Genetic Algorithm Workbench Documentation

Last Change: 30 November 1989.

CONTENTS

1. Introduction

This is a user's manual for the Genetic Algorithm Workbench program which is an interactive tool for demonstrating and experimenting with genetic algorithms using an IBM compatible personal computer. This is not a set of subroutines for inclusion in your own programs. If that is what you require, see the bibliography for details of GENESIS.

The manual commences with a description of the purpose of the Workbench in section 2.

Section 3 then describes how to use the program and gives details of hardware requirements, instructions for running the program, an explanation of the screen display, and explains how to control the program using the command menu and program control variables.

Section 4 contains a short bibliography.

Appendix A describes the detailed operation of the genetic algorithm employed by the Workbench and the effect of each algorithm input variable. It gives details of where to find further information about the theory of the different aspects of the genetic algorithm which are described. An explanation of each output variable displayed on the screen is also given.

Appendix B contains a short step by step example of using the Workbench.

Appendix C may help if you encounter problems trying to use the Workbench.

Appendix D includes an article as a general introduction to genetic algorithms and their applications.

Appendix E is a brief summary of the Workbench command menu.

2. Purpose

The purpose of the Workbench program is to allow experimentation with search and optimisation algorithms. It is primarily a tool for experimenting with different types of genetic algorithm, but is also intended for use in comparing genetic algorithms with other techniques although so far only the genetic algorithm has been implemented.

A genetic algorithm is a method for finding the peaks of difficult functions, and the Workbench program is both for demonstrating genetic algorithms and for evaluating their performance.

The idea is that you provide a function, the "Target Function" and see how quickly the selected algorithm is able to find the peak value, or indeed if it succeeds at all. You can vary the details of the algorithm used by tweaking several numeric control parameters and selecting different types of operator employed by the algorithm.

3. Using the Genetic Algorithm Workbench Program

The following sections describe hardware required and provide instructions for starting the Workbench program. The screen display is then explained followed by details of how to control the program, and finally a description of menu commands is given.

Appendix B describes a typical session using the Workbench, and will also be useful while learning how to use the program.

3.1. Hardware Requirements

To run the genetic algorithm Workbench, you require

IBM PC/XT/AT or compatible
EGA display
Microsoft compatible mouse

The Workbench has been tested on MS-DOS version 3.3 but should work on most versions of MS-DOS and PC-DOS. Reports of problems with other versions will be appreciated.

The program will work on a VGA mode screen, but unless you can put it in EGA emulation mode, the program will only use the lower two thirds of the screen, giving a squashed display.

3.2. Running the Program

Check that your system hardware is suitable for running the genetic algorithm workbench (see above).

Switch on the computer and wait for your MS-DOS prompt to appear on the screen. If you have a display adapter board that can emulate several display modes, ensure that it is in EGA mode. This might be done by setting configuration switches on the adapter card prior to switching your computer on, or by means of a special command provided with your display adapter. Refer to your display adapter manual for details of how to set it to EGA mode.

Now, ensure that your mouse driver is loaded. On some systems a command such as "MOUSE" is provided to load the driver, on others a command must be inserted into your config.sys file. Refer to the documentation for your mouse on how to install the mouse driver.

Now the final test. Insert the Genetic Algorithm Workbench disk into drive a: and type

    a:gaw

and press the <ENTER> key.

The program takes a little while to load but you should see a display similar to that shown in figure 1 (see next section), and if you move your mouse, the mouse cursor should be visible and will change as you move it over different areas of the screen.

If you don't think everything is as described here, refer to appendix C which describes possible problems and how to deal with them.

3.3. Screen Display

Figure 1 shows a screen display similar to the one you should see when the Workbench is first started. The main features of the display are as follows.

Command Menu

This is a menu of commands shown at the top left of the screen which are activated using the mouse. Each command is highlighted when the mouse cursor overlays it, and is executed if the left mouse button is pressed while the command is highlighted. See section 3.5, Program Control for details of these commands.

