CBR Assisted Explanation of GA Results

Sushil Louis[†] Gary McGraw[†*]
Richard O. Wyckoff[†]

[†]Department of Computer Science
[*]Center for Research on Concepts and Cognition
Indiana University, Bloomington, Indiana 47405
(812) 855-6486[†] (812) 855-6966[*]

Computer Science Technical Report number 361
CRCC Technical Report number 63

louis@cs.indiana.edu gem@cogsci.indiana.edu
rwyckoff@copper.ucs.indiana.edu

August 1, 1992

To appear in *Journal of Theoretical and Experimental Artificial Intelligence*

Abstract

This paper describes a system for explaining solutions generated by genetic algorithms (GAs) using tools developed for case-based reasoning (CBR). In addition, our work empirically supports the building block hypothesis (BBH) which states that genetic algorithms work by combining good sub-solutions called building blocks into complete solutions. Since the space of possible building blocks and their combinations is extremely large, solutions found by GAs are often *opaque* and cannot be easily explained. Ironically, much of the knowledge required to explain such solutions is implicit in the processing done by the GA. Our system extracts and processes historical information from the GA using knowledge acquisition and analysis tools developed for case-based reasoning. If properly analyzed, the resulting knowledge base can be used: to shed light on the nature of the search space, to explain how a solution evolved, to discover its building blocks, and to justify why it works. Such knowledge about the search space can be used to tune the GA in various ways. As well as being a useful explanatory tool for GA researchers, our system serves as an empirical test of the building block hypothesis. The fact that it works so well lends credence to the theory that GAs work by exploiting common genetic building blocks.

1 Introduction

This paper describes a system for explaining solutions generated by genetic algorithms (GAs) using tools developed for case-based reasoning (CBR). In order to make this paper accessible to researchers who may not know about both CBR and GAs, we have included two introductory subsections explaining each in basic terms. Subsections 1.1 and 1.2 may be skipped without loss of continuity.

1.1 Genetic algorithms

A genetic algorithm (GA) is a randomized parallel search method based on evolution. GAs have been applied to a variety of problems and are an important tool in machine learning and function optimization. David Goldberg's book gives a thorough introduction to GAs and provides a list of possible application areas (Goldberg 1989). The beauty of GAs lies in their ability to model the robustness and flexibility of natural selection.

In a classical GA, each of a problem's parameters is represented as a binary string. Borrowing from biology, an encoded parameter can be thought of as a gene, where the parameter's values are the gene's alleles. The string produced by the concatenation of all the encoded parameters forms a genotype. Each genotype specifies an individual which is in turn a member of a population. An initial population of individuals, each represented by a randomly generated genotype, is created. The GA starts to evolve good solutions from this initial population. The three basic genetic operators: selection, crossover, and mutation carry out the search. Genotypic strings can be thought of as points in a search space of possible solutions. They specify a phenotype which can be assigned a fitness value. Fitness values affect selection and in this way guide the search.

Selection of a string, depends on its fitness relative to that of other strings in the population. The genetic search process is iterative: evaluating, selecting, and recombining strings in the population during each iteration (generation) until reaching some termination condition. The basic algorithm, where $P(t)$ is the population of strings at generation t, is:

initialize $P(t)$
evaluate $P(t)$
while (termination condition not satisfied) **do**
    select $P(t+1)$ **from** $P(t)$
    recombine $P(t+1)$
    evaluate $P(t+1)$
    $t = t + 1$

Evaluation of each string (individual) is based on a fitness function that is problem dependent. This corresponds to the environmental determination of survivability in natural selection. Selection is done on the basis of relative fitness. It probabilistically culls from the population individuals having relatively low fitness. Mutation flips a bit with very low probability and insures against permanent loss of alleles. Crossover fills the role played by sexual reproduction in nature. One type of simple crossover is implemented by choosing a random point in a selected pair of strings (encoding a pair of solutions) and exchanging the substrings defined by that point. Figure 1 shows how crossover mixes information from two parent strings, producing offspring made up of parts from both parents. This operator which does no table lookups or backtracking, is very efficient because of its simplicity. One-point crossover, where A and B are the two parents, producing two offspring C and D, is shown in figure 1.

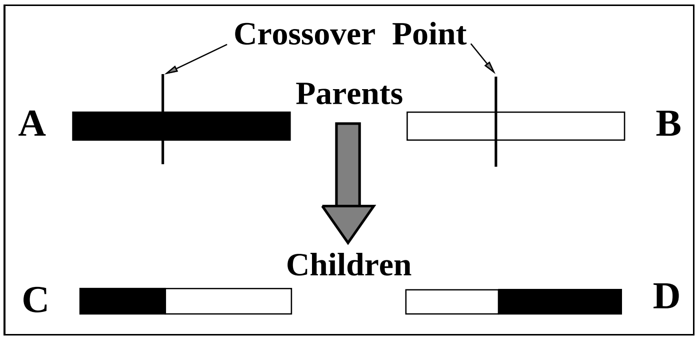

Figure 1: Single-point crossover of the two parents A and B produces the two children C and D. Each child consists of parts from both parents leading to information exchange.

The individuals in the population act as a primitive memory for the GA.

Genetic operators manipulate the population, usually leading the GA away from unpromising areas of the search space and towards promising ones, without the GA having to *explicitly* remember its trail through the search space (Rawlins 1991).

