GAVEL—A New Tool for Genetic Algorithm Visualization

Emma Hart and Peter Ross

***Abstract*—This paper surveys the state of the art in evolutionary algorithm visualization and describes a new tool called GAVEL. It provides a means to examine in a generational genetic algorithm (GA) how crossover and mutation operations assembled the final result, where each of the alleles came from, and a way to trace the history of user-selected sets of alleles. A visualization tool of this kind can be very useful in choosing operators and parameters and in analyzing how and, indeed, whether or not a GA works. We describe the new tool and illustrate some of the benefits that can be gained from using it with reference to three different problems: a timetabling problem, a jobshop scheduling problem, and Goldberg and Horn's long-path problem. We also compare the tool to other available visualization tools, pointing out those features which are novel and identifying complementary features in other tools.**

***Index Terms*—Genetic algorithm visualization.**

I. Introduction

IT CAN BE EASY to quickly set up a naïve genetic algorithm (GA) to tackle a problem, but analyzing the results to discover whether or not the process is efficient or could be improved is often extremely difficult. The representation, the choices of operators, and the associated parameters can each make a major difference to the speed and the quality of the final result. These decisions interact, so it can be very difficult to disentangle their effects. Furthermore, keeping a full record of everything that occurs during a GA run produces large quantities of data that cannot be analyzed conveniently by hand. Ideally, the choices made when configuring a GA would not be too delicate so that the user would not have to do a lot of fine tuning in order to get acceptable performance, but it would be unwise to simply assume that this was true. What, then, can the cautious user do?

One common approach is to rely on rules of thumb that have been used by other people—an allelewise mutation rate equal to 1/length, a "standard" crossover operator, and so on. Another is to do a modest range of comparative studies, for example, turning off crossover to see if the performance drops significantly (or increases). However, Jones [22] has argued that this method is not sufficient to justify if crossover is useful, i.e., is facilitating the exchange of building blocks between individuals. Yet another is to try to automate the choice process in some way or to obviate the need for certain choices by making the GA adapt some parameters as it runs [10], [34].

Manuscript received June 9, 2000; revised October 31, 2000. The work of E. Hart was supported by the Engineering and Physical Sciences Research Council.

The authors are with the School of Computing, Napier University, Edinburgh EH14 1DJ, Scotland, U.K. (e-mail: emmah@dcs.napier.ac.uk; peter@dcs napier.ac.uk).

Publisher Item Identifier S 1089-778X(01)02488-2.

None of these are very satisfying from a scientific point of view; they treat each single run of a GA as a black box, not delving into what happens internally during the run. We propose that a more scientific approach to choosing operators and setting parameters can be used based on explaining and then extrapolating the empirical observations made during a GA run. With these objectives in mind, we describe an offline GA visualization tool designed specifically to address these points. There is already a sizeable body of published work on GA visualization (see [18] for some recent papers). In this paper, we aim to present a set of techniques that complement existing ones and provide a GA user with a step forward in analysis tools.

The paper provides some background to the field of GA visualization and surveys some of the literature on the subject in Section II. We then describe the new tool GA Visualization of Evolutionary Links (GAVEL) in Section III and the new features it provides for studying the performance of the GA. These features are illustrated with reference to using a GA to solve a real-world timetabling problem. The results of using the tool to analyze two further problems, both commonly solved by GAs, are then described in Section IV. Finally, we compare the new tool to other available visualization tools in Section V. Some of these tools provide complementary features which make them ideal to be used alongside GAVEL to provide a good set of tools for performing an analysis of a GA.

II. State of the Art

According to Shine and Eick [32], visualization of GAs can be helpful in analyzing the extent to which a GA explores the search space, analyzing its convergence behavior, enabling the user to get some feeling for the fitness landscape and the dynamics of the evolutionary process, and fine-tuning the GA. The first three of these have been addressed specifically in [8], [9], [32], and more recently in [18]. Some of the methods used are described in detail later in this section. Pohlheim *et al.* [28] suggested a small set of types of data to track over various time frames in order to observe both the *course* and the *state* of an evolutionary algorithm; these include alleles and fitness of individuals, distances between individuals, and certain statistics about subpopulations to be tracked over periods ranging from a single generation to multiple runs. They provide examples of some of the ways in which this can be done using two-dimensional (2-D) graphs, bar charts, and colored image plots, all implemented in MATLAB. Using visualization to help understand the dynamics of the GA is further addressed by both Collins [6] and Bedau *et al.* [2], whose contributions are described below. However, none of those methods address the question of just *how* the GA traverses the search space, i.e., which operators were used to reach different parts of the space, and whether different operators were important at different times during the evolution. Accessing this information would allow us to understand if a GA is the right algorithm to use and, if so, to fine tune the choice and settings of such operators.

Collins [9] attempted to address the question of fine tuning operators by using the visualization of the search space as a "hands-on" interactive process to guide the progress of the algorithm. This on-the-fly tuning can be very successful in finding good results from the algorithm, but again, does not enhance our understanding of *why* the algorithm works. Also, such interactive tuning is often open to the criticism that the algorithm is being adjusted to solve some specific problem. VIS, described by Wu *et al.* [35], [36], goes further in that the family history of individual chromosomes can be displayed (i.e., the parents and offspring of each individual) and crossover and mutation locations in each individual identified. This kind of information also forms a central part of our analysis tool described in this paper, which extends the idea by displaying the *complete* history across all generations of *chromosomes*, individual *genes*, and *schemas*.

The following subsections categorize existing visualization ideas under three broad headings: identifying population features, visualizing the space searched by the GA, and visualizing the problem's entire search space.

A. Identifying Population Features

The most wholesale idea is that of a population data matrix in which the entire population is displayed in textual form—the PGA program [27] provides this feature. For large populations or very long chromosomes, this is likely to be far too much information to absorb visually. An improvement is suggested by Collins [7] and by Routen and Collins [30] in the form of allele-loci frequency matrices. In these diagrams, a matrix of allele values against loci is displayed, with each element of the matrix being a circle whose size indicates the relative frequency of the allele. This is somewhat akin to Hinton diagrams in neural nets. In a similar vein, the authors have used histograms of loci (x axis) versus frequency (y axis) to display the frequency of a requested allele in a single population; the user can watch how the histogram changes as the generations pass. For example, in the one-max problem, it is possible to see that hitchhiking can cause zeros to spread through almost the whole population in one or two loci before (perhaps) finally being driven out of the population.

Both allele-frequency matrices and histograms fail to represent information about the structure of individual chromosomes and, hence, information regarding specific schemas, nor do they address the question of chromosome fitness. Perhaps the most commonly observed form of visualizing a GA is to plot graphs of fitness over time. These maybe either 2-D or three-dimensional (3-D) [19] in which fitness (y axis) is plotted against generation (z axis) and fitness-ordered position (x axis). The major drawback of such graphs is that they contain no genetic information regarding chromosome specifications or positions of chromosomes in the search space. This has been addressed partially by Fang *et al.* [16], who make use of 3-D graphs by plotting locus (y axis) versus generation (y axis) versus allele-variance per locus (z axis). This at least attempts to relate information about the actual chromosomes structure to the fitness of the population, but only gives *relative* information about allele frequencies.

In a similar manner, Routen and Collins [30] use graphs in which the similarity of each chromosome in a generation is compared to a base chromosome and then a 2-D scatter plot is produced of fitness versus similarity. The drawback with this method comes in choosing a suitable base chromosome.

Allele-wave visualization is described by Bedau *et al.* [2]. In this method, "activity wave diagrams" highlight the quality of the main adaptive events from which phenomena such as competitive exclusion, cooperation, and frozen accidents can be observed. The "activity" of each possible allele is set initially to zero, but each time it is used, an associated counter is incremented (in this paper, an allele is a rule that may or may not be used by an agent in an artificial market). The counters are inherited along with the allele during reproduction, thus maintaining an increasing activity count over an entire lineage. The diagrams are constructed by plotting allele activity versus time. For this method to be successful, the notion of "use" must be well defined. For example, in iterated prisoners dilemma studies, GAs are sometimes used to evolve strategies and a chromosome can represent a lookup table of actions. It is then interesting to discover how many entries are actually consulted; Darwen and Yao [12] show that it may be only a few.

B. Visualizing the Space Searched by GA

The problem of visualizing multidimensional data has long been recognized and has received much attention in the fields of computer science, neural networks, and datamining. Some of the techniques from these fields have also been adopted for use in GA visualization. An overview of multidimensional visualization techniques that are amenable to GA visualization is given by Spears [33], including glyphs, projections techniques, and parallel coordinates. Further examples are principal component analysis (PCA) and biplots, which can be used to produce scatter plots, where the multidimensional data is projected to two or three dimensions. Note, however, that each of these methods can be applied only to a single distribution at a time so that if each chromosome in a population is treated as a vector and the population at one generation is, thus projected, onto two or three dimensions, then there may be no meaningful relationship between such plots at different generations. However, Collins [6] describes a methodology and associated tool based on PCA in which the eigenvectors computed from each population can be projected into the corresponding eigenspace. This is then plotted using a 3-D representation, producing a manifold that is a compact representation of the data. The manifolds can be conceptualized as a global signature of the dynamics of the underlying process.

