# Genetic Programming for Prediction and Control

**Dimitris C. Dracopoulos**
Dept of Inf Systems & Computing
Brunel University
London, UK

**Simon Kent**
Dept of Inf Systems & Computing,
Brunel University
London, UK

**Abstract**

The relatively "new" field of genetic programming has received a lot of attention during the last few years. This is because of its potential for generating functions which are able to solve specific problems. This paper begins with an extensive overview of the field, highlighting its power and limitations and providing practical tips and techniques for the successful application of GP in general domains. Following this, emphasis is placed on the application of genetic programming to prediction and control. These two domains are of extreme importance in many disciplines. Results are presented for an oral cancer prediction task and a satellite attitude control problem. Finally the paper, discusses how the convergence of genetic programming can be significantly speeded up through bulk synchronous model parallelisation.

**Keywords:** Genetic Programming, Parallel Computing, Prediction, Evolutionary Control, Evolutionary Computing

## 1 Introduction

Humans have always been fascinated with the idea of building intelligent machines for two main reasons: to mimic animal-like intelligent behaviour, and to provide functional solutions for difficult, complex tasks which cannot be addressed otherwise. The traditional computer approach of producing software to solve particular problems cannot be applied in general, in areas such as speech recognition, vision or robotics. These areas require advanced techniques from AI, which are able to cope with limited knowledge of problem domains and to develop solutions, in symbolic or numerical form, for the particular problem under consideration.

Genetic programming (GP), an extension of the well known field of genetic algorithms (GAs), belongs to the class of "intelligent" methods. With genetic programming there is no need to formulate an algorithm which is later coded in a particular programming language. Instead, GP evolves an algorithm by drawing on Darwinian evolutionary theory [4]. The principle of survival of the fittest has produced extremely complex functional solutions through biological evolution in the form of plants and animals. The biological evolutionary process did not analyse or understand the problem of any one species living within its specific environment, instead it allowed the best creatures to survive and procreate, and culled those which were below par.

Nature has provided the evidence of evolved systems that possess levels of complexity far beyond any man-made system. GP harnesses the evolutionary process in a computer, evolving programs instead of creatures. The theory then suggests that computer evolution could similarly

evolve very complex computer programs. In practice, GP is capable of finding successful solutions to various tasks, however there are certain limitations due to time constraints. In contrast to GP, the evolution of complex structures in nature took millions of years.

In this paper, an overview of the current state of GP is given, with particular emphasis on prediction and control tasks. The next section provides background material in the field of GP, the shortcomings of the field are noted, and measures to reduce their influence are suggested. Section 3 summarises some of the most successful applications of GP. Section 4 concentrates on prediction applications, with application to the oral cancer prediction problem, while Section 5 focus on the application of evolutionary computing in control theory, with application to the satellite attitude control problem. Finally, Section 6 shows how GP runs can become faster through bulk synchronous model parallelisation.

# 2 Genetic Programming

Although a number of people have claimed that they first introduced genetic programming, the field became publicised in the early 90's with the publication of Koza's book [20] on genetic programming. GP has its foundations in the well established field of GAs, and an earlier paper [19] actually refers to GP as hierarchical genetic algorithms. The principles which underly GA and GP are very similar. The most significant difference between them is that the data structure evolved by GAs is usually a string, whereas in GP the evolved structure is a tree, consisting of function and terminal nodes. This tree structure allows for much more diverse and complex structures of differing sizes, which are easy to map to mathematical structures or computer programs.

## 2.1 The Simulation of Evolution

The GP process is an iterative process. An initial population of individuals (programs) is randomly generated at the start of the run. The programs are written in a prespecified language consisting of functions and terminal variables. Each individual program in the population is evaluated against a function (fitness function), to measure how well it performs against the problem it is addressing. The result of the evaluation is known as the fitness of the individual. The next stage is the evolutionary stage where a new population of programs is created from the old population. Having measured the fitness, this information is used as the basis to decide which programs are better than others. During the evolutionary phase, those programs with a higher fitness are more likely to contribute either parts or all of their structure to individuals in the new population. This process of fitness evaluation and evolution is repeated as the GP process efficiently searches for an optimum or near optimum solution to the problem at hand. The flowchart of the process is shown in Figure 1.