Algorithm Control Chapter

This is the large box which spans the top of the screen to the right of the command menu. It is called a *chapter* because it can contain several *pages,* only one of which is visible at one time. Initially, it displays a page called "Simple Genetic Algorithm" which shows a number of input variables and their values, which are used to control the operation of this algorithm. Pages can be flipped through, forwards or backwards, by clicking the left mouse button on the arrows in the top right hand corner of the chapter. Each page is described below.

Simple Genetic Algorithm Page

This is the box within the algorithm control chapter entitled "Simple Genetic Algorithm". (Note that it may be necessary to select this page using the mouse before it is displayed within the chapter box - see "Algorithm Control Chapter" above.) This page is normally displayed when the Workbench is first started and lists a number of control inputs to the genetic algorithm together with their current values. The page shows two columns of input variables. Each variable displayed shows its name to the left, a pair of arrows in the middle, and the variable's current value to the right. Note that many of the variable values are text strings. You can alter the value of any of these variables by clicking the left mouse button on the up or down arrows to the left of each value. The meaning of each variable is described in appendix A.

General Program Control Variables Page

This is the box within the algorithm control chapter entitled "General Program Control Variables". (Note that it may be necessary to select this page using the mouse before it is displayed within the chapter box - see "Algorithm Control Chapter" above.) This page contains variables related to general program operation rather than a specific algorithm. The meaning of each program control variable is described in section 3.5.

Output Variables Box

This is the box at the bottom right of the screen which contains the current values of a number of variables relating to the current algorithm. These variables cannot be altered by the user; they indicate the current state of the algorithm. Each output variable is described in appendix A.

Target Function Graph

This is the graph labelled "Target Function" and is used for entry and display of the function to be solved. Using the Workbench involves drawing functions on this graph which are then solved using the selected algorithm. After a function has been provided, a small red triangle on the x axis marks the highest peak, which is the target for the algorithm to find.

Population Distribution Histogram

This is the plot entitled "Population Distribution" immediately below the target function graph and shows the distribution of organisms by value of x for the genetic algorithm.

Output Graph

This is the graph labelled "Output Plot" and is used to display plots of various output variables against time.

Axis Value Box

This box labelled axis value is used in combination with the mouse cursor to read values from any of the graphs described above. When the mouse cursor is moved over the plot area of any graph, it changes to a cross hair and causes the Axis Value box to display the coordinate values of the corresponding graph at the point indicated by the cross hair.

Figure 1 - Example Screen Display

3.4. Menu Commands

The program is controlled entirely through use of a mouse. General commands such as starting and stopping the algorithm are invoked through a menu. The target function is input by drawing the function on a graph using the mouse cursor, and algorithm and program control variables can be altered by clicking the mouse over the up and down arrows to the left of each value.

This section describes each menu command. Program control variables are explained in the following section.

The functions of the command menu shown in the top left of the screen are as follows:

Redraw
Redraws the whole screen.

Start Alg/Stop Alg
Start/pause algorithm operation. Note that an algorithm can only be run if it the algorithm chapter is showing the corresponding algorithm page. No algorithm can run while the chapter is displaying the page relating to general program control variables.

Step Alg
No function. Currently unimplemented.

Reset Alg

Resets algorithm ready for a run. See the relevant appendix for details of reseting an algorithm.

Plot Data

Plots currently selected data (see description of "Plot data" variable later), on the output plot. This displays the selected variable against time for the duration of the current algorithm run.

Plot Targ

Re-plot the target function. After redrawing the entire screen the target function graph is cleared. This command allows the current target function to be redrawn, but note that it will have no effect until a function has been entered using the "Enter Targ" command described next.

Enter Targ

Enter or re-enter target function. After executing this command, the mouse cursor is moved to the target function graph allowing a target function to be drawn. Clicking with the left mouse button plots a point for the function, and clicking with the right deletes a point. You should draw a function which spans the full width of the x axis from left to right and uses as few points as possible. Functions with many points slow down the algorithms. When you are happy with the function, press both left and right mouse buttons together.