A possible improvement is to use Sammon mapping [31], which is another technique for producing a low-dimensional analogue of a high-dimensional data set. For example, each datum is represented by a 2-D point placed initially at random in the plane. These points are then moved relative to each other in order to approximate the distance relationships between the actual data; if, for instance, two points are close in 2-D space, but far apart in the original space, then they are nudged apart in the 2-D space. This has the advantage that the entire search space can be represented and, hence, spatial relationships across generations can be captured to some extent. However, the complexity of constructing the map is quadratic with respect to the number of points and, hence, it soon becomes intractable. An example of this applied to GA visualization (of the entire search space) is given in Dybowski *et al.* [15].

Collins [8] proposes a new method that has only linear complexity known as "search space matrices." The idea is to map all possible chromosomes onto a 2-D matrix such that the Hamming distance between any two neighboring points is minimal. Chromosomes are displayed as a circle of fitness-dependent radius; if the chromosome is not present in the population, the matrix entry is blank instead. This technique can be used to produce a linear mapping for an entire search space, which maintains spatial relationships across generations. It is applicable only to categorical alleles rather than real-valued ones and only to problems in which the number of possible chromosomes is small. It has the advantage however that it can highlight schema information. Any genetic information regarding schemas or genes is completely obscured in Sammon mappings and PCA/biplots.

Another method of visualizing the search space is proposed by Shine and Eick [32], who break the analysis down into three areas. They describe a method of generating "coverage maps," which provide a means of estimating the distribution of the explored solutions relative to the entire search space. In such a map, if an individual appears in more than one generation, it is always depicted at the same position. Such maps provide a means of visualizing if the GA has become stuck in some particular small region of the search space for example. Shine and Eick suggested the use of quad codes to generate such maps, which were developed originally as a means of creating resolution-independent pixel images [25].

Coverage maps do not provide information about the relative distance between individuals and they hide genetic information about the population. To rectify this, [32] suggested the use of distance maps in which the members of a population can be depicted in 2-D space in such a way that the n-dimensional distances between the actual chromosomes are approximated by the 2-D distances with minimal error, as in Sammon mapping. The distances and, hence, coordinates can be computed either using simplex-like algorithms or even a GA. This method again suffers from the same problem associated with PCA-like scatter plots in that spatial relationships are not preserved from generation to generation.

Fitness information can be incorporated into both coverage and distance maps, either as an extra dimension on the 2-D map or by employing the use of color. Shine *et al.* [32] provide an example of incorporating color usage via contour maps. Generating a contour map is extremely computationally complex, however, especially with nonuniformally gridded data as is the case with a GA. Although algorithms such as Delaunay triangulation [29] exist to perform such transformations, the computational overhead is very high.

In summary, although all these methods can provide valuable information about how a GA traverses a search space, all are dogged to some extent by computational complexity and the enormous size of the space of possible chromosomes. Furthermore, with the exception of search-space matrices and perhaps distance maps, genetic information about individual chromosomes is always obscured.

C. *Visualizing The Problem Search Space*

Several authors have proposed methods of visualizing the entire search space defined by the problem rather than merely the part visited in a GA run in order to understand if a GA is likely to be successful in such a space and, if so, to aid in designing suitable operators.

Cartwright and Mott [5] suggest a method of visualizing the search space with the aim of trying to make predictions about population sizes and good crossover rates for flow-shop scheduling problems. A fitness landscape is generated by altering a chromosome such that neighboring points in the landscape differ by exactly one gene position—for example, a 3-D graph can be generated by initializing a chromosome at random and then altering it n times in each of two directions to give the horizontal coordinates and plotting the fitness of each point on the vertical coordinate. Mattfeld [26] points out, however, that tuning a GA based on a visual impression of such a landscape is not very satisfactory and, moreover, many landscapes generated in this way look similar. The ease with which the landscape can be generated is reliant on the problem representation and may be difficult in some cases.

Mattfeld goes on to suggest the use of "configuration space analyses" first suggested by Kirkpatrick and Toulouse [24]. This type of analysis relies on generating two pools of solutions—one at random and one containing locally optimal solutions—and compares the mean normalized Hamming distance of solutions to all other solutions in their pool. A plot of distance versus fitness is generated from which an estimate of the difficulty of the problem for a GA can be made. This approach is time-consuming due to the need to generate locally optimal solutions by some method.

III. GAVEL

GAVEL[1] is an offline tool that allows easy visualization of all the important features of the evolution, giving the GA user the ability to analyze the performance of each of the operators both visually and graphically and the means to perceive the dynamics of the evolutionary process.

The central idea is to start from the "end" of a GA run, from the best solution found, and *rewind* the evolution rather than taking the traditional approach of watching a solution being created. The parents of the best solution are found and then their parents, etc., all the way back to the initial population so as to produce the complete ancestry tree for the best solution. This ancestry tree forms the central feature of the display (see Fig. 1 for a schematic example[2]) in which each chromosome in the tree is displayed as a small colored rectangle. Moreover, as well as

[1]GAVEL was written in JAVA and is available from http://www.dcs.napier.ac.uk/~emmah/software.htm

[2]This diagram provides an illustration of a real screen shot; however, the details are obscured as the diagram is not reproduced in color.

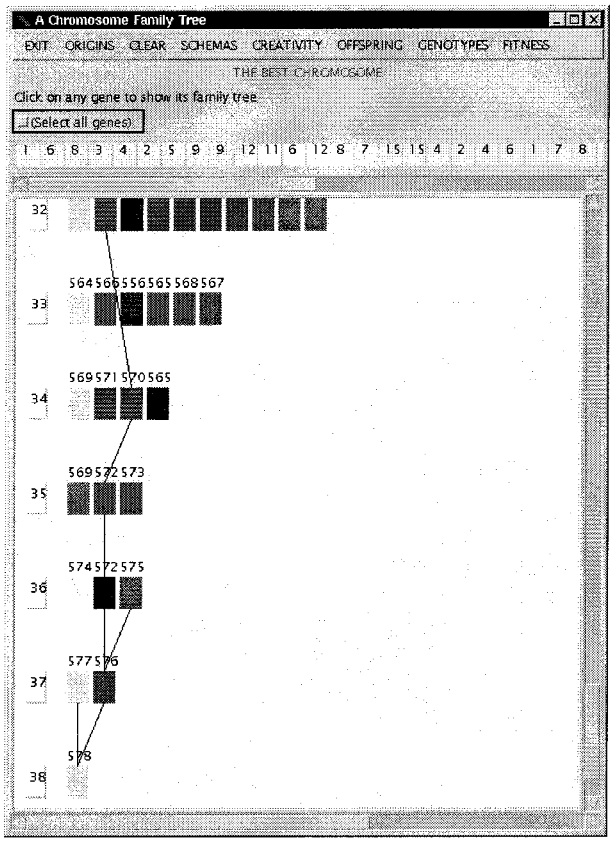


Fig. 1. Example screen shot from GAVEL. Main screen shows the best solution found by the GA and the ancestry tree for the entire GA run. Horizontal rows indicate generations, with the leftmost icon in each row indicating the generation number. Identification of each chromosome is given above each icon. Solid lines trace the parentage of selected chromosomes.

tracing the evolution of *individuals*, GAVEL also tracks the history of every individual *gene* contained in the final solution back through the population to the individual it originated in. The tree gives an immediate picture of the dynamics of the whole of the evolution of the best solution.

Although collecting and storing information about every gene in every chromosome encountered by a GA obviously results in enormous data files, GAVEL only needs to use the information that is relevant to the formation of the best solution. Using the best solution found as a starting point, the history of every gene in this solution is traced back through the population to the generation in which it first appeared using the information stored in the output file generated by the GA. All chromosomes that do not play any part in propagating these genes to the final solution are then disregarded, resulting in a data file that is considerably reduced in size. (The information output by the GA used to trace the history of each gene is described later in more detail in Section IV-A.) For example, running the benchmark problem $bf6$ [1] with a population size of 50 and chromosomes of length 32, using a generational GA typically results in an optimum solution begin found around generation 50—storing the required information about every single gene and chromosome produced results in a data file of approximately 3.3 MB. Extracting the information required to produced the ancestry tree, however, results in a data file of only 200 kB. The relevant data is extracted automatically from the output of a run by a program provided with GAVEL. Although this calculation is outlined for a modest-sized GA run, the numbers can obviously be scaled to give an impression of the size of file produced for runs using more generations and/or larger populations.

With the ancestry tree as the basic display, color is used to express various kinds of information about chromosomes in the tree. This is described in detail in Section III-A. Using the stored information, GAVEL can also derive a series of graphs from the tree, which can be plotted using an external plotting program. These graphs give useful information about the productivity of the genetic operators and the dynamics of the evolution.

