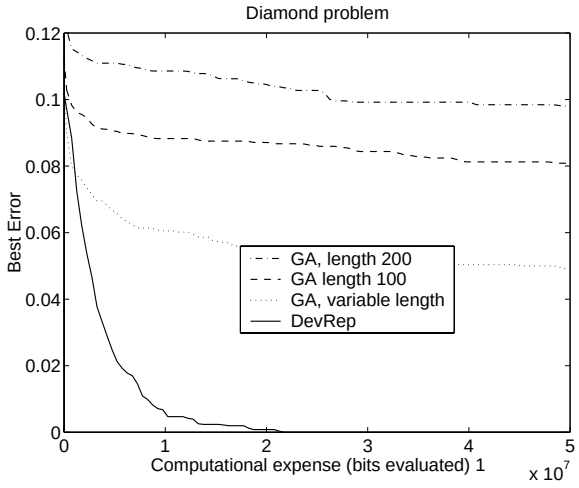

Figure 7: Performance on the *diamond* (left) and *octagon* (right) problems as a function of the number of bits evaluated.

12 Conclusions

Problems that require long solutions pose difficulties to standard genetic algorithms, due to the size of the associated search spaces. Still, if such problems have *structure*, they can in principle be addressed. While the idea to let modules and contexts develop simultaneously has been used by several authors, the evaluation of modules has so far not been addressed in a principled way. We aim to contribute to this by deriving objectives from Pareto-coevolution for module evaluation in variable length problems.

Functional modularity is defined, and it is shown that optimization of the objectives derived from Pareto-coevolution corresponds to optimization of functional modularity. Based on this evaluation principle, the DevRep algorithm for variable length problems is developed.

DevRep was tested on the Hierarchical XOR (H-XOR) problems up to size 1024, and on pattern generation tasks. It was found to develop large and ideal partial solutions and greatly improve performance compared to genetic algorithm methods used for comparison. The algorithm is able to exploit structure in certain large search problems, and may detect and exploit *functional modularity*, *hierarchical modularity*, and *repetitive modularity*. This is achieved by recursively forming modules and searching the space of combinations of such modules, thus forming modules in an open ended way. Several variants of the algorithm have been investigated; the conclusion from these control experiments is that all components of the method are necessary.

We conclude that certain forms of structure in certain large search problems can be exploited by gradually consolidating information extracted from the problem by the search method. By consolidating learned knowledge, the algorithm can explore subsequent choices without discarding earlier choices. Here, the patterns looked for by the algorithm take the form of *modules* (templates), but in principle any type of detectable pattern can be considered. According to this view, a challenge for research into large search problems is to identify the patterns present in problems of interest, and to develop corresponding algorithms exploiting

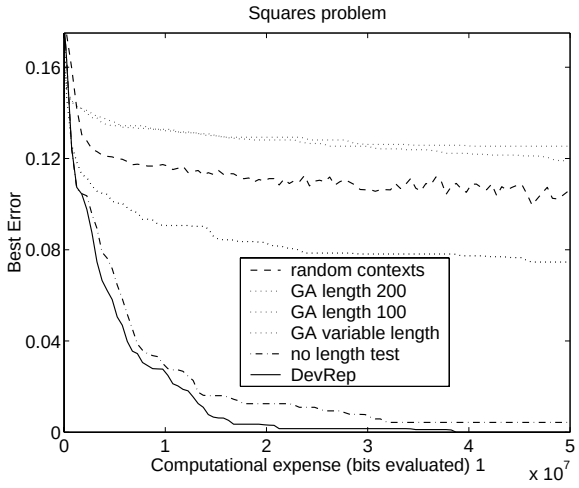

Figure 8: Performance of the DevRep algorithm and comparison methods, measured as a function of the number of bits evaluated.

those patterns.

Acknowledgements

This research was partially supported by a Talent-fellowship of the Netherlands Organisation for Scientific Research (NWO). The author wishes to thank NWO and the Agents Systems Group of the Vrije Universiteit Amsterdam for supporting this research, and members of the DEMO Lab at Brandeis, particularly Richard Watson, Sevan Ficici, and Anthony Bucci, for valuable feedback on earlier versions of this paper.

References

Angeline, P. J., & Pollack, J. B. (1992). The evolutionary induction of subroutines. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (p. 236-241). Hillsdale, NJ: Lawrence Erlbaum Associates.

Angeline, P. J., & Pollack, J. B. (1994). Coevolving high-level representations. In C. G. Langton (Ed.), *Artificial life iii* (Vol. XVII, pp. 55–71). Santa Fe, New Mexico: Addison-Wesley.

Baum, E. B. (2000). Evolution of cooperative problem solving in an artificial economy. *Neural Computation*, *12*, 2743-2775.