It is easiest to understand GAs in terms of function optimization. In such cases, the mapping from genotype (string) to phenotype (point in search space) is usually trivial. For example, in order to optimize the function $f(x) = x$, individuals can be represented as binary numbers encoded in normal fashion. In this case, fitness values would be assigned by decoding the binary numbers. As crossover and mutation manipulate the strings in the population thereby exploring the space, selection probabilistically filters out strings with low fitness, exploiting the areas defined by strings with high fitness.

Since crossover causes genotypes to be cut and spliced, it is impractical to consider or track individual strings in analyzing a genetic algorithm. Instead, the analysis must focus on substrings which define regions of the search space and are called schemas. The schema theorem will be discussed in subsection 1.3.

1.2 Case-based reasoning

Case-based reasoning (CBR) embraces the idea that reasoning and explanation can best be done with reference to prior experience, stored in memory as *cases*. The approach de-emphasizes reasoning from first principles. The strategy is first to amass and index a large and varied collection of examples, then, when presented with a new situation, to fetch and possibly manipulate the stored example which most closely resembles the new situation. Each stored case may be considered a previously evaluated data point, the nature and location of which is problem dependent. If the distance metrics have been well designed and the case-base includes sufficient variety, retrieved cases reveal useful conclusions from previous analysis. Because it may be impractical to store all prior experience in detail, CBR systems often include principled mechanisms for generalization and abstraction of cases. Note that except when comparing distinct cases, a CBR system need not include any notion of an underlying model, and that even during comparison one can make do without much of one. A great strength of the approach is that the generalizations and abstractions may in fact *induce* a useful domain model.

1.3 Applying CBR to GAs

Genetic algorithms (GAs), invented by John Holland in the early seventies, provide an efficient tool for searching large, poorly understood spaces often encountered in function optimization and machine learning (Holland 1975). GAs model natural selection, implemented through selection, recombination and mutation operators.

The probabilistic nature of GA search allows GAs to be successfully applied to a variety of **NP**-hard problems (Goldberg 1989; DeJong and Spears 1989; Jones and Beltramo 1991). The population-based, emergent nature of computation in GAs lends them a degree of robustness and flexibility that is often unobtainable using vanilla expert systems approaches. This fact has encouraged researchers to apply GAs to *creative* design tasks such as the design of semiconductor layout, aircraft engine design, and circuit design (Shahookar and Mazumder 1990; Bramlette and Cusic 1989; Louis and Rawlins 1991). This is not to say that GAs currently hold the answer to machine learning! The scarcity of GA theory is an alarming stumbling block in the way of progress.

Holland's *schema theorem* is fundamental to the theory of genetic algorithms. A schema is a template that identifies a subset of strings with similarities at certain string positions. For example, consider binary strings of length 6. The schema 1**0*1 describes the set of all strings of length 6 with 1's at positions 1 and 6 and a 0 at position 4. The "$*$" is a "don't care" symbol which means that positions 2, 3 and 5 can have a value of either 1 or 0. The *order* of a schema is defined as the number of fixed positions in the template, while the *defining length* is the distance between the first and last specific positions. The order of 1**0*1 is 3 and its defining length is 5. The *fitness* of a schema is the average fitness of all strings matching the schema. Remember that the fitness of a string is a measure of the value of the encoded problem solution, as computed by a problem-specific evaluation function. With the genetic operators as defined above, the schema theorem states that short, low-order, schemas with above average fitness increase exponentially in successive generations. Expressed as an equation:

$$m(h, t+1) \geq \frac{m(h,t)f(h)}{\overline{f}_t}\left[1 - (\text{probability of disruption})\right]$$

Here $m(h,t)$ is the number of schemas h at generation t, $f(h)$ is the fitness of schema h and $\overline{f}_t$ is the average fitness at generation t. The probability of disruption is the probability that crossover or mutation will destroy the schema h.

The building block hypothesis (BBH), which follows from the schema theorem, can be stated as follows:

> A genetic algorithm seeks near-optimal performance through the juxtaposition of short, low-order, high performance schemas or *building blocks* (Goldberg 1989).

Since both the order and defining length are syntactic measures, the BBH implies that syntactically-similar (short and low-order), high-performance individuals bias genetic search. This bias is achieved through the proliferation of specific nontrivial features of strong individuals.

Unfortunately, GA theory tells us little about the actual path a given GA will take through a space on its way to an answer. Because of the stochastic nature of GA operators, a solution's optimality is often suspect and its building blocks unknown.

Information about the building blocks is implicit in the processing done by a GA but is not documented explicitly or available to the user. At convergence, solutions may therefore be *opaque*, diluting confidence in the results and making analysis difficult. Keeping track of historical regularities and trends is key to acquiring the knowledge needed to explain *post facto* what sort of solution evolved, why it works, and where it came from. Discovery of building blocks allows principled partitioning of the search space. Empirical regularities can be formalized and possibly put to use. As noted in the GA introduction, genetic algorithms by their very nature trade the use of explicit memory for the speed and simplicity won by its omission. We use the methods and tools developed for case-based reasoning (Bareiss 1991; Riesbeck and Schank 1989) to extract, store, index and later retrieve information generated by a GA.

Case-based reasoning (CBR) provides a method of gathering knowledge in the form of cases (usually built from examples), indexing it in a meaningful fashion, and adapting it to hypothesize about the world. Individuals created by the GA provide a set of initial cases from which a well-structured case-base can be constructed. The BBH points to the use of fitness and genotype as metrics for indexing the case-base. These measures provide the key to indexing the cases and conveniently solve one of the hardest of CBR problems. For our system to work properly, the assumptions of the BBH must be true. Our working system provides empirical evidence in support of the building block hypothesis.