GAVEL was deliberately designed to function as an *offline* tool in order to provide a mechanism for visualizing the entire GA run as a single diagram that could then be examined in more depth as the user required. As we wished to eliminate *all* superfluous data produced during the GA run in order to focus on only relevant operations, it was essential to rewind the evolution from the best solution found; this prevents use of the tool in an online capacity. Futhermore, several tools already exist for interactively examining the effect of parameter changes as a GA progresses online; the aim of GAVEL is to complement these tools and not to reimplement such features. Also, as already discussed in Section II, there are legitimate objections that can be raised against such interactive methods.

A. Use of Color

An inherent problem with GA visualization lies in the multidimensionality of the data. There is an upper limit to the amount of information that can be understood easily on a screen and, hence, methods of visually displaying large populations over many generations are limited. Bertin [3] recommends that no more that 10 000 elements should be shown in any data matrix and DuToit *et al.* [14] point out that the information in such matrices is difficult to absorb—it is complex to grasp the multiple differences between rows or columns, let alone understand the interactions between them by simply examining a data matrix.

Color and/or shading helps—it is much easier for the eye to grasp color patterns and relationships than it is to understand text. Hence, some data matrix problems can be reduced. Despite this, all visual interfaces are still limited by the fact that only 256 different colors can be displayed on screen. As GAVEL makes use of color shading to indicate gene values (alleles), fitness, and gene origins (i.e., the generation in which a gene originated), these limitations are now discussed in more detail.

First consider the question of displaying alleles in which there is a clear ordering between possible values, fitness values, and gene origins as each of these categories potentially can cover an infinite range of possible values. GAVEL adopts a *shading* scheme approach. Although there are natural numeric orderings of RGB colors, there is no visually distinctive relationship between all consecutive colors and, hence, simply dividing the full RGB range into color intervals is not particularly helpful. However, by fixing the values of any two of the RGB values and varying the third between zero and 255, a graded color shading can be achieved spread between two colors. The usefulness of this approach decreases as numeric range of values that needs to be represented increases, eventually approaching the limit of how many shades are visually distinguishable by the user's eye, and is in any case limited to 256 different shades. Therefore, value ranges including more than 256 values cannot be mapped to individual shades. However, we suggest that this is, in fact, irrelevant because the eye cannot distinguish meaningfully so many shades anyway—what is more important in regards to understanding GA performance is that the user gains some visual impression of the distribution of values across chromosomes and generations. A shading approach satisfies this requirement as the user can gain a visual impression of color patterns apparent in the shading of the entire diagram. Furthermore, it is also worth remembering that of the order of 10% of all users will have some degree of color blindness.

In order to visualize categorical alleles in which there is no numerical ordering between values, GAVEL assigns a unique color to each allele. Again, the caveat applies that the number of possible values should not exceed the number of colors easily distinguishable by the eye for the display to be meaningful. The concept could in theory also be extended to real-valued alleles by partitioning the allowed allele range into intervals of fixed size and assigning a unique to color to each interval range, although the usefulness of this approach may depend on the range and allowed precision of the alleles in a given problem. This approach could also be used to partition fitness values or gene origins for problems with a wide range of possible fitness values or run for many generations. Although this is currently not supported in GAVEL, it is planned for future versions.

IV. Description of the GAVEL Visual Interface

This section describes the facilities included in GAVEL in depth. As the images are not reproduced in color, we do not attempt to provide screenshots from GAVEL, but instead present schematic diagrams in order to illustrate the relevant features more clearly. We first provide an overview of the visual interface and its operation in Section IV-A. A key feature of the design of GAVEL is in its use of data *views*. Using the ancestry tree as the basic display, GAVEL makes extensive use of color highlighting to flag properties of the chromosome rectangles. This allows the data to be visualized from a number of different perspectives. Each view is described in detail in Sections IV-B onwards.

A. Operating the GAVEL Interface

This section provides an overview of the management of the GAVEL visual interface and its mode of operation. As stated in the previous section, it is necessary to collect and store information about every gene in every chromosome generated in the entirety of a GA run in order to generate the ancestry trees used by GAVEL. It is necessary for the GA practitioner to modify his/her own program code in order to output a textfile of information during the running of the GA. For each chromosome in each generation, information is output regarding the unique identifier of the chromosome and the generation it was encountered. For every gene in each chromosome, data must also be output giving the value of each gene in the chromosome, the identifier of the parent chromosome from which the gene was inherited, and the gene origin, i.e., the generation in which the gene was created. Genes created in the initial population have the gene origin set to zero, for all genes thereafter created via

```
Generation:    15
Chromosome ID: 101
Alleles:       1  12  3  5  1  19
Parent Id      90 90  80 80 80 80
Gene Origin I  0  0   0  0  45 0
```

Fig. 2. Extract from the output text generated from a GA that is used in the GAVEL preprocessing steps to format the ancestry tree file read by the GAVEL software.

the mutation operator, the gene origin is set to the generation at which the mutation occurred. A sample from the output for a single chromosome is given in Fig. 2 for an example chromosome of length six genes.

When this information has been obtained, a program provided with GAVEL extracts the relevant information from this large data file, generates the ancestry tree, and writes it to a new file in a format readable by GAVEL. The original large data file is then deleted. When GAVEL is run, the main window displays the ancestry tree and the details of the best chromosome found in the entire run. The only highlighting on the ancestry tree identifies the best chromosome in each generation and those chromosomes that were copied into a new generation via operation of an "elite" strategy, which merely copies the best n members of the current generation into the next generation. The user can then impose a selection of views on the tree that are described in detail in Section IV-B by using the dropdown menus available from the main menu bar. More detailed views pertaining to individual generations appear in new windows and are actived by clicking on individual chromosome icons or generation identifier icons.

B. Ancestry View

The complete ancestry of any individual *chromosome* can be traced interactively on the tree by clicking on that individual. This causes lines to be drawn from each individual to its parents. This view can be "reversed" by requesting that the *offspring* of any given chromosome be highlighted. The lineage of any individual *gene* can also be traced on the tree in order to track the history of a gene from a selected chromosome back to the chromosome where the gene first appeared in in the initial generation. Usage of the ancestry view is illustrated in Fig. 3. In this diagram, the winning chromosome appeared in generation 13; the diagram thus shows the preceding six generations of ancestry of this chromosome. A gene in the winning solution (labeled $*$ and also shaded) has been selected and the ancestry of this gene is traced in dotted lines; the gene originated in generation nine (thus, via a mutation) and, hence, the generation number is emphasised in bold. The solid lines with arrows indicate the parents of the chromosome that the lines originate from. A chromosome in generation eight has also been selected (labeled P) and the offspring of this chromosome are displayed by hatched shading.

C. Fitness View

This view enables the user to gain a relative impression of the fitness distribution of chromosomes in the ancestry tree. Each individual icon is color shaded according to its relative fitness

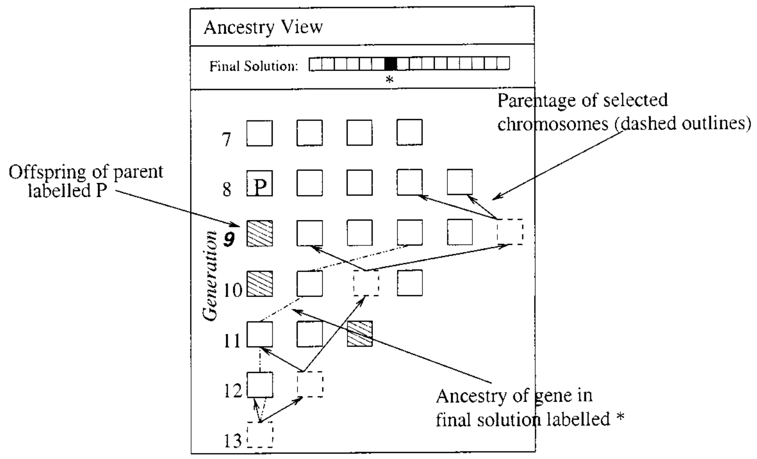

Ancestry View
Final Solution:
*
Parentage of selected chromosomes (dashed outlines)
Offspring of parent labelled P
P
Generation
7
8
9
10
11
12
13
Ancestry of gene in final solution labelled *

Fig. 3. Schematic representation of the ancestry view of the tree. Best solution found is shown at the top of the display. Horizontal rows indicate generations, with each box in each row indicating an individual chromosome. Clicking on individual genes (dotted lines) or chromosomes (solid lines) traces their parentage back through the tree.