Quit Exit the program.

Test Tests my jump-up menus (no function). Play by all means, but this has nothing to do with the Workbench program.

3.5. Program Control Variables

The program control variables are shown on the page labelled "General Program Control Variables". The meaning of each program control variable explained below. (Algorithm control variables are explained in appendix A.)

Program control variable meanings:

Plot data

This variable is used to determine the source of data for plotting on the graph entitled "Output Plot". The selections available include values (but not all values) of output variables from those displayed in the "Output Variables" box.

O/P Plot X-max

This variable sets scale of the output plot X axis by fixing the maximum x value that can be displayed.

O/P Plot Y-max
This variable sets scale of the output plot Y axis by fixing the maximum y value that can be displayed.

Plot period
This variable determines the frequency with which the population distribution histogram is updated. A value of 1 causes an update for every iteration (or generation) of the algorithm. A value of 2 causes update every other iteration, 3 every third and so on.

Random # seed
This value is used to seed the program's random number generator each time the algorithm is reset from the command menu.

4. Bibliography

This section lists some sources of information about genetic algorithms.

A brief and very general introduction to genetic algorithms is given in appendix D which contains a copy of an article from The Guardian newspaper.

The following text is a comprehensive textbook of genetic algorithm theory and applications up to the year 1989.

Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization & Machine Learning.* Addison-Wesley.

Users wishing to experiment with genetic algorithms in their own programs may be interested in a package called GENESIS. This is a set of very useful subroutines written in C and built into a tool for experimenting with different "flavours" of algorithm. GENESIS is far more flexible than the Workbench but is not interactive and has no graphical output built in. As an integrated tool, GENESIS will run on most Unix systems, but the genetic algorithm subroutines are easily ported to any system with a standard C compiler. GENESIS can be obtained from its author:

John J. Grefenstette,
Navy Center for Applied Research in Artificial Intelligence,
Naval Research Laboratory, Washington, D.C. 20375-5000.
USA.

The author of the Workbench program is happy to correspond with users interested in learning more about genetic algorithms, or who wish to discuss their relevance to a particular kind of problem. He can be reached by writing to the following address:

Mark Hughes,
256 Milton Road,
Cambridge,
CB4 1LQ.
UK.

Internet: mrh@camcon.co.uk

5. Appendix A - Workbench Algorithms

This appendix describes the implementation of the genetic algorithm and operators used in the Workbench program.

5.1. Solving Problems with a Genetic Algorithm

A genetic algorithm evolves solutions to a problem through natural selection, breeding and genetic variation. This involves generating a population of solutions, measuring their suitability or fitness, selecting the better solutions for breeding which produces a new population. The process is repeated and gradually the population evolves towards the solution.

In the genetic algorithm Workbench, the problem is to find the value of x for which the target function has a maximum value of f(x). Each individual in the population represents a solution to this problem in the form of a candidate value for x. The suitability or fitness of the individual is simply taken by calculating the value of f(x) for the individual. This leads to an individual whose value of x corresponds to a high value of f(x) being fitter, and consequently being given a greater chance of breeding, than an individual whose value of x corresponds to a lower value of f(x).

5.2. Genotype Coding

In the same way that the genetic information of animals is coded as a string (of DNA), the genetic information of each individual, i.e. its value of x, is also coded as a string. In this case as a string of zeros or ones which can be interpreted as a simple binary code. The choice of a string representation is deliberate because it allows processes which act on strings of DNA during natural evolution to be implemented by the computer version of the genetic algorithm.

The string coding of each individual is known as its genotype in biological terminology, while its interpretation, i.e. its value of x, is referred to as its phenotype.

5.3. Genetic Algorithm Implementation

The genetic algorithm implemented by this program boils down to the following steps. (Note that a number of new terms, shown in italics, are introduced during the following steps without full explanation. These terms refer to operations that are explained subsequently.)