The methods by which new offspring are generated in nature, namely sexual and asexual reproduction, are mirrored in GP. It is very unlikely that a perfect solution to the problem will be seen in the first, randomly generated population. Therefore in subsequent generations, new individuals must be produced by combining "pieces" of individuals found in the previous population. In GP this operation, akin to sexual reproduction, is called crossover. It is hoped that over successive generations, all the useful sub-components initially spread throughout the population will come together in a single individual which will offer a good (near optimal) solution. To avoid the loss of good individuals which have already been formed, and to improve

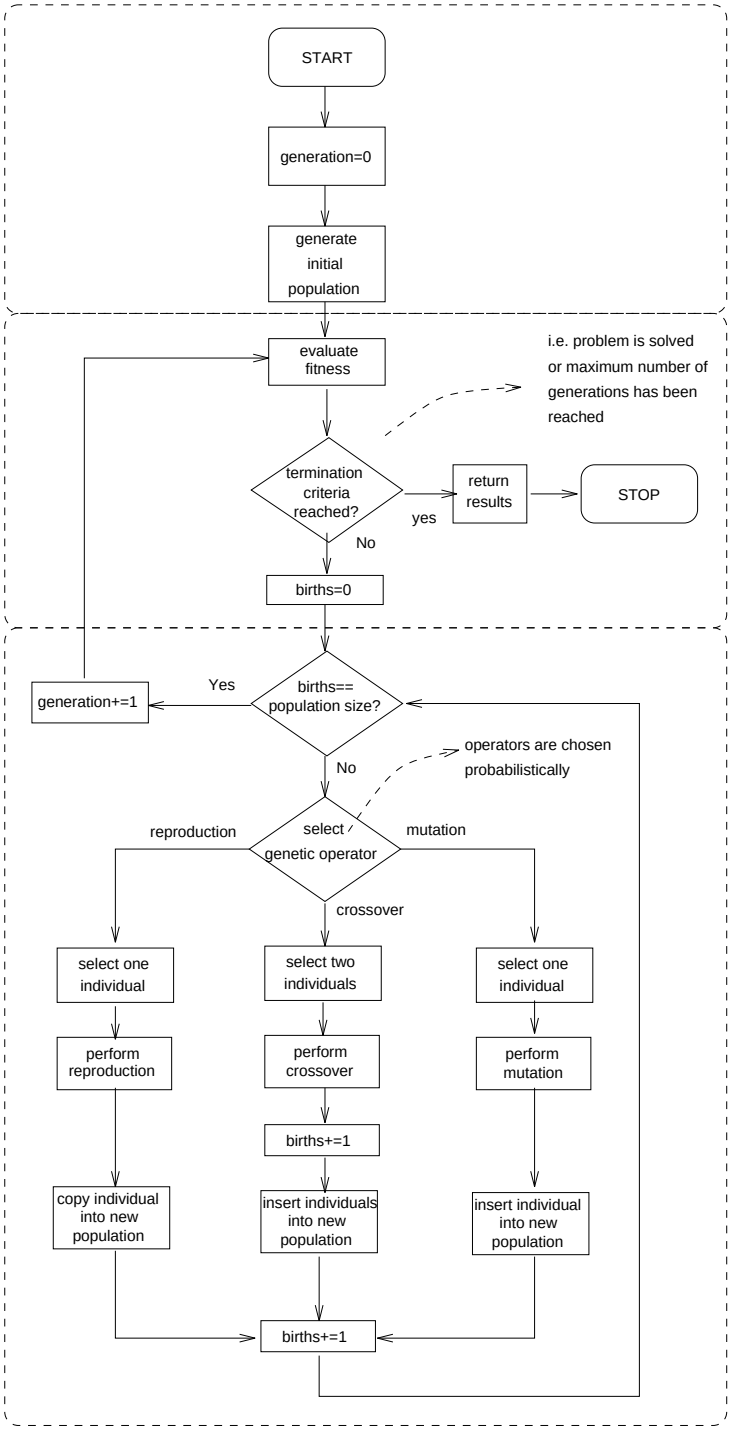



Figure 1: Flowchart of the GP process

the convergence of the algorithm, reproduction (asexual) is also used to copy some of the better individuals in their entirety to successive generations. In nature these processes operate on the DNA of the organism involved, whilst in GP they operate on the tree structures as demonstrated in Figure 2.