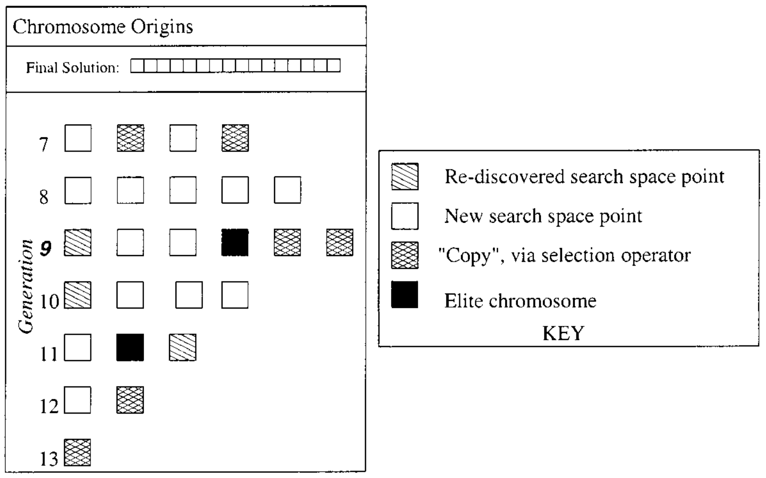


Fig. 4. Schematic representation of the *origin* view of the tree. Best solution found is shown at the top of the display. Horizontal rows indicate generations, with each chromosome icon colored according to its *origin*, i.e., **Rediscovered**—the chromsome is newly created via recombination and/or mutation operators, but has previously been discovered in an earlier generation; **New**—the chromosome was created as a result of applying recombination and/or mutation and is new; **Copy**—the chromosome was copied from the previous generation via the selection operator; **Elite**—the chromosome was copied from the previous generation as due to the elite mechanism.

value. A generation is displayed as a horizontal row of icons and the icons in each row are arranged in order of decreasing fitness. The user can also highlight all chromosomes whose fitness satisfies a user-supplied constraint via a dialog window. The tree appears as in Fig. 4 with each chromosome icon shaded in greyscales, indicating its relative fitness.

For problems in which multiple genotypic representations can lead to the same fitness value (e.g., indirect representations of scheduling problems), another feature is the ability to select all chromosomes having some particular fitness x and then color them by assigning each unique *genotype* a different color. Thus, a user can identify not only the number of different genotypes that represent the same phenotype, but can also look at which of these genotypes are most common and monitor the spread of building blocks resulting from each genotype.

D. Origin View

The icons in the ancestry tree can be colored according to their *origin*, i.e., as to whether or not the individual was created in the generation it appears in by recombination or mutation, whether or not it was copied from a previous generation via the selection mechanism, or whether or not it was created in the current generation, but does not represent a newly discovered point in the search space. Chromosomes that are copied from a previous generation are further distinguished as to whether they were *selected* by the selection mechanism or automatically copied due to the presence of an *elitist* selection strategy. This helps the user to understand the extent to which the ancestry tree is dependent on the genetic operators exploring the search space and also to gain some impression of how much of the search space is actually explored. This view is illustrated in Fig. 4.

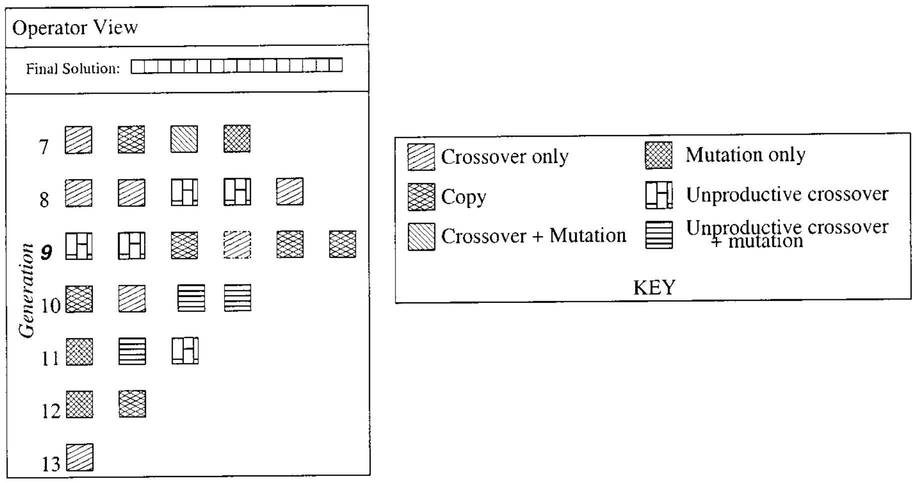


Fig. 5. Schematic representation of the operator view. This view of the tree indicates the role each genetic operator played in creating the chromosomes that make up the ancestry tree. For each chromosome, the key indicates which operators were responsible for the formation of the chromosome.

E. Operator View

In order to determine the productivity of the genetic operators, a color highlighting scheme can be applied that indicates the genetic provenance of each chromosome by categorising it into one of the following groups, according to how it was produced:

1) no genetic operators involved—chromosome was copied via selection;
2) crossover only;
3) crossover followed by mutation;
4) mutation only;
5) "unproductive crossover" (see below for explanation);
6) "unproductive crossover" followed by mutation.

Categories 5 and 6 refer to "unproductive crossover." The term "crossover productivity" was introduced by De Jong and Spears [13] and refers to the ability of the crossover operator to produce offspring that differ from their parents in some way, rather than just being cloned. Thus, an unproductive crossover operation is one in which the resulting child chromosome is in fact identical to one or both of its parent chromosomes. As the chromosomes in each generation of the ancestry tree are displayed in order of decreasing fitness, then superimposing the operator view also gives an impression of how the relative fitness of chromosomes correlates with the operators that produced them. This view can be used in conjunction with the graphs described in Section V to determine the usefulness of each operator and perhaps alter the operator settings based on this analysis.

An example of the operator view is given in Fig. 5.

F. Schema View

The ancestry tree can be used to trace the formation and spread of building blocks through the generations by selecting a schema via a simple "point-and-click" interface. All individuals containing the schema are then highlighted on the ancestry tree. Again, the fact that each generation is displayed in order of decreasing chromosome fitness gives an impression of the relative fitness of chromosomes containing a selected schema. The view is also useful in determining how and when selected schemas were formed. This can help the user to study directly the important but often neglected question as to whether or not the solution appears to be composed to discernible building blocks. This is regularly claimed to be a distinguishing feature of GAs, but needs much more investigation.

G. Population and Individual Views

GAVEL also allows any population of chromosomes in a single generation (or a single individual) to be viewed as a color matrix (as explained in Section III-A).

The "view" of the matrix can be changed, either to indicate *allele values*, *gene origins*, or *operator origins*. Viewing allele values gives an impression of the allele distribution in each generation. If gene origins is selected, then alleles are shaded according to the generation in which they first appeared, giving a view of how mutation affects the overall performance of the GA. The darker the shading, the more recent the mutation. If operator origins is selected, then the coloring shows either which parent the gene was passed from if it appeared as a result of crossover or else the fact that the value had been created by a mutation. An illustration of these views is given in Fig. 6 for a single generation. Each row of each matrix represents a single chromosome. In the operator-origins diagram, the two parents are represented using black and white with hatching to indicate mutations. In the gene-origins diagram, all unshaded genes originate in generation zero.

V. Plotting Graphs From Data Generated by GAVEL

In the future, a graph-plotting tool will be linked directly with GAVEL, but at the moment, the software simply formats data to plot a number of different graphs, which are then output to files. The files can be read directly by an external plotting tool such as Excel. Eight different graphs are currently generated, which are described in detail in the remainder of this section. In order to illustrate the potential utility of each graph, graphs have been generated for a timetabling problem, which is described briefly below.

This is a real timetabling problem faced by the Informatics Division in Edinburgh each year. The task is to schedule M.Sc.

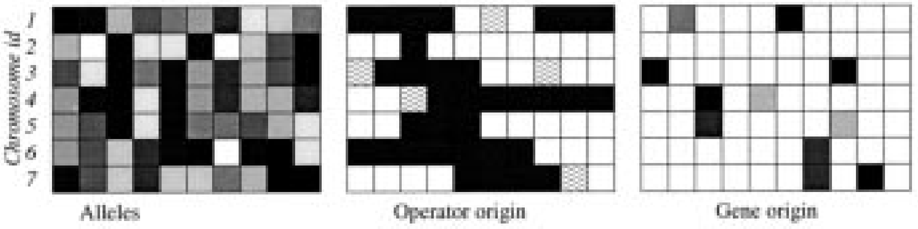


Fig. 6. Schematic representation of the population views. Population views can be generated for each generation in the tree and allow the chromosomes making up a generation to be viewed from three perspectives: the gene values, the origin of the genes (i.e., the parent each gene was inherited from and any mutations that have occurred), and the gene origins, i.e., the age of each gene.

student exams over a short period of time such that no student has clashing exams and also that each student has at least one free slot between each exam taken. In the problem shown, 44 exams had to be scheduled in 36 slots over a period of nine days. The exams had to be spread such that no student had to take two exams too close together. A straightforward direct representation is used in which each gene represents an exam with the value of the gene representing the slot in which the exam takes place. Two-point crossover is applied with probability 0.6. A targeted mutation operator is used in which (with probability 1.0) a problematical exam is selected by tournament selection of size five and has its slot changed to a better slot, chosen again by tournament selection of size five. The population size is 50 and rank selection is used with the fitness of a timetable being determined by a penalty function. The problem is solved perfectly by this GA.