(1) Generate an initial population of organisms. The random number generator is seeded with the value of "Random # seed" (see section 3.5), and a new population of organisms is generated each with a different random genotype. This happens whenever the "Reset Alg" command is invoked from the main menu. The command also

resets the generation counter to 0 and clears the output variables which are updated during each algorithm run. The number of organisms generated depends on the value of the *population* input variable.

(2) Calculate and scale new fitnesses. Each new individual's fitness is calculated by reading a value of f(x) from the target function at the individual's value of x. If selected, *fitness scaling* is now done.

(3) Select individuals for breeding. A subset of the population is selected for breeding.

(4) Breed to produce new population. The set of breeders are taken in random pairs and mated to produce pairs of new organisms, the progeny.

(5) Disperse progeny into the population. The new progeny are inserted into the population, displacing existing individuals.

After the last step, the algorithm begins again at step 2, starting the next generation.

5.4. Summary of Algorithm Input Variables

Each input variable to the simple genetic algorithm is summarised below.

5.4.1. Population

This input variable determines the number of individuals in the population. That is, the number of candidate solutions being manipulated by the algorithm at any time. Too small a population and there will be little opportunity for genetic variation, too large and the algorithm will reduce to a random search.

For the problem posed in the Workbench, populations as low as 5 to 10 can be quite effective but in more complex problems where there are many more possible solutions larger populations are required. However, even for very complex problems, populations rarely exceed a few hundred individuals.

One of the reasons why surprisingly small populations can be effective is the intrinsic tolerance of noise exhibited by the genetic algorithm which arises out of the repeated sampling of small chunks of the genetic material (so called schemata) over a number of generations.

There is a discussion of the issues in selecting suitable population sizes in Goldberg 1988.

5.4.2. Fitness Scaling

Fitness scaling occurs between the production of new individuals in the population and the use of their fitness values for selection. It is a method of adjusting the probability distribution which determines the likelihood of an individual being selected for breeding. It is usually used to emphasise the relatively small differences in relative fitness when a population begins to converge. Without it, the rate of convergence will slow down as diversity decreases and most individuals in the population have similar fitnesses.

The kind of fitness scaling employed depends on the value of the corresponding input variable which has one of the following values.

None No scaling. An individual's scaled fitness value is the same as its unscaled value.

Linear

Each individual's scaled fitness f' is calculated from its unscaled fitness f according to the formula

$$f' = a.f + b$$

where a and b are chosen so that the mean scaled fitness is equal to the mean unscaled fitness of the population, and so that the maximum scaled fitness is a given multiple of the maximum unscaled fitness. The multiple is typically two.

Several methods of fitness scaling are discussed in Goldberg 1989.

5.4.3. Breeder Selection

Breeder selection involves choosing a number of individuals according to (scaled) fitness which will be used for breeding.

The number chosen depends on the number of new individuals required, which is the product of the current population size and the *generation gap.* The latter is an input variable between 0 and 1 which represents the proportion of the current population replaced during each generation.

The method of selecting the individuals depends on the value of the input variable *breeder selection:*

Roulette

This method is so named because of its similarity to spinning a roulette wheel. In effect, an imaginary roulette wheel is marked out with one slot per individual in the population, but the slots are of differing sizes giving some individuals a better chance of being selected for breeding than others. By making the slot size proportional to the (scaled) fitness of each individual, the fitter individuals have a correspondingly better chance of being selected and passing on their characteristics.

The imaginary wheel is spun once for each individual required, one individual being selected per spin. This allows some individuals to be selected more than once and others not to be selected at all.

Expected Value

There is a potential problem with roulette wheel selection because it is a stochastic process. In other words, its random element allows some individuals to be selected more often than their fitness deserves (and others to be selected less often). Expected value selection reduces this stochastic error by ensuring that no individual can be selected more than one more time than it deserves. (Obviously some stochastic error must remain because the number of times and individual is selected must be an integer whereas its "selection merit" is a real number).