We apply CBR to GAs as an analysis tool in order to track the history of a search. The CBR system creates a case for each individual that the GA has evaluated. These cases are indexed on syntactic similarity and fitness. When properly formed and analyzed, this case-base can contribute to the understanding of how a solution was reached, why a solution works, and what the search space looks like. At the interruption (or termination) of a GA run, the case-base allows the interpretation and subsequent modification of discovered solutions.

If the system digests enough information during a run, it may be able to generate hypotheses about the space that are largely correct. Given proper hypothesis formation, the CBR module can guide the evolution more directly towards convergence. The existence of false hypotheses provides supporting evidence of deception in the current search space.

1.4 Motivation

Our prime motivation is to be able to explain (in an automatic way) the *why* of an evolved phenotype. Such a tool can be used to attack the following problems:

1. The utility of GAs as machine learning tools is attenuated by their inability to provide easily explained solutions. If GAs produce superior solutions which we do not fully understand, we may not gain much insight into their strengths or critical components. This problem is especially apparent in Koza's Genetic Programming work (Koza 1992).

2. It is common for a GA to search a space and find a solution without shedding any light on the nature of the search space itself. Such information would be very valuable in tuning GA search, and could be used to design highly tailored search strategies for future use in the current space or spaces thought to be similar.

3. When a GA converges to a sub-optimal solution, the post-processing which can be done to improve upon it is at present extremely lim-

ited.

4. GAs may become side-tracked into sub-optimal regions of the space. One cause of this behavior is known as *deception* (Goldberg 1989), in which the absolute best solution is hidden as an exception in an area which is generally inferior and would therefore be selected against. Side-tracking may also occur with *premature convergence*, where the domination of an early-discovered building block carries with it coincident suboptimal features of the earliest individuals which included it. In essence, the detracting features "hitch-hike" and gain popularity as parasites might.

2 The System

Our system is made up of a GA module which runs in tandem with a CBR analysis module [see figure 2].

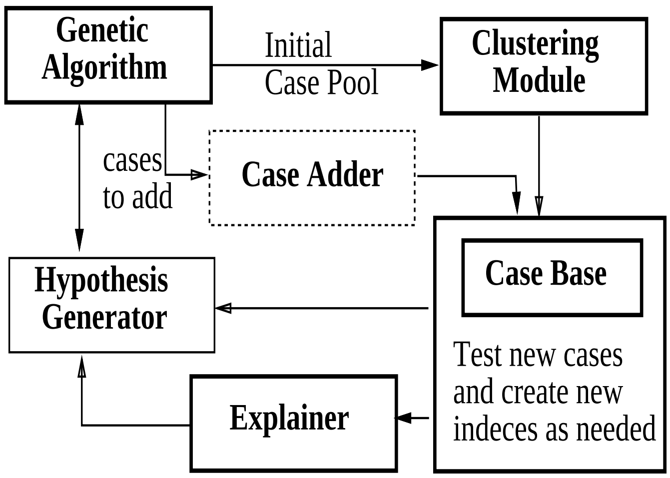

Figure 2: Schematic diagram of our system. The Case Adder is still under construction.

The GA module records data for each individual in the population as it is created and evaluated. This data includes a fitness measure, the genotype, generation data, as well as some information on the individual's parents. We call this collection of data the *initial case data*. Though normally discarded by the time an individual is replaced, all of the case data we collect is usually contained in any GA at some point and is easy to extract.

We generally employed very basic operators and standard parameter settings in our GA experiments. However, we do use uniform crossover (Syswerda 1989) in lieu of one-point crossover. Unlike the latter, this common variant is insensitive to the defining length of a schema. The order of the schema remains important.

After creating a sufficient number of individuals over a number of generations, the initial case data are sent on to a clustering program. We use a standard hierarchical clustering program which clusters the individuals based on both the fitness and the alleles of the genotype. This clustering constructs a binary tree in which each leaf includes the data of a specific individual. The binary tree structure provides an index for the initial case-base. Figure 3 shows a small part of one such tree. The numbers at the leaves of the tree correspond to the case number (an identification number) of an individual created by the GA. Internal nodes are denoted with a "$*$". An abstract case is computed for each internal node based on the information contained in the leaves and nodes beneath it. The final case-base includes: 1) cases corresponding directly to GA individuals (at the leaves) and 2) more abstract cases made up of information generalized from the leaves.

After investigating a number of clustering techniques and clustering sets (e.g., fitness and genotype data versus fitness alone) we found that standard hierarchical clustering on fitness and genotype data produces a coherent initial index for the case-base. The fact that fitness and genotype matter the most is predicted by the BBH and is borne out by our experiments. After creating an index, several easy computations are recursively performed to flesh out the abstract cases (these computations are explained in §3).

The current version of the hypothesis generator is still fairly preliminary, and many of the initial tests of the system reported in this article were run without it. The hypothesis generator runs in tandem with the GA. Using information from the the case-base, it engineers individuals for injection into the population. In a later version, information about the measured performance of such individuals, relative to the remainder of the population, will be fed back into the hypothesis module for analysis. The success or failure of generated hypotheses can be used in decisions regarding the validity of the current case-base.