The following sections all refer to the graphs shown in Fig. 7.

A. Fitness Versus Time

Graphs can be plotted showing the best, worst, and average fitness of the chromosomes in each generation of the ancestry tree. Useful information can be determined by comparing this graph to the corresponding plot for the entire GA. For example, in some problems we have determined that the average fitness of chromosomes in the ancestry tree is generally superior to that of the whole population for any given generation. Conversely, it is also possible to observe that the final solution does not necessarily inherit genes from the fittest chromosomes in each generation. Thus, this graphical information may be used to determine if a problem contains some deceptive features.

In Fig. 7(a) note that the average fitness of the ancestry tree tracks that of the whole population until around generation 42 when the ancestry tree starts to become fitter relative to the entire population. The best fitness of the whole population and the ancestry tree remain the same throughout the evolution, indicating that the problem has no deceptive features.

B. Operator Utility

A new term "operator utility" is introduced which refers to the ability of an operator to produce offspring that occur in the ancestry tree. (This avoids confusion with the already established term *crossover productivity* [13]). Thus, the utility U of an operator o at generation g is defined as the fraction of chromosomes in the ancestry tree at generation g produced as a result of applying operator o to the previous generation. Thus, plotting operator utility against generation [see Fig. 7(b)] allows comparison of the actual utility of an operator to the rate at which the operator is applied and, hence, gives a very useful indication of the importance of each operator. This information could then be used to adjust operator settings.

Fig. 7(b) shows that the mutation operator acting alone produces a small fraction of the ancestors of the final solution, i.e., in the absence of any crossover. In the later generations, the crossover operator is often unproductive—this is effectively equivalent to simply applying mutation. A significant fraction of each generation consists of copies of chromosomes from the previous generations.

C. Operator Fitness

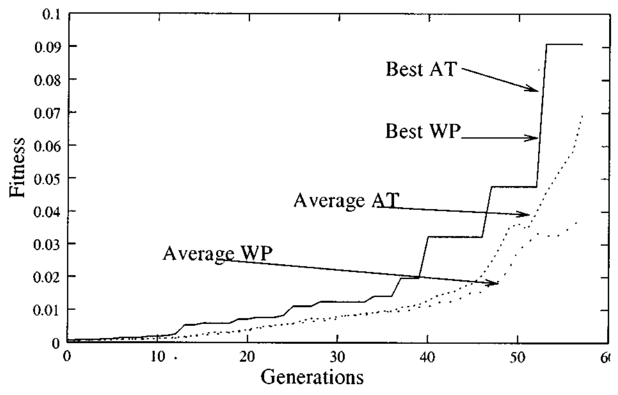
The *operator fitness* of an operator o is defined as the average fitness of the chromosomes in a generation g of the ancestry tree that were produced as a result of applying operator o to the preceding generation. GAVEL generates a graph that shows the fraction of the total fitness of chromosomes in a given generation attributable to each operator [see Fig. 7(c)]. Examining this graph provides clues as to the mechanisms by which fit areas of the search space were reached. This information can potentially be used to guide the choice of operator settings.

D. Number of Chromosomes in the Ancestry Tree per Generation

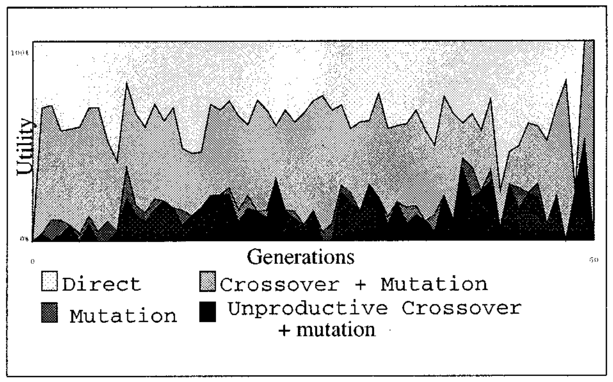
Fig. 7(d) represents the number of chromosomes in each generation of the tree as a bar graph. This gives the user some idea of the dynamics of the evolution process. For example, Fig. 7(d) shows that between generations 17 and 40, a large number of chromosomes contribute to the ancestry tree. Hence, the GA is combining information from many different chromosomes, implying that population-based search is advantageous.

E. Origin of Genes in Final Solution

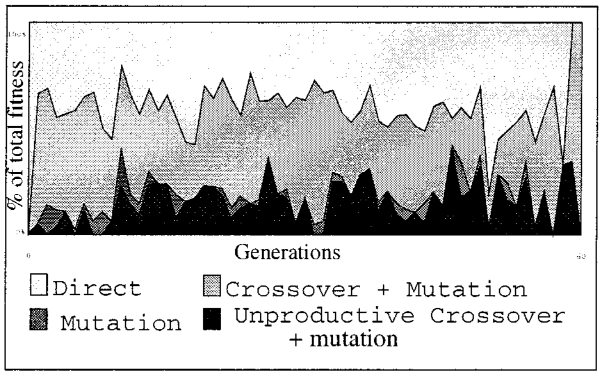
Two data files are produced that provide information regarding the origin of genes in the final solution. Fig. 7(e) depicts the number of chromosomes in each generation that contribute genes to the final solution. This information can be combined with that shown in Fig. 7(f), which shows how many of the genes in the final solution originated in each generation. Interpreting these two graphs provides further valuable information about the utility of crossover and mutation. For example, we see from Fig. 7(e) that just under half of the genes originated in the original generation, but from Fig. 7(f), all of these genes were present in one single chromosome. Thus, the role of crossover has not been primarily in bringing new building



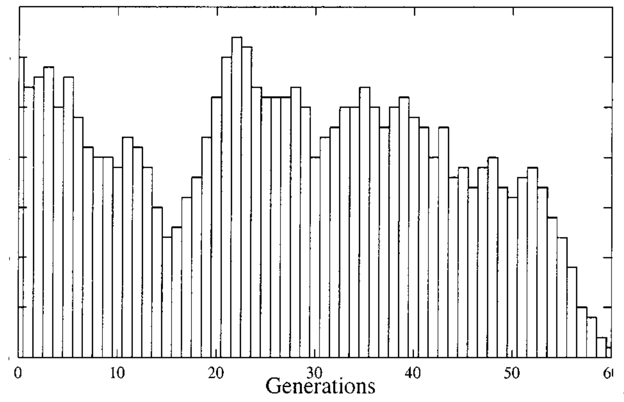
(a)



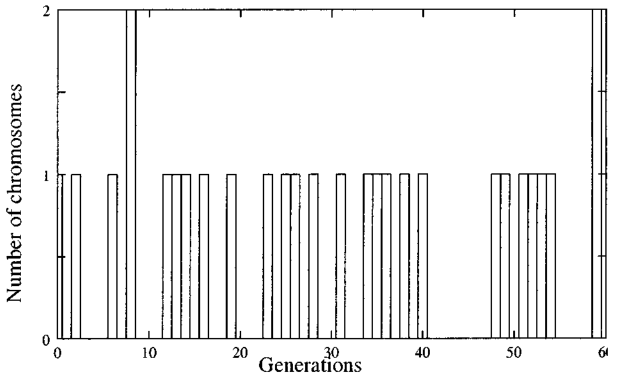
(b)



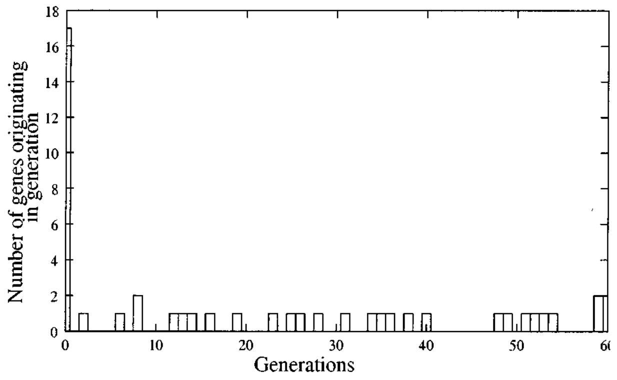
(c)



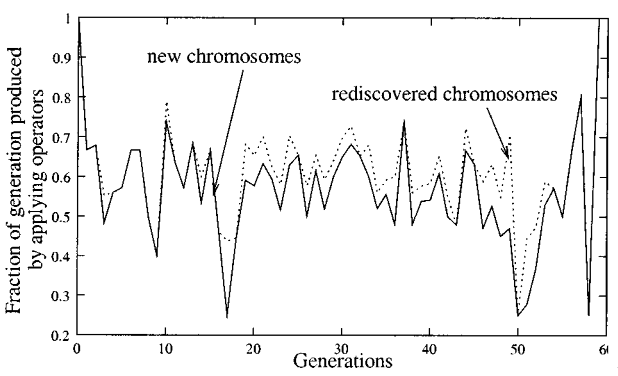
(d)