Baxter, J. (1995). Learning internal representations. In *Proceedings of the 8th annual conference on computational learning theory (COLT'95)* (pp. 311–320). New York, NY: ACM Press.

Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research*, *12*, 149-198.

Blickle, T., & Thiele, L. (1994). Genetic programming and redundancy. In J. Hopf (Ed.), *Genetic Algorithms within the Framework of Evolutionary Computation. Workshop at KI-94* (pp. 33–38). Saarbrücken, Germany: Max-Planck-Institut für Informatik (MPI-I-94-241).

Bosman, P. A. N., & Thierens, D. (1999). Linkage information processing in distribution estimation algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, & R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (Vol. 1, pp. 60–67). San Francisco, CA: Morgan Kaufmann.

Bucci, A., & Pollack, J. (2002). Order-theoretic analysis of coevolution problems: Coevolutionary statics. In *Proceedings of the 2002 genetic and evolutionary computation conference workshop: Understanding coevolution.* New York.

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*, 41-75.

Cliff, D., & Miller, G. F. (1995). Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In F. Morán, A. Moreno, J. J. Merelo, & P. Chacón (Eds.), *Proceedings of the Third European Conference on Artificial Life : Advances in Artificial Life* (Vol. 929, pp. 200–218). Berlin: Springer.

Coello, C. A. C. (2000). An Updated Survey of GA-Based Multiobjective Optimization Techniques. *ACM Computing Surveys*, *32*(2), 109–143.

De Jong, E. D., & Oates, T. (2002a). A coevolutionary approach to representation development. In E. de Jong & T. Oates (Eds.), *Proceedings of the icml-2002 workshop on development of representations.* Sydney NSW 2052: The University of New South Wales. (Online proceedings: `www.demo.cs.brandeis.edu/icml02ws`)

De Jong, E. D., & Oates, T. (Eds.). (2002b). *Proceedings of the icml-2002 workshop on development of representations.* Sydney NSW 2052: The University of New South Wales. (Online proceedings: `www.demo.cs.brandeis.edu/icml02ws`)

De Jong, E. D., & Pollack, J. B. (2001). Utilizing bias to evolve recurrent neural networks. In *Proceedings of the international joint conference on neural networks* (Vol. 4, p. 2667-2672).

De Jong, E. D., & Pollack, J. B. (2002). *Principled evaluation in coevolution.* (*In Submission*)

De Jong, E. D., & Pollack, J. B. (2003a). Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines*. (Accepted for publication)

De Jong, E. D., & Pollack, J. B. (2003b). Principled evaluation in coevolution. *Evolutionary Computation*.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms.* New York, NY: Wiley & Sons.

Ficici, S. G., & Pollack, J. B. (2000). A game-theoretic approach to the simple coevolutionary algorithm. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, & H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature - PPSN VI* (Vol. 1917). Berlin: Springer.

Ficici, S. G., & Pollack, J. B. (2001). Pareto optimality in coevolutionary learning. In J. Kelemen (Ed.), *Sixth European Conference on Artificial Life.* Berlin: Springer.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms, ICGA-93* (pp. 416–423). San Francisco, CA: Morgan Kaufmann.

Fonseca, C. M., & Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, *3*(1), 1–16.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Analysis, and first results. *Complex Systems*, *3*(5), 493–530.

Gordon, D., & (eds.), M. desJardins. (1995). Special issue on bias evaluation and selection. *Machine Learning*, *20*(1/2).

Gruau, F. (1994). *Neural network synthesis using cellular encoding and the genetic algorithm.* Unpublished doctoral dissertation, PhD Thesis, Ecole Normale Supérieure de Lyon.

Harvey, I. (1992a). Species adaptation genetic algorithms: a basis for a continuing SAGA. In F. J. Varela & P. Bourgine (Eds.), *Proceedings of the first european conference on artificial life. toward a practice of autonomous systems* (pp. 346–354). Cambridge, MA: The MIT Press.

Harvey, I. (1992b). The SAGA cross: the mechanics of recombination for species with variable-length genotypes. In R. Männer & B. Manderick (Eds.), *Parallel Problem Solving from Nature - PPSN II* (Vol. 2, p. 269-278). Amsterdam: North-Holland.

Hillis, D. W. (1990). Co-evolving parasites improve simulated evolution in an optimization procedure. *Physica D*, *42*, 228–234.

Juille, H. (1999). *Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm.* Unpublished doctoral dissertation, Brandeis University.

Juille, H., & Pollack, J. B. (1996). Co-evolving intertwined spirals. In L. J. Fogel, P. J. Angeline, & T. Baeck (Eds.), *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming* (pp. 461–467). Cambridge, MA: The MIT Press.

Juille, H., & Pollack, J. B. (1998). Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules. In *Proceedings of the third annual genetic programming conference.* Madison, Wisconsin.