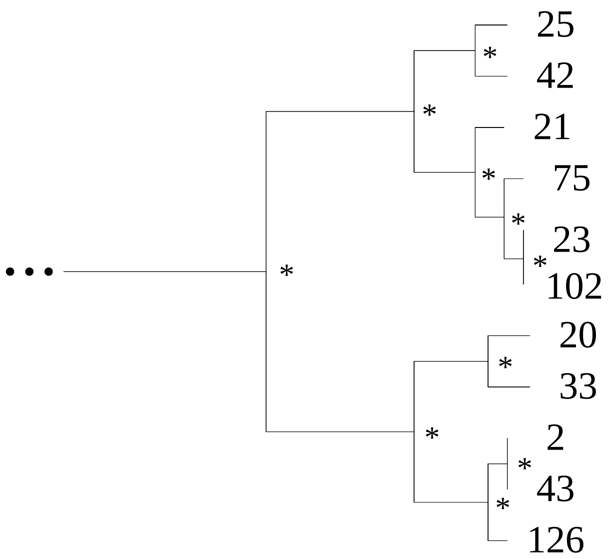

Figure 3: A small subsection of a typical binary tree created by the clustering algorithm. Individuals are represented by the numbers at the leaves. Internal nodes where abstract cases go are denoted with a "$*$". The binary tree provides an index for the case-base.

In order to support incremental updating of the case-base, we plan to install a case-adder module. The case-adder will become active only after the initial case-base is indexed. As the GA runs and new initial cases are created, an attempt will be made to fit these initial cases into the existing case-base by finding a place for them in the index tree. Periodically it may be necessary to reorganize at least parts of the case-base, but at all times it would be suitable for use by the hypothesis generator. This is discussed further in §7.3.

At the end of a run, a well-developed case-base can be used to explain how the GA came up with its answer, and why that answer is strong. Also available is knowledge about the problem space. The system can explain something about where the winning answer came from, which of its building blocks are the strongest, and which the weakest. This sort of data allows any future search of the space to be fine-tuned. It also allows for a sophisticated *post facto* analysis. Much of the knowledge that is usually discarded by a GA is made available in processed form for future manipulation.

3 The Case-Base

After clustering the first few hundred individuals using genotype and fitness values, a case-base is created. As explained above, clustering determines the indexing scheme of the case-base. All abstract cases, which are indexed at the internal nodes of the tree, must be recursively computed from the leaves up. Although much of the initial case data saved during the GA run is not used by the clustering algorithm, it is added to the cases in the case-base.

Cases include the following information:

- Case number
- Distance from the root of the tree to the level of the case
- Schema for the case (see discussion in §1)
- Schema order
- Average fitness
- Weight: Number of leaves (individuals) below
- Generation information: the earliest and latest leaf occurrence as well as the average in the subtree
- Additive schema: Bitwise sum of alleles of the genotypes of all the leaves below

Using the information supplied in the case-base, our system produces a report regarding the GA run so far. Using fitness as a metric, computing the top or bottom ranked n cases is easy, as is finding the top or bottom ranked n case schemas. These schemas can be combined to make new schemas by applying standard schema creation rules. For example, the two schemas **110**10 and **100***0 combine to the new schema **1*0***0. The resulting schemas show (among other things) which alleles are important in the genotype and which ones are not. Since order, longevity, and weight information is available, it is straightforward to find the top few cases with given sets of these characteristics.

We find it useful to calculate not only the schema which includes the subset of leaf cases, but one which records in a more democratic fashion the relative frequency of allele settings for the positions which are not absolutely fixed. Since the normal schema-producing rules result in a "$*$" wherever there is disagreement about an allele among leaves, schemas near the top of the tree tend to be filled with "$*$"s. To avoid this information loss, we compute an *additive schema* by computing the bitwise sum of the genotypes associated with the leaves below.

An *elected schema* can be computed by dividing the additive schema by the weight and squashing the results to 0, 1, or "$*$" using thresholds that can be changed. Elected schema information provides a more informative way of looking at upper-level schemas. Elected schemas play a large role in hypothesis generation discussed in §7.

A schema actually describes a convex hull for the leaves from which it was computed. That is, it describes a portion of the space within which the unaltered leaves fit. Since the normal schema-producing rules result in top-level schemas with a large number of "$*$"s and the hull tends to span too large a portion of the entire space, we prefer to describe a smaller hull, one for which the average hamming distance from an actual leaf to the hull surface is less than some predetermined constant. In other words, the leaves deviate from their closest encompassed relative by only a small amount on average. The elected schema describes this smaller hull.

Generation information can be used to determine when each part of the search space covered by the GA was considered. This information is very useful in detecting premature convergence, finding local minima, and computing the relative longevity of a subspace. The search space itself can be carved into approximately equally populated subsections by chopping the tree at nodes having a certain maximum weight. Analysis of these subsections across dimensions of fitness and longevity sheds light on the search space.

All of this processed information is available to the user of our system on–line. If desired, any case can be displayed. Also available is the tree information in graphical form with case numbers in their proper locations. The graphics are zoomable so that subtrees can be thoroughly investigated. The "report" is meant for user consumption and usually raises some questions which can be answered by pulling more information from the system.

It is important to emphasize that our system is an interactive *tool* for use in explaining GAs. Although it automates the information extraction and processing to some extent, human perception is a critical ingredient in the final analysis. However, without reasonable organization and pre-processing which our system provides, the large amount of data created during a GA run is impossible to digest even for an experienced GA researcher.

4 Results

Several experiments in function optimization and circuit design show that clustering techniques, if properly applied, provide a strong case-base, yielding useful data about the GA run. A list of problems on which we successfully tested our system is shown in table 1. We also ran a few experiments with the hypothesis module in place.