Both *generation gap* and several kinds of *breeder selection* are discussed in Goldberg 1989.

5.4.4. Generation Gap

The *generation gap* input variable determines the proportion of each the population replaced during each generation. See breeder selection above.

5.4.5. Mates Selection

Following selection of a pool of individuals for breeding, pairs are taken from this pool and bred to produce a pool of progeny. The input variable *mates selection* determines how these pairs are chosen. Currently only one method is supported:

Random

Each individual chosen for mating from the breeding pool is selected at random and is immediately removed from the pool to prevent it being selected again.

In this implementation all individuals are identical and so purely random selection of a mate is always valid. However, more complicated schemes are feasible where mating is restricted in some way, perhaps to simulate the formation of niche populations or species. This is discussed further in Goldberg 1989 (p188-197). See also section 5.4.9 which describes dispersal by crowding.

5.4.6. Mating

Having selected a pair of individuals for mating, they are mated to produce new individuals which are collected in a pool of progeny. The method used is determined by the value of the *mating* input variable, but this currently only supports one method:

Simple

Simple mating produces two progeny from two parents as follows. First a copy of each parent is taken and *crossover* is applied to produce two individuals each of which receives some genetic material from both parents. Finally, *mutation* is applied to each individual which may introduce a random change to the genetic material.

Crossover and mutation are described in the following sections.

Currently only simple mating is supported, but many variations can be envisaged, for example incorporating other genetic operators than crossover and mutation. These operators are copied directly from natural processes and their are many other such operators, both natural and man made, that can be tried.

5.4.7. Crossover

As mentioned earlier, each individual represents a candidate solution to the problem in the form of a string of zeros and ones. Crossover is a process which takes two such strings and exchanges portions of the strings to produce two new strings, each of which incorporates information from both the initial strings. Many variations of the crossover operator have been experimented with. The following are available according to the value of the *crossover* input variable:

Single Point

A point is chosen at random from the first to the last but one binary digit of one of the strings. The digits following this point are exchanged between the two strings. For example, given the two initial strings and a crossover point indicated by the ’ˆ’.

0	1	0	0	0	0	0
1	0	1	1	0	1	0
		ˆ				

the following new strings would be produced.

0	1	0	1	0	1	0
1	0	1	0	0	0	0
		ˆ				

Two Point

Two point crossover is similar except that two distinct points are chosen, again randomly, and the segment between the two points is exchanged. The operator works in such a way that single point crossover is a legal special case of two point crossover. This is the case if either of the two points is at the extremes of the string.

Uniform

Uniform crossover passes along the length of the strings and chooses to take each bit from either one parent or the other according to some specified probability. The origin of each bit is independent of all the other bits and in this implementation has an equal chance of being taken from either parent. This produces more mixing of bits than the former operators.

The effect of crossover, whatever its form, is to produce new individuals which contain genetic material from two parents. No new material is introduced, and so there is a limit to the genotypes that can be produced throughout crossover alone.

Several such operators including single point and two point crossover are described in Goldberg 1989. Uniform crossover is described (and compared with several other crossover operators) in Syswerda 1989.

5.4.8. Mutation Probability

Mutation involves the chance introduction of a change to any particular bit of an individual's string. Each bit is considered in turn, and is flipped from a zero to a one (or vice versa) with probability determined by the value of the *mutation probability* input variable.

Mutation can result in new genetic material being introduced into the population, since it can produce values that were not present in either parent, or indeed in the entire population.

Typical mutation rates are of the order one in a hundred to one in a thousand bits. Much higher rates tend to disrupt the action of crossover, preventing convergence and leading to a more random type of search.

5.4.9. Dispersal

Dispersal is the method by which progeny are placed into the existing population, which requires some existing individuals to be deleted. This is done by the method indicated by the value of the *dispersal* input variable:

Random

Individuals are chosen at random from the existing population to be replaced by progeny until all progeny have been inserted. Measures are taken to ensure that progeny inserted by the current dispersal are not displaced by the insertion of other progeny.