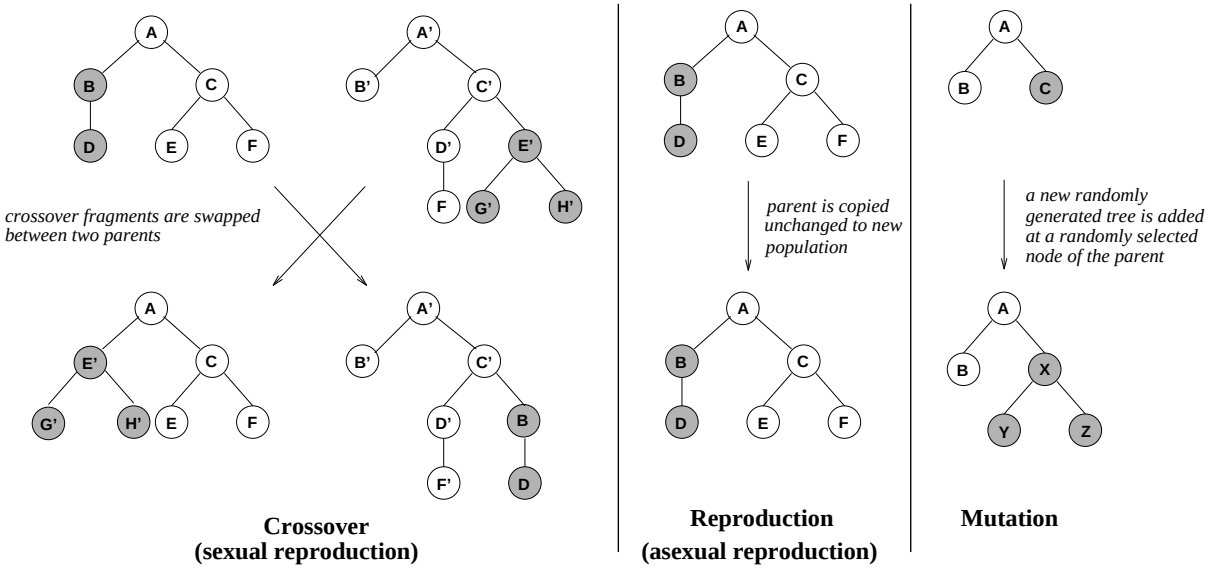



Figure 2: The main operators used in GP

Crossover and reproduction are the main operators used, typically to generate 90% and 10% of a new population respectively. Sometimes, it may be useful to introduce a mutation operator applied to individuals with a much lower probability (typically only 1%). The mutation operation involves the random selection of an individual's subtree and the replacement of this subtree by another randomly created subtree. This alteration can be useful to prevent premature convergence to a solution which is sub-optimal, and can be seen as the addition of a small amount of random noise to the whole process.

Other more specialised genetic operators exist, and full details of the most common of them can be found in [20]. For some problems it may be appropriate to adopt special operators which apply some heuristic to the domain of the problem. In parallel with molecular biology, the GA/GP data structure is often referred to as the genotype, whereas the expression of the data structure in the problem domain is known as the phenotype. Xiao, Michalewicz, Zhang and Krzystof [40], for example, introduced operators which modified a robot path (phenotype) as well as the the more traditional crossover, reproduction and mutation operators which were applied to the data structure representing the robot path (genotype). Operators which edit mathematical or logical structures in order to simplify them also operate on the phenotype rather than the genotype as they require domain specific knowledge to work.

## 2.2 A Programming Language for the Problem

In the general case genetic programming evolves trees or programs. However, to solve a specific problem, the trees must be tailored by specifying a language or representation scheme which can express a solution to the problem. The scheme is specified in terms of functions which reside in the internal nodes of the tree and take one or more arguments, and terminals which sit at the leaf nodes of the trees.

The definition of an appropriate language for the specific problem being solved is of paramount importance. A necessary condition which must be met in order to be able to solve a particular problem using GP, is that of sufficiency of the terminal and function sets. If a solution cannot be expressed in terms of the specified terminal and functions, then it is pointless to use any technique to search for a solution. On the other hand, the search space grows exponentially with the number of terminals and functions, therefore their choice must be appropriate so as not to increase the search space unnecessarily. It is obvious that a search in an extremely large search space can be very difficult or sometimes infeasible in "normal amounts" of computational time.