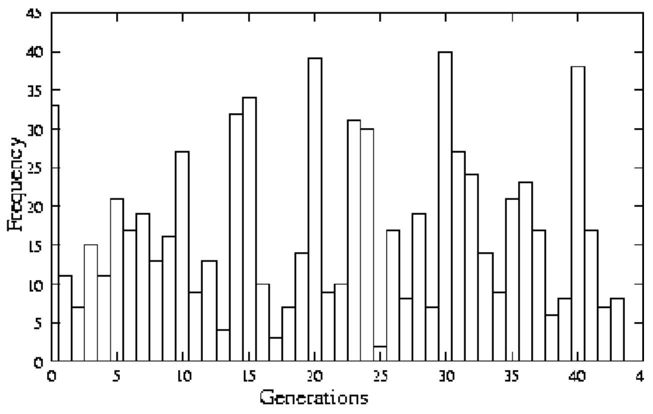
(e)



(f)



(g)



(h)

Fig. 7. Gavel generated graphs (plotted using gnuplot). In each case, the data required to format the graph is output directly from GAVEL. (a) Fitness versus time comparison for whole population (WP) versus ancestry tree (AT). (b) Operator utility. (c) Operator fitness. (d) Number of chromosomes in ancestry tree. (e) Number of chromosomes contributing genes to final solution. (f) Origin of genes in final solution. (g) Exploration of search space. (h) Frequency of targetted mutation per locus.

blocks together but in maintaining good building blocks , which are then reproduced via the selection mechanism. The formation of the final solution seems to have relied heavily on mutation to find the optimal values for other genes throughout the entire evolution.

F. Exploration of the Search Space

Fig. 7(g) shows the fraction of the total number of chromosomes in each generation that represent new points in the search space "discovered" by applying the reproductive operators to the previous generation. The graph indicates if those points have been found previously in other generations or if they represent previously untested points. The fraction of new chromosomes discovered can be compared to the rate at which the operators were applied. In this example, crossover was applied with probability 0.6 followed by mutation with probability 1.0—in the majority of generations, the fraction of chromosomes in the ancestry tree exceeds this value, implying that the crossover operator has been useful, however, in several generations the fraction of new chromosomes produced is significantly less than 0.6. Furthermore, we see that during the middle section of the evolution, i.e., from generations 20 to 50, the operators tend to produce chromosomes that have been discovered in preceding generations. One explanation of this is that perhaps due to lack of diversity in the populations in these generations, the operators are not particularly effective in exploring the search space over this region. An alternative explanation, however, is that at this point in the evolution, the search has homed in to the correct region of the search space and the GA is being somewhat inefficient in repeatedly generating points that have already been discovered when a few mutations to these chromosomes would produce the final solution. Therefore, reduced surrogate crossver [4] might be an option worth considering in this problem since it restricts the choice of crossover points to places where the parents disagree.

G. Mutation Frequency

The final graph Fig. 7(h) shows the frequency at which mutation occurs at each locus. This is measured by noting the locations at which mutation has occurred each time a *new* chromosome appears in the ancestry tree. This can provide problem-specific information; for example, in the timetabling case, loci with high rates of mutation indicate particular exams that are causing problems. This type of information is of value to users of the system who may wish to modify the data in some manner.

VI. Other Example Applications of GAVEL

In this section, we briefly describe other example problem areas in which GAVEL has been used by the authors to enhance understanding of particular GA problems. In the first example, GAVEL was used to provide a principled method of setting the mutation rate in a job-shop scheduling problem (JSSP). In the second example, GAVEL was used to explain the observed performance of a GA on a long-path problem.

A. Refining the Mutation Rate in JSSP

Ross [16] showed that promising results with JSSP problems could be obtained using a simple indirect representation that guaranteed that feasible chromosomes were produced using a uniform crossover operator. These experiments are described in detail in [16]. However, to summarize, in the original papers, experiments were peformed on a number of problems ranging in length from 30 to 400 operations in which the population size used was 50, uniform crossover was applied with $p = 0.8$, a simple mutation operator was used that with probability $p_m = 0.5$ performed a swap between two randomly chosen alleles, and the results averaged over 50 experiments. We examined many individual runs of this GA (each using a different seed) for one of these problems that contained 45 operations to be scheduled. When the *utility* of the mutation operator was examined from the graphs generated by GAVEL for each of these individual runs, it appeared that when this mutation operator was applied in conjunction with the uniform crossover operator, then the average utility of the mutation operator was significantly below the chosen probability $p_m = 0.5$. When the *origins of the gene in the final solutions* were observed, it appeared that a significant proportion of them (generally at least 35%) did originate via a mutation. Thus, it appeared that although mutation was not particularly useful in producing chromosomes that contributed to the ancestry tree; on the occasions when it was used, the mutations appeared important.

Furthermore, when the *ancestry view* was used to trace the ancestors of all the genes in the final solutions of several experiments, it was clear that many of the lines connecting the chromosomes containing these final genes connected chromosomes that originated via a mutation operation. Also, these chromosomes were seldom highly fit compared to other members of their generation.

The result of the analysis suggested, therefore, that better results could be obtained by increasing the mutation rate beyond its current setting. Further experiments (which were repeated 50 times and the results averaged) confirmed this hypothesis. Although the same result could have been determined through a time consuming trial-and-error approach, GAVEL provided an easy way in which to do this, based on an understanding of the effects of the operator.

B. Long-Path Problem

This section shows how GAVEL can be used to open the GA "black box" and provide some explanation of how and why a GA arrives at a solution for a particular problem.

Consider the well-known long-path problem. This problem was first introduced by Horn and Goldberg in [21]. The original idea was to show that a problem that consists of a single hill may still be solved faster by a GA than by a hillclimber; the hillclimber, which merely tests what happens when each bit is flipped in turn, can always find the one and only bitflip that represents the next step up the path. Despite this, the hillclimber is still faced with a very long climb even though there is never a doubt about which direction is uphill.

A long path is constructed recursively in a universe of binary chromosomes of width N in such a way that it never comes within a Hamming distance 1 of itself. Thus, a search of a neighborhood within Hamming distance 1 of a given chromosome will find a unique uphill step on the carefully designed path. The path begins at "000...00"; hence, a search algorithm must first find the start of this foothill before it can begin proceed along the path. The path is constructed recursively from this point. Clearly not all of the possible 2^N chromosomes lie on the path—any chromosome not on the path is given a fitness equivalent to the number of zeros it contains, which directs the search toward the start of the path, the zeros-max foothill. A detailed description of the original problem can be found in [21]. Although a GA may solve the problem faster than a simple hillclimber, Höhn and Reeves [20] show that a GA still takes time of the order L^2 (where L is the length of the string) to find the optimum and Kallel and Naudts [23] provide results on a more general version of the problem that suggest that the expected number of generations to find the optimum solutions is exponential in the string length.

While investigating the effects of parameter values on this problem, some interesting properties emerged not reported previously. A series of experiments was performed initially using a generational GA on an instance of the long-path problem of length 29 (and, hence, maximum fitness 49 178), recording the average fitness of the best solution found by a GA over 50 runs, and also the average number of evaluations required to find the best solution. The experiments used two-point crossover and each was run for a maximum of 1000 generations or stopped if the population had converged. The mutation operator simply flipped each bit in the genome with some assigned probability m, varied from 0 to 0.1 in steps of 0.01, for a variety of population sizes in the range 50–250. The results of these experiments are summarized in Figs. 8 and 9. Taking the results at face value, it would seem that the GA performs well on this problem, i.e., it finds the optimum solution in the majority of cases over a wide range of parameter values. Bear in mind that a simple hillclimber needs to take around 49 000 uphill steps and at each step needs to consider about $N/2$-bit flips to discover which step is uphill, requiring of the order of 700 000 evaluations.

However, when the results of the experiments were examined using GAVEL, it was clear that the results were to some extent an artefact of the problem design rather than being attributable directly to the searching power of the GA. Several of GAVEL's facilities were useful in the analysis—in the following discussion, the analysis of a single experiment is described that was found to be typical of the majority of those performed. First, GAVEL's *fitness view* was used to highlight all chromosomes in each generation that were known to actually lie on the long path and not on the zeros-max foothill that leads to the start of the path. This showed that for typically 50%–75% of the generations, no chromosomes *in the ancestry tree* actually lay on the long path. This view also showed that furthermore, only one chromosome that actually lay on the path was *ever* discovered and that the fitness of this chromosome indicated that it was actually only one step from the summit. Copies of this fit chromosome spread gradually through the ancestry tree until the summit was reached suddenly.