Crowding

Crowding is a mechanism used to prevent premature convergence of the algorithm. The chance of an individual being displaced is made to depend on the degree of similarity between it and the new individual that is replacing it. When a new individual is ready to be placed into the existing population, it is compared (bit for bit) with a given number of individuals chosen at random from the existing population. The one most like the new individual is chosen to be replaced. The number of individuals chosen for comparison is determined by the value of the *crowding factor* input variable.

Dispersal by crowding is so called because it simulates competition for scarce resources by individuals which are genetically similar and so encourages the formation of niche populations. Mate selection, described in section 5.4.5 is another method that can be used to encourage the formation of niche populations. These and other methods are discussed further in Goldberg 1989 (p188-197).

5.4.10. Crowding Factor

The *crowding factor* input variable determines the number of individuals that are compared with each new individual during during dispersal by crowding (see above). A crowding factor of 1 would prevent crowding from working and be equivalent to random dispersal.

5.4.11. Elitism

Elitism is a feature that is active dependent on the value (on or off) of the *elitism* input variable.

When active, elitism ensures that the best, or fittest, individual cannot be lost from the population through dispersal. A copy of the fittest individual in each generation is kept and is re-introduced into the population if, after dispersal, no individual in the new population is at least as fit.

When elitism acts to re-introduce a lost individual it must choose an individual to be replaced. For details of how this is done, see the section entitled "Sacrifice Selection" below.

Elitism is discussed in Goldberg 1989.

5.4.12. Sacrifice Selection

The method by which an individual is selected for replacement due to the operation of elitism (see above) is determined by the value of the input variable *sacrifice selection* as follows:

Random

The individual to be replaced is chosen randomly.

Weakest
The weakest (i.e. least fit) individual is chosen.

5.5. Output variables

With each generation the display of output variables is updated. Each variable is explained below:

Generation
The number of generations (iterations of the genetic algorithm) completed so far.

Optimum Fitness
The maximum value of the function f(x).

Current Best Fitness
The highest value of f(x) for any individual in the current population.

Average Fitness
The average value of f(x) for all individuals in the current population.

Note that the above fitness values relate to unscaled fitnesses.

Optimum x
The value of x which corresponds the the maximum value of f(x) of the target function.

Current Best x
The x of the individual in the population which has the highest value for f(x).

Average x
The average of all x values in the current population.

5.6. References

Goldberg 1988
Goldberg, D. E. (1988). *Sizing Populations for Serial and Parallel Genetic Algorithms.* TCGA report no. 88005. University of Alabama.

Goldberg 1989
Goldberg, D. E. (1989). *Genetic Algorithms in Search Optimization & Machine Learning.* Addison-Wesley.

Syswerda 1989
Syswerda, G. (1989). *Uniform Crossover in Genetic Algorithms.* Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufman.

6. Appendix B - A Typical Session Using the Workbench

This appendix describes a typical session using the genetic algorithm Workbench.

To run the Workbench program you require an IBM compatible personal computer with EGA display and Microsoft compatible mouse. To start the program, make sure your mouse driver is loaded, your display is in EGA mode and type

    gaw

followed by pressing the <ENTER> key. The program does not have any command line parameters. (See also section 3.2, "Running the Program".)

Check that the program display is similar to that of figure 1 shown earlier in the manual. If you think something is wrong, refer to *problems, appendix C which describes possible problems and their solution.*

Typical use involves the following steps.

1. Click on "Enter Targ" in the menu and enter a function extending from leftmost to rightmost points on the graph. (Note that clicking the left mouse button sets a point, clicking the right deletes a point, and pressing both together ends input of the function). You should enter a function starting from the graph origin (bottom left), and finishing at the bottom right of the graph. To end entry of the function, press left and right mouse buttons simultaneously. If you go wrong, click on "Enter Targ" and try again.