A second condition which must usually be met is that of closure: the value of any terminal and the values returned by any function, must be processable by all other functions. That is, an evolved program must always be able to run. To this end, certain measures may need to be taken with some mathematical functions. A common example is division, which is undefined when dividing a number by 0. In this case a wrapper must be applied to the function, to ensure that it does not simply generate an error, but instead returns a value, such as 0 or 1, which can be coped with by the other members of the function set. This is known as protected division [20]. An alternative to the closure condition, is that of strong typing [31]. This involves having functions and terminals which can return different types. Type checking mechanisms must be adopted during the creation of the initial generation and subsequent evolution of the population such that type inconsistencies are removed. In most cases however, a suitable closure can be defined for a problem without the complication and the extra time overhead of strong typing.

## 2.3 The "fitness" of a program

The choice of the fitness function used to measure the "goodness" of a a candidate program is a key point for the successful application of GP. The fitness is the single means by which the GP process can choose which genetic material should be propagated from generation to generation. The GP process which relies on the the feedback provided by the fitness value, resembles that of reinforcement learning techniques [6, 34]. Individuals are punished (extinguished) if their fitness is low, while those with above average fitness are rewarded, through their selection for further reproduction.

During fitness evaluation each program in a GP population is evaluated against the problem. In some cases, it may be possible to link the program directly to the real world application. For genetically generated art or music where the fitness is subjective, a human may be required to enter a score. However, the most common applications involve training against a simulation as in the Artificial Ant problem (Section 6.1) where a single fitness case may suffice, or against some collected data as in the Oral Cancer Diagnosis problem (Section 4) where many fitness cases are presented to the evolved program (a classical example of supervised learning in which a set of input and desired outputs is provided). In the latter, the fitness is aggregated over all the fitness cases to arrive at a fitness for that individual. Several different types of fitness are described by Koza [20] (chapter 6) but the important feature of fitness is that it provides a continuous scale of program performance.

## 2.4 Survival of the Fittest

The underlying principle of evolution is that of "survival of the fittest", whereby the good genetic material perpetuates at the expense of the bad genetic material. In nature this occurs through

competition between members of a species firstly to survive and secondly for the right to mate. In evolutionary computing, the selection of individuals to participate in the genetic operations must be made according to an algorithm.

The most popular method, known as proportional fitness or roulette wheel selection [10], involves calculating, for each individual its proportion of the total population fitness, such that the sum of all the individual fitnesses equals 1.0. On a line between 0.0 and 1.0, the better individuals will take up a larger proportion of the line than will less fit individuals. An individual is then chosen by generating a random number between 0.0 and 1.0 and then selecting the individual in whose region of the number line the random number falls. By repeating this each time an individual needs to be chosen, the better individuals will be chosen more often than the poorer ones, thus fulfilling the requirements of survival of the fittest.

Another method often used is tournament selection. It is closer to the type of competition in nature where two animals will fight for the right to mate. Various rules exist for tournament selection, but in general two or more individuals are chosen randomly from the population, and their fitnesses are compared. The individual with the highest fitness wins. This type of selection does not require the evaluation of the fitness for all individuals in the population. It may be possible to evaluate only the fitness of an individual as and when it is chosen for a tournament. An alternative (rank selection) is to sort the individuals' fitnesses and rank them accordingly. During the tournament, the ranks can be compared rather than the fitnesses. The relative merits of different selection schemes are discussed by Blickle [3] in the context of GA.

## 2.5 The GP Population

To allow evolution to progress, sufficient diversity must exist in the population as a whole. In nature, a loss of genetic diversity brought about by incestuous behaviour shows itself in genetic abnormalities. In GP, a loss of diversity will result in the premature convergence of a run to a sub-optimal solution. The mean fitness will become close if not equal to the fitness of the best individual. The population is "polluted" with the same genetic material such that new individuals which differ sufficiently from the rest of the population can no longer be generated.

The problem of loss of diversity and premature convergence can be caused by a number of factors. If there is insufficient diversity at the outset, then this is clearly going to cause rapid convergence. A simple way to ensure diversity at the beginning of the run, is to ensure that each randomly generated program is unique. A population which is too small for the problem being tackled is another way in which insufficient diversity may occur.

Assuming that the initial population is sufficiently varied, it may still be possible for diversity to be lost during the run. For example, the adoption of an elitist approach whereby the top $n$ individuals are always copied to the next generation may speed up convergence, but may do so towards a "bad" solution (premature convergence). The use of asexual reproduction with a very high probability in combination with the proportional fitness selection method, may also cause loss of diversity.