$f(x) = c$	c is a constant (control)
$f(x) = ones(x)$	number of ones in genotype
$f(x) = x$	signed, genotype length 10
$f(x) = x^2$	signed, g-length 10
$f(x) = x^3$	signed, g-length 10
$f(x,y) = x^y$	signed, g-length 20
$f(x) = 2^x$	all positive, g-length 10
$f(x) = log(x)$	all positive, g-length 10
$f(x,y) = x^2 - y^2$	all positive, g-length 20
$f(x_i) = \sum_i^3 x_i^2$	DeJong F1, $(-5.12..5.12)$
$f(x_i) = 100(x_1^2 - x_2)^2 + \sum_i^2 (1 - x_i)^2$	DeJong F2 $(-2.048..2.048)$
$f(x_i) = \sum_i^5 \text{integer}(x_i)$	DeJong F3 $(-5.12..5.12)$
A pair of complicated deceptive functions.	

Table 1: Test Problems.

The tree structure resulting from clustering defines a schema hierarchy [figures 4 and 5]. Schema of lower order are found near the top of the tree. Schema order gets higher and higher towards the leaves, which, with all alleles fixed, have the maximum possible order. Figure 4 depicts the index tree of an entire case-base for a simple function. Some salient features of the space are immediately recognizable on the graph (when one can zoom in and look at cases). We have marked these features in figure 4.

Figure 5 shows an enlargement of the small boxed area in figure 4 (labeled "Area of Detail"). In this figure, the positions of both initial cases (individuals) to the right of the figure and abstract cases at the internal nodes are represented by their schemas. Schemas closer to the root of the tree are more general (i.e., of lower order) than those towards the leaves.

A Simple Example: $f(x) = x$

Figure 4 shows the shape of the case-base index for a GA which maximizes the simple function: $f(x) = x$. The case-base was created by clustering the 400 GA individuals from the first 20 generations (which resulted in a binary tree) and then computing the abstract cases at the internal nodes. The resulting case-base had 799 cases, 400 of which came directly from the GA. By using the report mentioned above in conjunction with the index-tree graph we were able to find the salient features of the search space. We have marked these interesting features on the graph.

Clustering nicely partitions the individuals into sets sharing common features, namely specific allele values (syntactic similarity) and similar fitness. In this case, all the low fitness cases are clumped together toward the top of the graph. The high fitness cases are at the bottom. We included a sign bit in the genotype (values ranged from $[-511..511]$). The sign-bit branching point is shown in figure 4 as well. Not surprisingly, all the negative numbers are clustered together above the sign-bit branchpoint, and very little time was spent in that portion (half!) of the search space by the GA. This information was discovered by zooming in on the subsections of the graph and noting the obvious common features while consulting the on–line case-base.

Case zero is located at the root of the tree. Actual output from a query shows case-information:

```
> (show-case 0)
+-------------------------------------------------------------+
| Case: 0     | Distance: 0    Weight: 400   Fitness: 8441.44   |
|------------+ Order: 0       Schema: **********              |
| Additive schema: (396 333 170 283 338 330 20 72 332 392)     |
| Longevity: 19    (lo:1  avg:10.5  hi:20)                     |
| Left subtree: 1                          Right subtree: 396 |
+--------------------------------------------------------------
```

Since case 0 is located at the root of the tree, its distance from the root is 0. There are 400 leaves below it, making its weight 400. Fitness is calculated by averaging the fitnesses of the cases below. The case schema is filled with "$*$"s, showing once again why the additive schema information is important. Leaves under case 0 spanned all generations from 1 to 20 resulting in a longevity value of 19. Case 0 has two pointers, one to case 1 and another to case 396. This reflects the binary nature of the index. Had the case been a leaf it would not have pointed to further cases.

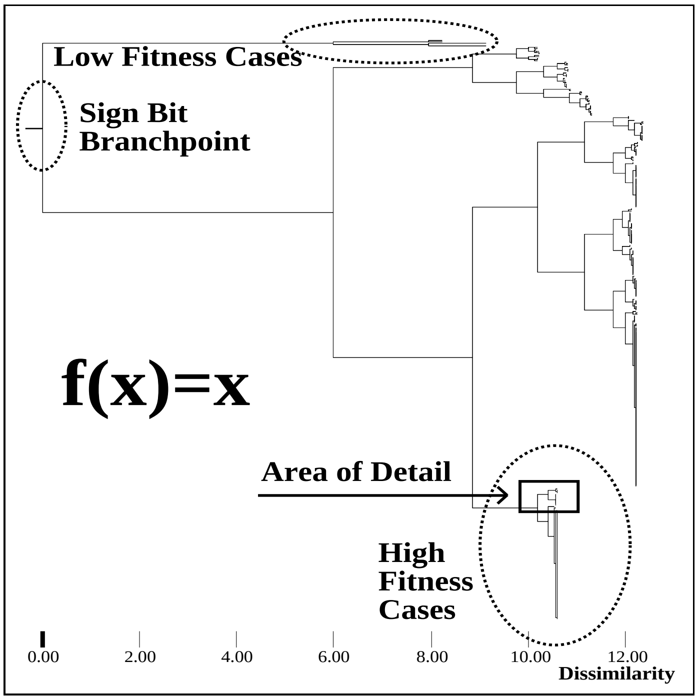

Figure 4: Tree structure of the cases. Nodes represent abstracted cases.