2. Click on "Reset Alg" to initialise the genetic algorithm. Notice the histogram of population distribution in the bottom left hand corner is redrawn, along with the output variables in the box at the bottom right corner of the screen.

3. Click on "Start Alg" which causes the algorithm to run. Note that the menu option changes its name to "Stop Alg", and so clicking on it again will pause the algorithm.

While running, a count of generations is maintained in the "Output Variables" box, along with several other algorithm variables. Every so often, the histogram is updated as the population attempts to converge on the value of x which corresponds to the highest peak in the target function input earlier.

At any time, a plot of average fitness against time can be produced by clicking on "Plot Data". The algorithm can be paused/re-started by clicking on "Stop Alg" and "Start Alg". New target functions can be provided, as described earlier, and the algorithm allowed to run with the current population distribution or with a new distribution by clicking on "Reset Alg".

While the algorithm is paused, various program variables can be changed by clicking the mouse button while the cursor is hovering over the *up* and *down* arrows next to values displayed in the large box spanning the top of the screen. Pressing and holding the mouse button enables variables to be changed quickly.

Note that the large box spanning the top of the screen contains two pages, only one of which is displayed at a time. These can be thumbed through by clicking on the arrows at the very top right of the display, immediately to the right of the copyright message. The first page is called "Simple Genetic Algorithm" which allows algorithm variables to be adjusted. The second page, called "General Program Control Variables", allows selection of different data for plotting on the output graph, changes to the scale of output graph axes and so on.

Note that the algorithm will only run while the "Simple Genetic Algorithm" page is selected. Alternative algorithms, simulated annealing for example, will be implemented by providing additional pages of algorithm variables.

7. Appendix C - Problems and How to Fix Them

This appendix is intended to help sort out problems encountered when trying to run the genetic algorithm Workbench program.

If you are unable to fix any problems using the list of potential problems and solutions which follow, it is wise to take the following steps before giving up in despair:

(1) Prevent installation of any unnecessary resident programs such as pop-up utilities, disk caches or command line editors. These are often installed by commands in your autoexec.bat file and could be the source of your problem.

(2) Prevent installation of any unnecessary device drivers. These are usually installed by commands in your config.sys file such as "device=filename".

The only resident program or device driver that you will need installed is your mouse driver which may be installed by either of the above methods depending on the software supplied with your mouse.

Problem: Display is squashed

The top third of the screen is blank, but everything is displayed correctly in the lower two thirds of the screen. The mouse cursor may also be missing.

Your display adapter is in the wrong mode, possibly VGA mode. Refer to your display adapter manual for details of how to set it into EGA mode.

Problem: Display corrupt or blank

Your display adapter is probably in the wrong mode. Refer to your display adapter manual for details of how to set it into EGA mode.

Problem: No mouse cursor

The mouse cursor is not visible but it is possible to highlight options in the command menu by moving the invisible cursor towards the menu and "circling" with the mouse.

This probably indicates a problem with your mouse driver. It may not be compatible with this mode of your display adapter. Note that the Workbench program operates in *graphics* mode, and so this problem may occur even if you have used your mouse with the display in EGA *character* mode (which would display a block mouse cursor).

8. Appendix D - General Introduction to Genetic Algorithms

The following is an article which appeared in the Guardian newspaper on 14 September 1989.

Why Nature Knows Best About Design

Engineers now need a helping hand from Nature in order to solve their problems. Mark Hughes reports.

All life on earth, including its most intricate and ingenious features is the product of a genetic algorithm, known more commonly as evolution. However, genetic algorithms need not be confined to nature. They can be used to help solve many design and optimisation problems. Computer implementations of genetic algorithms are being used to tackle difficult problems in fields as far ranging as turbine blade design, automatic integrated circuit layout, and even in the training of neural networks.