The evolutionary process described so far evolves a new population which replaces the current one. An alternative, which is more similar to natural evolution, is to generate new population by just replacing one or two individuals in the current population. This steady state GP (the equivalent of the current state GA [35, 39]) is less prone to premature convergence and it usually improves the quality of solutions found by the GP run.

Another approach to maintaining diversity is to divide the population up into smaller, sub-populations referred to as demes. If communication of genetic material is restricted to a small amount, the convergence of the global population is slowed, as diversity is maintained. This is because every few generations new genetic material is introduced into each deme as individuals are migrated between sub-populations. Although demes can be implemented in a single processing environment, they are very amenable and ideal for parallelisation as discussed in Section 6.

### 2.6 The Control Parameters of GP

Besides the function set, the terminal set, the fitness function and the genetic operators applied to the genetic programming process, a number of other control parameter must be determined, in order to apply the GP approach to a problem. First, associated with each genetic operation there is a probability $p$, which determines how often the genetic operation is applied to the current population. As mentioned previously, the crossover probability $p_c$ is quite high with the other operators receiving smaller values.

A second control parameter which has to be determined is that of population size. A typical value of 500 is used many times, but depending on the complexity of the problem and the size of the search space, this value might need to be increased or decreased.

If left unchecked, the size of the data structures in GP would grow to an enormous size as evolution progressed. To stop this, limits are set on both the size of the initial, randomly generated programs, and the subsequent size to which they may evolve. If the use of a crossover or mutation operator with individuals would give rise to oversize offspring programs, the operation is cancelled, and a new operator/program combination is tried. The limits may be set in terms of the maximum depth to which trees may grow, or the maximum number of nodes which trees may contain.

The termination of a GP run usually happens when a good-enough solution has been found. However, there are many cases where this is not possible, either due to computer usage limitations or due to the fact that a solution does not seem to be found in "normal" time limits. For this reason, a maximum allowed number of generations $G$ has to be introduced. If a solution has not been found within $G$ generations, then the GP run is terminated. Then one could try to change some of the parameters, terminal and function sets or simply rerun the GP using exactly the same data and parameters as the run which failed. This is a typical case and something which newcomers to the GP have to keep in mind. GP belongs to the class of probabilistic algorithms which give a different result every time they are run. Different initial population (due to different random seeds), and the application of different genetic operators and in a different order (probabilities $p$ of genetic operations) result in different answers. Hence one has to run GP multiple times (typically 10-15) before concluding that some changes have to be made in the GP set up. Given the limited time and resources available for GP runs on a specific problem, it is generally more efficient to use fewer generations with larger population sizes, rather than the other way round. This is because populations converge much faster early in the run, so to maximise the improvement per generation, it is better to restart runs rather than continue them.

A common problem encountered in the case of multiple data (supervised learning paradigm), is the stopping point of training. Certainly, many times a perfect solution $x_1$ can be found in relation to the training data. However, when the solution is required to generalise well on previously unseen data (something which is required for real machine intelligence), the generalisation

error has to be considered as well. That is, assuming that the stopping point of GP was at a solution $x_2$ with larger training error than the perfect (in training) one—then in reality this solution is better than $x_1$, if its generalisation error is smaller than the $x_1$ (subject to sufficiently small training error). To determine which stopping point is ideal, the available data should be split into three sets: training, validation and test, and these sets used as a decision for the stopping point of the run. The details of this approach can be found in [6] and also in the other paper in this issue by the first author.

### 2.7 GP problems and solutions

The most obvious problems with GP are related to computer hardware. Given infinite power and memory, GP could solve any problem (although this is also true of a random search). As the complexity of the problems tackled increases, the computational and memory requirements tend to increase very rapidly. This is because more complex problems typically require larger function and terminal sets. This increases the search space, which may mean that larger populations are required. Larger populations will obviously take longer to evaluate, and the time taken will increase still further if the simulator used in fitness evaluation is more complex. A larger population will require more memory. Even if the host machine has limited memory, the process should still be able to run if the operating system supports virtual memory. The drawback here is further increases in run time if a large amount of swapping takes place between real memory and hard disc.