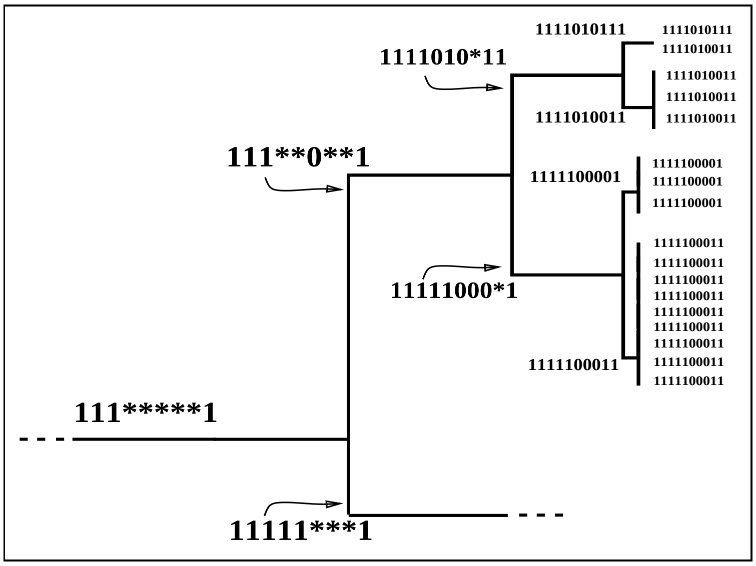

Figure 5: A closer look at the clustered cases reveals nested schemas.

We tested several very simple functions in order to discover just how much information the case-base can provide about the problem space. Salient features are apparent from the structure of the graph. This tendency could prove to be important in understanding black box fitness functions.

5 Methodology

We have found in some experiments that a useful high-level description of the case-base and the search recorded within it can be generated as follows:

1. Select a subset of the internal nodes which implicitly include all of the leaves, have no overlap, and are of fixed maximum size.
2. Sort these nodes by increasing average birth generation (age).
3. Report the available generational, weight, fitness, order, and elected case-schema information.

One can quickly see evidence of the GA's path through the search space. Many of the earlier subtrees may have low average fitness and low order, whereas later subtrees will have higher order if convergence was approached. Typically, later nodes have higher fitness, but as clustering is sensitive to genotype, it is possible to have a weak but young subtree.

Very low-order long-lived subtrees with poor fitness are often associated with crippled individuals formed by mutation or unfortunate crossover. If there are many weak subtrees one can infer a stronger focus on *exploration* of new parts of the space as opposed to *exploitation* of strong, higher-order schema.

High-order short-lived subtrees offer evidence of temporary concentration of search effort, which often follows discovery of a beneficial building block. Once a building block has been incorporated into the population (exploited), the search effort may move on to further exploration.

In problems where there is deception (that is, where most of the gradient in the space draws the GA *away* from the absolute global maximum but toward some broader, more obvious local maximum) one would expect to see the GA focus first on the local maximum. If the GA manages to discover points near enough to the global maximum that they are favorably evaluated, a paradigm shift may occur. This has been evident in our case-bases.

If one notices that a small but strong subtree was discovered but quickly lost and not surpassed, one might be well advised to alter the GA parameters in favor of a more elitist selection strategy (e.g., rank selection). Note that "Crowding" operators have been shown to be powerful in this regard, but they add overhead and are often unnecessary. When there is suspicion of deception, it may be confirmed or discredited by looking at the lower levels of the quickly lost cluster. If one discovers that within the cluster the later and stronger subclusters had schema changes which appear to cause relative *decline* in other contemporary clusters then one should continue to suspect deception. At this point one may wish to more thoroughly investigate that part of the search space without much competition.

6 Circuit Design

We tested our system on the *combinational circuit design problem*. This problem's search space is poorly understood, discontinuous, and highly non-linear. The problem can be stated as follows: given a set of logic gates to work with, design a circuit that performs a desired function. One instantiation of this problem is a *parity checker*, a circuit that returns the parity of a fixed number of bits (Louis and Rawlins 1991). When encoded for the GA, a 5-bit parity checking problem results in a 2^{80} sized search space having the troublesome properties listed above. We used a genotype of length 80, a population size of 20 and ran the GA for 25 generations. A candidate design is evaluated for fitness by testing a simulation on all possible inputs.

The GA does not solve the problem in 25 generations. In our experiments, a case-base was created by clustering the 500 individuals created by the GA and computing the 499 abstract cases. After examining the report and following the steps outlined in the methodology section, we examined the disjoint subtree list. The strongest such subtree (H) was also heavily weighted and spanned many generations. In addition, the best partial solutions all fell within H. We therefore suspected that a complete solution would fit H's schema. Since the order of the schema was appreciable (53 of 80), we could constrain the search to that schema and still hope to find a solution. The solution was in fact contained in H's schema.

Figure 6 shows the circuit indicated by our analysis of H. By decoding H, we were able to discern the general structure of a solution and determine

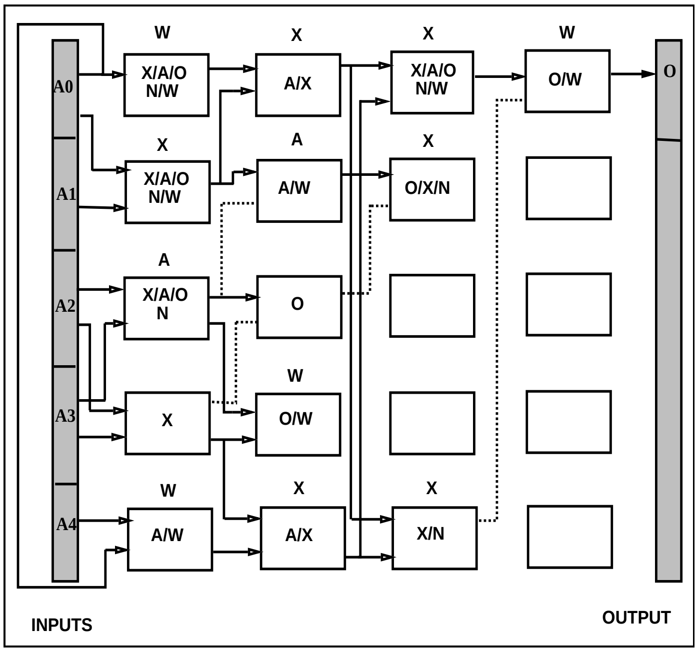

Figure 6: Parity circuit indicated by H, and the correct instantiation of choices.