Examining the *operator view* in conjunction with the individual chromosomes showed that the only chromosome that lay

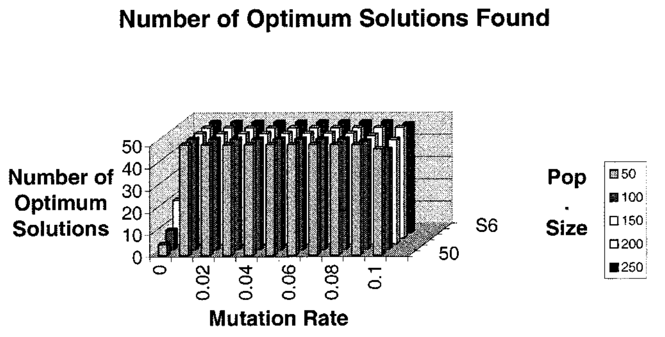


Fig. 8. Results from long-path experiments showing the number of optimum solutions (from 50 experiments) found as the population size and mutation rate is varied.

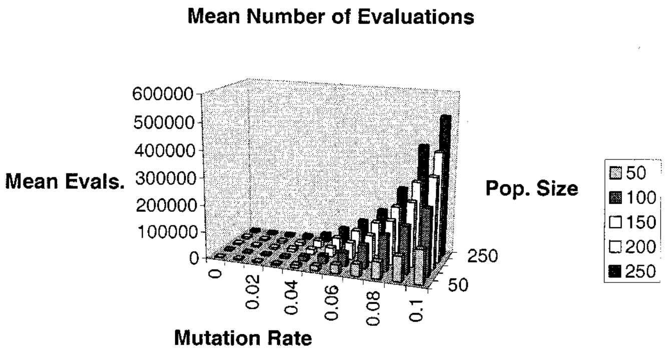


Fig. 9. Results from the long-path experiments showing the mean number of evaluations per experiment for varying combinations of population size and mutation rate. Each experiment was stopped after a maximum of (1000 × population size) evaluations, or when the optimum solution was found, whichever was sooner.

on the path was discovered via a fortuitous crossover and that at a later stage, a single mutation to this chromosome located the summit of the path.

The preceding analysis showed clearly that the GA made very little, if any, progress along the actual path, despite discovering the optimum solution. Further clues were provided by viewing the genetic makeup of the chromosomes in each generation. In early generations, the chromosome populations contained sufficient ones scattered throughout each population that fortuitous crossover and/or mutation operations could "teleport" the search to the top of the path from a state in which none of the chromosomes actually lie on the path. The fact that this happens is attributable to the design of the problem. The end of the path lies very close to the start of the path, only two bits different, and traversing the initial zeros-max foothill forces most of the ones out of the population. By the time the GA is about to tackle the long path itself, most of the population is already very close, in Hamming terms, to the maximally fit answer. This means that there is a high probability of crossover and mutation managing to stumble across the optimum solution without having to actually climb the path at all. Using mutation increases the probability of finding the optimum solution. Thus, the features contained in GAVEL led to a significant insight into the *reasons* why a generational GA performed well on this problem. These insights were not apparent from simply analyzing the final results of the experiments.

VII. Comparison to Other Tools

This section provides a brief overview of some of the other tools that are available for visualizing GAs. Several tools provide some similar features and other provide a set of complementary techniques that could be used in conjunction with GAVEL to provide a complete analysis toolkit. We divide this section into "online" tools, which support real-time analysis of the GA as it runs, and offline tools, which are used in a similar manner to GAVEL.

A. Online Tools

Several tools concentrate on facilitating visualization of the *course* of the GA. For example, GIGA [11] and GONZO [9] provide fitness-versus-time graphs. The ability to display genetic information about the best chromosome in each generation is also common, for instance, in both of the previously mentioned tools and also in GAPlayground [17]. Limited support for visualization of the current state of the GA at a given generation is provided by both GIGA and GAPlayground, which output textual files containing population data matrices.

One feature that is provided in several of these tools (GIGA, GAPlayground) is the ability to display phenotypic information in a meaningful manner. For example, tours could be displayed in the case of the travelling salesman problem. This facility is not yet included in GAVEL , but would make a useful addition for certain problems.

The only tool reviewed that supported visualization of the course of a GA through the actual problem search space was GONZO. This tool enables a user to observe their algorithm's search in the problem space during evolution and explore the search path and search space after evolution. High-dimensional problem space is represented explicitly in two dimensions using a scatter plot in which the size of each point in the plot illustrates the fitness of the individual chromosome that the point represents. Rather than attempt to display a complete picture of the evolution (as in GAVEL), it makes use of a movie player controller that enables the user to step forward and backward through an algorithm's evolution, one generation at a time. Alpha sliders are also provided which enable the user to identify the amount of information to be shown. This allows information at the *population* level to be visualized, but it does not retain information about individual gene origins or show how one generation relates to the preceding generation. GONZO, like GAVEL, also provides a mechanism for letting a user identify schemata of interest. In GONZO, this is achieved using the 2-D display of the problem space. The columns and rows that contain the selected values in the schema are highlighted using a colored ribbon in the margins of the problem space view and areas identified during the course of an algorithm's evolution that contain all of the selected values are identified with a colored background. GONZO would be a useful complement to the features offered by GAVEL, as it would provide a set of mechanisms for examining the search space in detail.

Although no other tool we reviewed in this category provided any means of *visually* analyzing gene ancestry or operator utility, both GIGA and GAPlayground include a very simple mechanism to at least provide limited information about reproductive operations. Both of these tools can output textual information containing information about every crossover and mutation operation performed. In each case, however, no facility is provided to analyze this information in any visual way.

B. Offline Tools

One other offline tool contains many similar features to GAVEL. This tool, VIS [36], was developed independently by Wu *et al.* The philosophy behind the tool appears in many ways to be similar to GAVEL, although the information is presented in a very different manner. VIS is organized as a collection of clickable windows displaying data in varying levels of resolution in a manner that allows the user to easily move through time and through populations of a GA run. The key feature of the tool is the ability to examine the structure of individuals in detail and includes the capability to display them in several different formats as well as access information about the immediate ancestry of each *individual*, including crossover and mutation locations. The tool also allows multiple individuals to be displayed and compared, facilitating the detection of trends in a population or during a run and allowing the user to gain an insight into characteristics of the GA such as levels of speciation, convergence, and diversity. The ability to derive and plot graphs automatically does not yet exist, but is envisaged.

Thus, although the principles behind VIS appear very similar to those that inspired GAVEL, GAVEL presents the information in a different manner and also provides some additional features. The differences are summarized below.

1) GAVEL displays the complete ancestry tree of a solution on one screen, rather than deriving the tree via a series of clickable windows.
2) GAVEL provides the ability to determine ancestry at both *gene* and *individual* level.
3) GAVEL provides a number of features to facilitate the investigation of operator productivity.
4) GAVEL provides a (limited) mechanism for visualizing how much of the search space is explored by the GA.
5) GAVEL can derive graph data automatically; however, as previously mentioned, it cannot display them automatically as yet.

VIII. Conclusion

This article has described a GA visualization system for use in analyzing performance of GAs. Its novelty lies first in using the ancestry tree of a solution found by a GA as a basis for visualizing the progress of the GA, something that no other tool yet seems to provide. Second, it facilitates an in-depth analysis of the utility of the genetic operators, leading to a greater understanding of the role they individually play during evolution of a solution. It can be used both to attempt to improve the design and settings of operators in a principled manner and also to try and explain the reasons why a GA appeared to work or arrived at a particular solution to a problem. The facility to analyze operator utility and examine the genetic makeup of solution ancestors should prove useful in either confirming hypotheses predicted by GA theory or inspiring new theory to explain observed results. In particular, the tool can be used to test if one of the fundamental cornerstones of the GA—the formation and spreading of building blocks—holds true for a given problem.

As with many other visualization techniques, the question of scalability needs to be addressed. GAs running with large population sizes and/or many generations will be difficult to visualize in the tree format due to the limitations of screen size. The screen display is currently controlled by the use of scroll bars—adding alpha sliders to control the amount of information shown as used in GONZO may partially address this point. A second drawback of the approach is the need to initially record information about every gene in every generation—although GAVEL significantly reduced this information by extracting only data relevant to formation of the family tree, this still requires large amount of diskspace in the first instance.

In summary, however, it is envisaged that despite these limitations, if it is used in conjunction with a search space analysis tool such as GONZO, it should provide a means of performing a thorough analysis of GA performance.

References

[1] T. Bäck and H.-P. Schwefel, “An overview of evolutionary algorithms for parameter optimization,” *Evol. Comput.*, vol. 1, no. 1, pp. 1–23, 1993.

[2] M. Bedau, S. Joshi, and B. Lillie, “Visualizing waves of evolutionary activity of alleles,” in *Proceedings of the 1999 Genetic and Evolutionary Conference Workshop Program*, A. Wu, Ed. San Mateo, CA: Morgan Kaufmann, 1999, pp. 96–98.

[3] J. Bertin, *Graphics and Graphic Information Processing*. Berlin, Germany: de Gruyeter, 1981.

[4] L. Booker, “Improving search in genetic algorithms,” in *Genetic Algorithms and Simulated Annealing*. San Mateo, CA: Morgan Kaufmann, 1987, pp. 61–73.