One claim of GP is that it evolves solutions which can later be examined by humans. For instance it can be used to generate a function to approximate some data. The researcher can examine the function and may be able to gain some insight into the data. In reality, the evolved structures can be rather unwieldy, containing hundreds of nodes. Mathematical or logical structures may be edited and simplified automatically either at the termination of the run, or during its progress. Another approach is to introduce the idea of 'parsimony' into the fitness. In this way the fitness is based not only on the problem being solved, but also on evolving a neat, compact solution to the problem.

A method to introduce parsimony in the solutions found is the extension of GP to automatically defined functions (ADF). According to this, GP is able to evolve subroutines (ADF) as part of the large general solution routine. The details of the incorporation of ADF to the standard GP is found in [21]. ADFs have also been shown to enable GP to solve problems more quickly than plain GP, or in some cases to solve problems which plain GP cannot.

The improvement of GP as a technique is still an area of very active research. It will be of great benefit to the GP field if the problems discussed above are addressed, such that convergence to optimum or near optimum solutions can be achieved in a smaller number of iterations.

## 3 Applications

GP has developed very rapidly since 1992, with the publication of over 600 papers at conferences and in journals. This section provides an insight into some of the wide ranging areas of GP research. More information on ideas for future work in GP may be found in [22].

It is fitting, as GP has its roots in Biology, that there are problems in biology and medicine to which GP may be applied. GP as an automated diagnostic tool is presented in Section 4, but

there are also problems in more theoretical molecular biology where there is currently extensive research into the identification and classification of DNA and protein sequences. Traditional approaches to research in molecular biology are very labour intensive and the structures which are being researched are enormous. The human genome contains roughly 100,000 genes, each containing around 1,000 bases. Intelligent techniques such as GP are being adopted [13, 14, 24] to support the traditional methods, and thus speed up research.

One of the most interesting areas of GP application is in the evolution of electrical hardware. Koza *et al* [25] have used GP to evolve analogue electronic circuits. The technique uses component creating functions and connection modifying functions to generate the final circuit by operating on a simple embryonic initial circuit. The evolved circuits are evaluated on a patched version of the freely available SPICE (Simulation program with Integrated Circuit Emphasis) simulator [33]. This is a very computationally expensive application which uses parallel computing as described in Section 6.

Over the past couple of years some very interesting work has been developed on the direct evolution of Hardware. This Evolvable Hardware (EHW) [16] field has arisen only due to the availability of the Field Programmable Gate Array (FPGA). This is a large array of cells which can be instructed to act as one of a number of logic gates. The device may be programmed as many times as required, thus allowing direct, hardware evaluation of GA/GP structures. One, or a mesh of FPGA devices may be connected to a standard PC hosting the evolutionary process. Work in this area was pioneered by Thompson [36] and has more recently been adopted by other researchers [26, 28]. The specialist hardware required undoubtedly presents a barrier to entry into this area, but the promise of faster GP is likely to give rise to a considerable increase in research in this area.

GP has yet to make a major contribution to industrial applications, but there is certainly interest in the commercial sector from organisations such as the Ford Motor Company, British Telecom and the British Defence Evaluation and Research Agency (DERA).

# 4 Oral Cancer Diagnosis

There is a wide range of problems which historically have required experts to examine samples of data, and from their analysis assign a classification or make a prediction. The experts draw on their previous knowledge gained through formal training and experience.

A particularly important area where such prediction is used is in medical screening programmes. The idea of such programmes is to detect potentially harmful conditions in patients sufficiently early, to enable effective treatment. The manual screening process is labour intensive, and therefore expensive, with the consequence that only a limited number of patients are screened. The less patients that are screened, the less problems will be identified early enough for treatment.

Computer diagnosis enables very large patient samples to be analysed, either as the only method of screening, or as a preliminary screening process to identify those most at risk of a particular condition and therefore in need of more thorough manual screening. A specific example of such screening is for oral cancer and pre-cancer.

Data was made available from the Eastman Dental Institute following a previous study [18] which provided information on patient's lifestyles and habits. It was hoped that GP would be able to learn a rule to classify patients into two groups. The rule should predict whether or not a patient was 'at risk' from oral cancer or pre-cancer.