which parts of it were important. The unlabeled boxes represent unimportant dimensions of the search space which can be safely ignored. Labeled boxes represent one of the logic gates e**X**clusive-or, **A**nd, **O**r, **N**ot, and **W**ire. Since we are drawing the circuit represented by a schema, multiple labels on the boxes show its possible instantiations. Labels above boxes represent a correct instantiation found near the 50^{th} generation of an unconstrained run. The top row should be considered adjacent to the bottom.

7 Hypothesis Generation

Preliminary work on hypothesis generation focuses on use of the heuristics sketched in the Methodology section (§5). The convex hull of each subtree selected by the method outlined there contains a very large number of possible individuals which have never been evaluated. The hypothesis module picks some specific individuals from within the hull of the most promising subtree and instantiates them for participation and evaluation within the GA population. Only a small number of generated individuals are injected in a given generation.

Since individuals which fit in the reduced hull of the *elected schema* (discussed in §5) may be considered more prototypical of the subtree than those which do not, the elected schema provides a better template for the generator than does the regular schema. As will be explained, hypotheses are in fact chosen from this reduced hull.

Our motivations for including a hypothesis generator are multifold. If hypothesis-produced individuals prove to be better on average than GA-produced individuals, the GA will be accelerated. They also provide evidence that produced solutions are reasonably well understood. Further support is furnished by the identification of building blocks.

Finally, consider cases where a small but strong cluster is discovered but quickly swamped out by outside evolutionary pressures. Injection of some new members of the "lost" subspace into the population would seem wise, as the undersampled space may deserve further investigation.

7.1 Experiments

To assess the ability of the hypothesis generation module to enhance the system, we compared the overall performance of the GA without hypothesis injection against the performance of a GA with hypothesis injection, both averaged over ten runs. The baseline case was a GA running with a 25% generation gap, where only 25% of the population is replaced each generation by new individuals. In a GA system with a generation gap, individuals may survive in the population for many generations, possibly being used for crossover more than once. Note that cases are only constructed for new individuals.

Since the case-adder module is not yet implemented, it was not possible to incrementally maintain the case-base, so injections in the experiments were perforce punctuated. In the first experiment, 1200 individuals were evaluated in total, with case-bases created in the same manner as before, but for three disjoint sets of 400 individuals each. The first case-base was created as soon as the first 400 initial cases were accumulated. Instead of allowing crossover and mutation to provide the partial population replacement in the immediately following generation, however, the hypothesis generator was employed. Following the application of our set of heuristics, which identify the most promising subspace, individuals from within the reduced hull (of elected schema) were generated at random and injected into the population. This procedure was repeated for the second set of 400 initial cases when they became available, and a final case-base for user interaction was created at the end of the run.

The second experiment had the same structure as the first, but for the hypothesis generation copies of previously evaluated individuals that fit within the *regular* schema of the subspace were selected at random and drawn into the reduced hull by flipping all rogue alleles. These massaged individuals retain the free alleles (those not fixed in the elected schema) of their originals and so include less variation than those produced for the first experiment. In the first experiment, the free alleles are set at random.

7.2 Results

In each of the solution spaces searched by the GA, hypothesis injection on average provided detectable and reproducible improvements. Even though injection was performed only twice per experiment, hamming distance to the optimal solution was decreased by more than ten percent of the problem size over the baseline case. In several instances, the solution was discovered within one generation of the injection. A GA converges when the average hamming distance between individuals in a population stabilizes. In both experiments the GA converged more quickly and to better solutions when hypothesis injection was used.

The speed improvement was most pronounced in the second experiment, though there was also a smaller variance in the injected individuals, and the final solution was not as notably improved.

Overall, the best solutions after 1200 fitness evaluations were products of the first experiment. There, injected individuals sampled promising subspaces with sufficient breadth to eventually locate superior solutions. Note that the abstract case defining the chosen subspace may include data from previous generations. As the injected individuals were prototypical of that case (from which the population may have since wandered) they may markedly increase the population's diversity. This diversity prevents convergence only temporarily. Following exploration, the superior individuals proliferate.

7.3 Future work

Clearly, hypothesis generation can be implemented in a more reasonable way once the case-adder module is in place. Much cleaner experimentation could be run given a continuously maintained case-base. Injection of hypotheses could be done in smoothly varying rates better tailored to the state of the search.

The hypothesis module should track the performance of its injected individuals and adjust its hypotheses accordingly. The successes and failures of a genetically engineered individuals can be used to make decisions regarding the validity of recent hypotheses and the rate at which they are to be injected.

If a selected promising subspace were in fact small, the generator could relax the constraint that injection candidates be drawn into a reduced hull. In such cases, the generator could choose to inject some individuals from some distance outside the hull.