Kauffman, S. A. (1993). *Origins of order: Self-organization and selection in evolution.* Oxford: Oxford University Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: The MIT Press.

Langdon, W. B., & Poli, R. (1998). Fitness causes bloat: Mutation. In W. Banzhaf, R. Poli, M. Schoenauer, & T. C. Fogarty (Eds.), *Proceedings of the First European Workshop on Genetic Programming* (Vol. 1391, pp. 37–48). Berlin: Springer.

Laumanns, M., Thiele, L., Deb, K., & Zitzler, E. (2002). Combining convergence and diversity in evolutionary multi-objective optimization. *Evolutionary Computation*, *10*(3), 263–282.

Lipson, H., Pollack, J. B., & Suh, N. P. (2002). On the origin of modular variation. *Evolution*, *56*(8), 1549–1556.

Mahfoud, S. W. (1995). *Niching Methods for Genetic Algorithms.* Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. (IlliGAL Report 95001)

McPhee, N. F., & Miller, J. D. (1995). Accurate replication in genetic programming. In L. Eshelman (Ed.), *Genetic Algorithms: Proceedings of the Sixth International Conference, ICGA-95* (pp. 303–309). San Francisco, CA: Morgan Kaufmann.

Minsky, M. L. (1963). Steps towards artificial intelligence. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and thought* (pp. 406–450). McGraw-Hill. (Originally published in *Proceedings of the Institute of Radio Engineers*, January, 1961 **49:**8–30)

Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. In L. Spector, E. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, & E. Burke (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001* (pp. 511–518). San Francisco, CA: Morgan Kaufmann.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (1999). BOA: The bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, & R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference* (Vol. 1, pp. 525–532). San Francisco, CA: Morgan Kaufmann.

Peshkin, L. M., & De Jong, E. D. (2002). Context-based policy search: transfer of experience across problems. In *Proceedings of the icml-2002 workshop on development of representations.* Sydney, Australia.

Pollack, J. B., & Blair, A. D. (1998). Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, *32*(1), 225–240.

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

Ravasz, E., Somera, A., Mongru, D., Oltvai, Z., & Barabási, A.-L. (2002). Hierarchical organization of modularity in metabolic networks. *Science*, *297*, 1551–1555.

Rosca, J. P., & Ballard, D. H. (1996). Discovery of subroutines in genetic programming. In P. J. Angeline & K. E. Kinnear, Jr. (Eds.), *Advances in Genetic Programming 2* (pp. 177–202). Cambridge, MA: The MIT Press.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and their Applications* (pp. 93–100). Hillsdale, NJ: Lawrence Erlbaum Associates.

Simon, H. A. (1968). *The sciences of the artificial.* Cambridge, MA: The MIT Press.

Sims, K. (1994). Evolving 3D morphology and behavior by competition. In R. Brooks & P. Maes (Eds.), *Artificial Life IV* (pp. 28–39). Cambridge, MA: The MIT Press.

Smith, S. (1980). *A Learning System based on Genetic Adaptive Algorithms.* Unpublished doctoral dissertation, University of Pittsburgh.

Srinivas, N., & Deb, K. (1994). Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, *2*(3), 221–248.

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, *7*(4), 331–352.

Tumer, K., & Wolpert, D. (2000). Collective intelligence and braess paradox. In *Proceedings of the 7th conference on artificial intelligence (AAAI-00) and of the 12th conference on innovative applications of artificial intelligence (IAAI-00)* (pp. 104–109). Menlo Park, CA: AAAI Press.

Van Veldhuizen, D. A. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* Unpublished doctoral dissertation, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Watson, R. (2002). *Modular interdependency in complex dynamical systems: recent results and discussion.*

Watson, R., & Pollack, J. (2000). Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, & H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature - PPSN VI* (Vol. 1917). Berlin: Springer.

Watson, R. A. (2001). Analysis of recombinative algorithms on a non-separable building-block problem. In W. N. Martin & W. M. Spears (Eds.), *Foundations of genetic algorithms* (Vol. 6). San Francisco, CA: Morgan Kaufmann.

Watson, R. A. (2002). *Compositional evolution: Interdisciplinary investigations in evolvability, modularity, and symbiosis.* Unpublished doctoral dissertation, Brandeis University.

Watson, R. A., Hornby, G. S., & Pollack, J. B. (1998). Modeling building-block interdependency. In *Parallel Problem Solving from Nature - PPSN V.* Amsterdam.

Watson, R. A., & Pollack, J. B. (1999). Incremental commitment in genetic algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, & R. E. Smith (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-1999* (Vol. 1, pp. 710–717). San Francisco, CA: Morgan Kaufmann.

Watson, R. A., & Pollack, J. B. (2000). Recombination without respect: Schema combination and disruption in genetic algorithm crossover. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, & H.-G. Beyer (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2000* (pp. 112–119). San Francisco, CA: Morgan Kaufmann.