### 4.1 Patient Data

The data used for the project was derived from data collected from questionnaires distributed to over 2000 dental patients. The data were presented to the GP diagnosis system by preparing a number of predicates shown in Table 1. For each patient, the system was therefore presented with a `true` (1) or `false` (0) value for each of the 12 predicates. In addition to the data there was a diagnosis for each of the patients. This diagnosis was from a specialist who had access to all the necessary definitive diagnosis aids, such as biopsy, at his disposal.

| |
|---|
| 1. Gender is female (i.e. 0=Male, 1=Female) |
| 2. Age $> 50$ |
| 3. Age $> 65$ |
| 4. Age $> 80$ |
| 5. Age $> 95$ |
| 6. Have been smoking within last 10 years |
| 7. Currently smoke $\geq 20$ cigarettes per day |
| 8. Have been smoking for $\geq 20$ years |
| 9. Beer and/or wine drinker |
| 10. Fortified wine and/or spirit drinker |
| 11. Drink excess units (i.e. Male $> 21$, Female $> 14$ |
| 12. Not visited dentist in last 12 months |

Table 1: Patient lifestyle and habit predicates

Some of the predicates need explanation. Firstly, the age of the patient from the questionnaire was broken down into a ranges of 15 years. All patients in the trial were over 40 years of age, which is why the ranges start so high. There are subtle differences in the smoking predicates. The first specifies if the patient has smoked at any time in the last ten years, whether or not they currently smoke. The next predicate is true if the patient currently smokes 20 or more cigarettes per day. The final predicate reflects whether the patient has, at any time during their life, smoked for a period of 20 or more years. The alcohol predicates are true if the patient drinks any of the respective drinks, and the excess units predicate is set if the patient drank over the recommended weekly units of alcohol. At the time of the study this was 21 units for men and 14 for women.

Using this method of representing the data, each patient's details were reduced to a 13-bit string consisting of the 12 predicates plus a diagnosis bit. A training set and a test set were used. The training set contained 991 individuals of which 948 had negative diagnoses and 43 had positive diagnoses. The test set contained 132 records, 121 with negative diagnoses and 11 with positive diagnoses. A lot of records seem to have been lost from the original 2000 patients questioned. This is because the data contained a lot of negative records, and very few positive records. a large number of negative diagnosis patients were removed to balance the sets somewhat.

## 4.2 GP Approach

It has become fairly standard practice in the GP field to present a "tableau" which defines the parameters for a particular application of the technique. This tradition is observed for this problem in Table 2. This is expanded on below.

| | |
|---|---|
| Objective: | Evolve a rule to predict the presence of oral cancer or pre-cancer from patient data. |
| Terminal set: | 12 boolean values corresponding to data predicates |
| Function set: | `AND, OR, NOT` |
| Fitness: | Special 'shifted' fitness as described in text |
| Fitness Case: | 991, 13-bit strings corresponding to patient data. |
| Parameters: | population = 500, generations = 600 |

Table 2: Tableau for Oral Cancer Diagnoses

### Function and Terminal Sets

The function set is a very straightforward set of of three logical operators: `AND`, `OR`, and `NOT`. A solution to the problem will therefore be a logical expression using patient attributes as variables. The patient attributes are instantiated in the evolved rules in the terminals. There are therefore 12 bit-fields corresponding to the predicates in Table 1. The programs which are evolved evaluate to either `true` for a positive diagnosis, or `false` for a negative diagnosis.

### Fitness Measure

The fitness measure needs to reflect how well an evolved program performs over all the records in the training set. To this end, there are two simple candidates for measuring fitness. The first is the mean-square error:

$$\frac{1}{n}\sum_{i=1}^{n}(o_i - p_i)^2 \ ,$$

where $n$ represents the number of patient records against which the individual was evaluated, $o_i$ is the observed diagnosis of the patient, and $p_i$ is the diagnosis predicted by the genetic program.

The problem with such a measure is that it steers the evolutionary process in a subtly wrong direction. There are some programs in the population which are contradictory — they always evaluate to `false` irrespective of the input values. Contradictory individuals will inevitably appear in the first, randomly created generation of the evolutionary process. Because there are a disproportionately high number of training records with negative diagnoses, these contradictory programs will perform very well, correctly diagnosing most of the training examples. The GP process will rapidly converge; the individuals will have high fitness , but will always predict a negative diagnosis.

A second method is simply a proportion of correct diagnoses:

$$\frac{TP + TN}{TP + FP + TN + FN} \ ,$$

where $TP, FP, TN, FN$ represent the number of true positive, false positive, true negative and false negative diagnoses respectively. This measure again falls down due to the high number

of negative diagnosis records present in the training data, encouraging rules which only predict negative diagnoses.