The hypothesis generator should also take into account other promising areas of the search space. By discovering the important building blocks in a space (perhaps in different subtrees) and combining them, the hypothesis module could speed the search considerably. Though higher level abstract case schema are unlikely to have fixed bits, our elected schema may have quite a few. By adjusting the thresholds used to compute the elected schema we may be better able to identify building blocks. After producing a hypothesis as in the initial experiments, we could then alter it to include the hypothesized building block of some other strong subtree, which may have been explored at a different time.

As very strong but suboptimal building blocks may have swamped the population quickly, any final solution containing building blocks which were evident particularly early in the evolution could be genetically engineered to check for parasitic alleles.

Concerning the case-adder itself, we propose placing new cases in the case-base as special children of the closest subtree identified by our methodology. The initial classification task could therefore be performed by a variety of existing algorithms. When the weight of a subtree became excessive, we would invoke the clustering algorithm only on the leaves of this subtree. This limits the cost of reorganization and allows continuous use of the first-level classifier. When a new case was assigned to the reclustered tree, an additional level of classification would be performed. To save space, we could discard some of the lower level parts of a subtree since the abstractions being used have already been computed.

8 Conclusions

We have tested the model on a series of problems including the DeJong functions, a pair of home-brewed deceptive problems, and the circuit design problem. Results show that our CBR tools are useful for understanding GA solutions in many different kinds of spaces as exemplified by our choice of test problems.

We have also tested a preliminary version of the hypothesis generator. Hypotheses generated by CBR tools can provide principled direction to otherwise less-productive GA search as well as help in the avoidance of deception. Although it is possible to damage the search through hypothesis injection, this risk can be reduced by meta-level application of CBR techniques. In the future, the hypothesis generator should track the progress of genetically engineered individuals and update its knowledge base. Even without meta-level knowledge, cautious injection of hypotheses proves profitable in some cases.

Combining GAs with CBR allows us to successfully attack the four problems listed in §1.1. Our system provides a tool with which GA researchers can explain discovered solutions while simultaneously discovering important aspects of the search space. The CBR module is able to make explicit the building blocks used by a GA. Analysis of the building blocks and the path taken by a GA through a search space provides a powerful explanatory mechanism. Results of the explanation can be used to guide post-processing in order to improve results in case of premature convergence or no convergence. The system also addresses some aspects of GA deception. The hypothesis module provides a mechanism by which deception can be avoided. If a space is found to be too deceptive for GA search, other search methods can be suggested based on the analysis of the space provided by the CBR module.

As well as providing a system for GA explanation and search space analysis, our system provides empirical evidence that the building block hypothesis is correct. As mentioned above, the hierarchical clustering of a set of GA individuals (distributed over several generations) according to fitness and genotype leads to nested schemas. Analysis shows that the subtrees with the most fit schemas receive the most reproductive energy while the least fit schemas are culled from the population. Although we knew that clustering on fitness and genotype should yield nested schemas (as a result of the BBH), we nonetheless tested several other clustering possibilities, including more generational information and parental information. Clustering on fitness and genotype was by far the most successful technique. The nested schemas that result from such a clustering provide a coherent index for the case-base.

The general ability to gain insight into a problem and the path a GA took while searching for its solution may widen the applicability of genetic algorithms to problems which were previously inappropriate by virtue of having relatively expensive evaluation functions. Because principled constraints can be applied to the search periodically, fewer fitness evaluations need be computed. Following confirmation that the case-base accurately models the space through testing, we can avoid expensive fitness function evaluation in favor of case-based estimation.

Acknowledgments

Our thanks to David Leake, Terry Jones, Jim Marshall and Melanie Mitchell who commented on early drafts of this paper. Also thanks to Gregory Rawlins whose GA seminar provided the impetus for this work.

References

[1] Bareiss, R. (1991) *Proceedings of the Case-Based Reasoning Workshop*, Morgan Kaufmann, Inc.

[2] Bramlette, M.F. and R. Cusic. (1989) "A Comparative Evaluation of Search Methods Applied to Parametric Design of Aircraft." In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 213-218.

[3] De Jong, K.A. (1975) "An Analysis of the Behavior of a class of Genetic Adaptive Systems." Doctoral Dissertation, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

[4] De Jong, K.A. and W.M. Spears. (1989) "Using Genetic Algorithms to Solve NP-complete Problems." In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 124-132.

[5] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

[6] Goldberg, D.E. (1989) "Genetic Algorithms and Walsh Functions: Part II, Deception and its Analysis." *Complex Systems 3*: 153-171.

[7] Holland, J.H. (1975) *Adaptation In Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press.

[8] Jones, D.R. and M.A. Beltramo. (1991) "Solving Partitioning Problems with Genetic Algorithms." In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kauffman, 442-449.

[9] Koza, J.R. (forthcoming) *Genetic Programming*, MIT Press.

[10] Louis, S.J., McGraw, G.E. and Wyckoff, R.O. (1992) "Automating Explanation of Genetic Algorithm Results (two paradigms collide)." In *Proceedings of the Florida Artificial Intelligence Research Symposium (FLAIRS-92)*, April 1992, Ft. Lauderdale, FL.

[11] Louis, S.J. and G.J.E. Rawlins. (1991) "Using Genetic Algorithms to Design Structures." In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kauffman, 53-60.

[12] Rawlins, G.J.E. (ed.) (1991) *Foundations of Genetic Algorithms*, Morgan Kauffman, 1-10.

[13] Riesbeck, C.K. and R.C. Schank. (1989) *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates.

[14] Shahookar, K. and D. Mazumder. (1990) "A genetic approach to standard cell placement using meta-genetic parameter optimization." *IEEE Trans. Computer-Aided Design*, 9, May 1990, 500-511.