[5] H. Cartwright and G. Mott, “Looking around: Using clues from the data space to guide genetic algorithm searches,” in *Proceedings of the 4th International Conference on Genetic Algorithms*, R. Belew and L. Booker, Eds. San Mateo, CA, 1991, pp. 108–114.

[6] J. J. Collins, “Visualization of evolutionary algorithms using principal components analysis,” in *Proceedings of the 1999 Genetic and Evolutionary Conference Workshop Program*, A. Wu, Ed. San Mateo, CA: Morgan Kaufmann, 1999, pp. 99–100.

[7] T. Collins, “The Visualization of Genetic Algorithms,” M.Sc. thesis, De Montfort Univ., Leicester, U.K., 1993.

[8] ——, “Using software visualization technology to help evolutionary algorithm users validate their solutions,” in *Proceedings of the Seventh International Conference on Genetic Algorithms*, T. Bäck, Ed. San Mateo, CA: Morgan Kaufmann, 1997.

[9] ——, “Understanding evolutionary computing: A hands on approach,” in *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*. Piscataway, NJ: IEEE Press, 1998.

[10] D. Corne, P. Ross, and H.-L. Fang, “Fast practical evolutionary timetabling,” in *Evolutionary Computing AISB*. Berlin, Germany: Springer-Verlag, 1994, vol. 865, Lecture Notes in Computer Science, pp. 251–263.

[11] T. Dabs and J. Schoof, “GIGA—A Graphical User Interface for Genetic Algorithms,” Lehrstuhl Informatik II, Univ. Würzburg, Würzburg, Germany, Tech. Rep. 98, 1995.

[12] P. J. Darwen and X. Yao, “On evolving robust strategies for iterated prisoner’s dilemma,” in *Proceedings of the AI Workshops on Evolutionary Computation: Progress in Evolutionary Computation*, X. Yao, Ed. Berlin, Germany: Springer-Verlag, 1995, vol. 956, Lecture Notes in Artificial Intelligence, pp. 276–292.

[13] K. A. De Jong and W. M. Spears, “An analysis of the interacting roles of population size and crossover,” in *Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Berlin, Germany: Springer-Verlag, 1991, pp. 38–47.

[14] S. Du Toit, A. Steyn, and R. Sumpf, *Graphical Exploratory Data Analysis*. New York: Springer-Verlag, 1986.

[15] R. Dybowski, T. Collins, and P. Weller, “Visualization of binary string convergence via sammon mapping,” in *Fifth Annual Conference on Evolutionary Programming*, L. Fogel, P. Angeline, and T. Bäck, Eds. Cambridge, MA: MIT Press, 1996, pp. 377–383.

[16] H.-L. Fang, P. Ross, and D. Corne, “A promising genetic algorithm approach to job-shop scheduling, re-scheduling, and open-shop scheduling problems,” in *Proceedings of the 5th International Conference on Genetic Algorithms*, S. Forrest, Ed. San Mateo, CA: Morgan Kaufmann, 1993, pp. 375–382.

[17] GA-Playground [Online]. Available: http://www.aridolan.com/ga/gaa/gaa.html

[18] A. Wu, Ed., *Proceedings of the Genetic and Evolutionary Computation Conference, Visualization Workshop*. San Mateo, CA: Morgan Kaufmann, 1999.

[19] L. Harvey and A. Thompson, “Through the labyrinth evolution finds a way: A silicon ridge,” in *Evolvable Systems: From Biology to Hardware*, T. Higuchi, M. Iwata, and L. Weixin, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, Lecture Notes in Computer Science, pp. 406–416.

[20] C. Höhn and C. Reeves, “Are long path problems hard for genetic algorithms,” in *Parallel Problem Solving from Nature—PPSN IV*, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 134–143.

[21] J. Horn, D. Goldberg, and K. Deb, “Long path problems,” in *Parallel Problem Solving from Nature—PPSN III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds. Berlin, Germany: Springer-Verlag, 1994, vol. 866, Lecture Notes in Computer Science, pp. 149–158.

[22] T. Jones, “Crossover, macromutation, and population-based search,” in *Proceedings of the 6th International Conference on Genetic Algorithms*, L. Eshelman, Ed. San Mateo, CA: Morgan Kaufmann, 1995, pp. 73–80.

[23] L. Kallel and B. Naudts, “Candidate longpaths for the simple genetic algorithm,” in *Foundations of Genetic Algorithms 5*, W. Banzhaf and C. Reeves, Eds. San Mateo, CA: Morgan Kaufmann, 1999, pp. 27–43.

[24] S. Kirkpatrick and G. Toulouse, “Configuration space analysis of travelling salesman problems,” *J. Phys.*, vol. 46, no. 8, pp. 1277–1292, 1985.

[25] S. Li and M. Loenw, “The quadcode and its arithmetic,” *Commun. ACM*, vol. 30, no. 7, pp. 621–626, 1987.

[26] D. Mattfeld, *Evolutionary Search and the Job Shop*. Berlin, Germany: Springer-Verlag, 1996.

[27] PGA program, Dept. Artificial Intelligence, Univ. Edinburgh. [Online]. Available: ftp://ftp.dai.ed.ac.uk/pub/pga

[28] H. Pohlheim, “Visualization of evolutionary algorithms—Set of standard techniques and multidimensional visualization,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Eds. San Mateo, CA: Morgan Kaufmann, 1999, vol. 1, pp. 533–540.

[29] F. Preparata and M. Shamos, *Computational Geometry—An Introduction*. Berlin, Germany: Springer-Verlag, 1985.

[30] T. Routen and T. Collins, “Visualization of AI techniques,” in *Proceedings of the International Conference on Computer Graphics and Visualization (COMPUGRAPH 93)*. New York: ACM Press, 1993.

[31] J. Sammon, “A nonlinear mapping for data structure analysis,” *IEEE Trans. Comput.*, vol. C-18, pp. 401–408, May 1969.

[32] W. Shine and C. Eick, “Visualizing the evolution of genetic algorithm search processes,” in *Proceedings of 1997 IEEE International Conference on Evolutionary Compuation*. Piscataway, NJ: IEEE Press, 1997, pp. 367–372.

[33] W. Spears, “An overview of multidimensional visualization techniques,” in *Proceedings of the 1999 Genetic and Evolutionary Conference Workshop Program*, A. Wu, Ed. San Mateo, CA: Morgan Kaufmann, 1999, pp. 104–105.

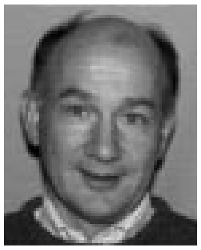
[34] A. Tuson and P. Ross, “Adapting operator settings in genetic algorithms,” *Evol. Comput.*, vol. 6, no. 2, pp. 161–184, 1998.

[35] A. Wu, K. De Jong, D. Burke, J. Grefenstette, and C. Ramsey, “Visual analysis of evolutionary algorithms,” in *Proceedings of the Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press, 1999, vol. 2, pp. 1419–1425.

[36] A. Wu, C. Ramsey, K. De Jong, J. Grefenstette, and D. Burke, “VIS: A genetic algorithm visualization tool,” in *Proceedings of the 1999 Genetic and Evolutionary Conference Workshop Program*, A. Wu, Ed. San Mateo, CA: Morgan Kaufmann, 1999, pp. 106–109.

Emma Hart received the B.A. (Hons.) degree in chemistry from Oxford University, Oxford, U.K., in 1990 and the M.Sc. degree in knowledge based systems form Edinburgh University, Edinburgh, U.K., in 1994.

She is currently a Lecturer in the School of Computing, Napier University, Edinburgh, Scotland, U.K., where she is also a Member of the Evolutionary Computing Research Group. She was a Research Associate at the University of Edinburgh for four years, where she worked on Engineering and Physical Sciences Research Council (EPSRC) grants on the application of evolutionary computing techniques to timetabling and scheduling problems. She is Chair of the European Network on Evolutionary Computing working group on Timetabling and Scheduling (EvoSTIM). She currently coholds an EPSRC grant investigating the use of hyperheuristics across a variety of domains and is also active in the field of artificial immune systems.



Peter Ross received the B.A. (Hons.) degree in mathematics from Cambridge University, Cambridge, U.K., in 1973 and the Ph.D. degree from Imperial College, London University, London, U.K., in 1976.

He is currently a Professor of Computing at Napier University, Edinburgh, Scotland, U.K. He was Director of the Artificial Intelligence Applications Institute at Edinburgh University and was the Head of the Department of Artificial Intelligence until 1998. He is an Associate Editor of the *Evolutionary Computing Journal* (Cambridge, MA: MIT Press) and has been on the programm committee of 16 international conferences in the area. He is currently the Principal Investigator on an Engineering and Physical Sciences Research Council grant that aims to investigate the potential of using hyperheuristics to solve difficult OR problems. His research interests include evolutionary computing and artificial intelligence.