All living things carry a kind of "blue-print" for their construction in the DNA of each living cell. Over a period of time, changes (e.g. mutations) occur to the DNA giving rise to organisms which are more likely to survive, and so have a greater chance of passing their improved characteristics on to future generations. Of course, not all changes will be beneficial, but those which are not tend to die out. This is evolution. It is analogous to engineers making design changes in order to improve their company's product and so gain a competitive advantage or increase profitability. The genetic algorithm is the mechanism invented by nature for trying out alterations to DNA.

In the past, evolution has been perceived as a slow and hap-hazard process, relying on random mutations, and thus unsuitable for use in engineering. This perception caused problems to scientists trying to explain the rapid rate of evolution evident from the fossil record, and has been shown to be false. Although scientists have been aware of genetic operators other than random mutation for some time, it was not until the 1970s that a mathematical analysis revealed their importance (Holland 1975). Holland showed that nature's genetic algorithm was highly efficient at search and optimisation, and in no sense hap-hazard.

In engineering, computer simulations of genetic algorithms can be used to evolve better designs for a variety of systems. The computer is used to maintain a population of competing designs each with its own computer representation of DNA (usually a binary string). Here, instead of determining animal characteristics such as size, number of limbs and eye colour, the "DNA" is decoded to produce design characteristics such as lengths, angles, equations or rules. At each generation, the better (cf. fitter) designs are chosen to reproduce, and operators borrowed from nature are used to make changes to their "DNA" in the search for improvements. The designs are then tested by decoding the "DNA", and over several generations the population evolves and improves the criteria chosen by the engineer.

The freedom to select fitness criteria allows genetic algorithms to be applied in many fields. For example, you may wish to evolve rules for trading in financial markets, improve the aerodynamics of a vehicle, or simply solve an abstract mathematical function. But why use a genetic algorithm, when in the last case for example, there are plenty of established methods for solving mathematical functions?

The reason is that existing methods are fine so long as the problem is not too complex. A genetic algorithm allows extremely difficult functions to be solved efficiently - even the design of a living organism.

In engineering terms, the strengths of genetic algorithms can be summarised by their abilities to cope with a variety of very difficult problems, to work without prior knowledge about the function being optimised, to optimise "noisy" functions, and to do without secondary information such as gradients. In plain language they can cope with the difficulties represented by real-life problems which are generally insoluble by other methods.

A recent and most impressive testament to the fact that genetic algorithms are now coming of age has been provided by General Electric in the USA who used the technique to design an improved gas turbine blade. Their computer model indicates efficiency improvements of 2% for the design, a significant saving in this field, and they are currently spending around $1m verifying the prediction. If this succeeds they will spend $70m re-tooling their production line to produce the new type of blade.

Applications in addition to those mentioned already include gas pipeline management, medical image registration, adaptive filter design and mechanical structure optimisation. But engineering is not the only area to benefit. Genetic algorithms have been used to generate rules for financial trading systems, to classify forensic evidence, and to identify insurance risks.

A lack of computer power has been a factor limiting the usefulness of genetic algorithms as they sometimes require large amounts of computation, particularly if complex computer models are involved. But this is no longer such a problem because computing power has become relatively cheap. Even very difficult problems can now be solved because of the efficiency with which genetic algorithms can be implemented on today’s parallel computer architectures.

Engineering problems are getting so complex that man’s intuitive approach to design is becoming too primitive. Genetic algorithms are one of the tools that engineers are using to make up for their short-comings.

Reference: Holland J. H. (1975). Adaptation in Natural and Artificial Systems. Univ. of Michigan Press: Ann Arbor, MI.

9. Appendix E - Main Command Menu

The functions of the command menu shown in the top left of the screen are as follows.

Option	Function
Redraw	Redraws the whole screen
Start Alg/Stop Alg	Start/pause algorithm operation
Step Alg	No function
Reset Alg	Generate initial random population
Plot Data	Plot graph of average or best fitness
Plot Targ	Re-plot the target function
Enter Targ	Enter or re-enter target function
Quit	Exit the program
Test	Tests my jump-up menus (no function)