To encourage convergence towards a good solution to the problem, a slightly different fitness measure was devised which more accurately rewards a good individual. The ms-error evaluates to 0.0 for a program with a perfect prediction rate, and 1.0 for a completely useless individual. It was instead decided to adopt a measure which rewards for true, positive diagnoses and punishes for false, negative diagnoses. The measure is defined as:

$$\frac{\frac{TP}{TP+FP} - \frac{FN}{TN+FN} + 1.0}{2.0}.$$

This fitness allows the addition of between 0.0 and 1.0 for correctly predicted positive diagnoses, and the subtraction of between 0.0 and 1.0 for incorrectly predicted negative diagnoses, resulting in a total range of -1.0 to +1.0. This is then normalised so that the contradictory individuals which previously caused problems are centred at 0.5, thus relatively reducing their fitness. The optimum rule which correctly predicts all positive diagnoses and does not falsely predict any negative diagnoses is valued at 1.0. The worst case is at 0.0, where the rule incorrectly diagnoses all patients. This method rewards for correct diagnoses and punishes for incorrect diagnoses more effectively than the previous two methods. This shifted fitness method is the one which was used in all diagnosis experiments.

### 4.3 Results

The performance of the evolved rule is measured using a range of metrics used in the evaluation of diagnosis tools. These are defined in Table 3 with results for the best evolved GP rule. For comparison, values are given for a Neural Network rule developed in Elliot [9], and for a manual dental screener.

| Metric | Description | Definition | Performance Ratings | | |
|---|---|---|---|---|---|
| | | | GP | NN | Manual |
| Sensitivity | Performance of a test in terms of its ability to accurately predict a positive outcome, given that the outcome was actually positive. | $\frac{TP}{TP+FN}$ | 73% | 64% | 74% |
| Specificity | Performance of a test in terms of its ability to accurately predict a negative outcome given that the outcome was actually negative | $\frac{TN}{TN+TP}$ | 65% | 68% | 99% |
| Positive Predictive Value | Performance of the test in terms of percentage of negative outcomes | $\frac{TP}{TP+FP}$ | 15% | 15% | 67% |
| Negative Predictive Value | Performance of a test in terms of a percentage of negative outcomes | $\frac{TN}{TN+FN}$ | 96% | 95% | 99% |

Table 3: Table performance metrics for best evolved diagnostic rule

These results show that although the evolved rule does not perform as well as a manual screener, it does perform very similarly to a neural network implementation of a diagnosis tool.

Although the evolved rule could not act as a substitute for a dental screener, it could act as a suitable pre-filter which could be embedded in patient management software.

# 5 Control theory

The limitations of classical control theory and the need for more robust controller designs under large amount of uncertainty, has led control theorists to pursue their research into newer designs, which come under the name "intelligent control". Intelligent controllers can be classified as either "pure" or hybrid architectures [7]. Neural, evolutionary and fuzzy approaches are used or integrated to build systems with improved performance [38].

Genetic programming offers an ideal candidate for controller designs, due to the direct matching of its individual structures and control rules. One particular problem in the application of GP to control problems is the consideration of stability issues. There are two ways to deal with the stability of controllers of this type:

- try to prove the stability of a derived controller after the end of a GP run
- somehow try to incorporate the stability behaviour of a controller in the fitness measure.

So far, GP has been applied to a small number of control problems but without major consideration of stability issues (with the exception of [5]). Some of the successful GP control applications include the solution to classical control theory problems, the broom balancing problem and the truck backer upper problem [20]. However, there is a current "explosion" of GP application to control problems (for an overview of evolutionary algorithms for control applications see [7]). Here one of the most successful applications of GP to control theory problems is presented.

## 5.1 The Attitude Control Problem

Attitude control is the process of achieving and maintaining an orientation in space by performing attitude manoeuvres (an attitude manoeuvres is the process of re-orienting a satellite or spacecraft from one attitude to another). The attitude of the body is relative to some external reference frame. This reference frame may be either inertially fixed, or slowly rotating, as in the case of Earth-oriented satellites.

GP can be utilised with the main aim of the discovery of control laws which are able to detumble a satellite (reduce its kinetic energy). The second part of the attitude control problem (achieving a desired orientation) is not addressed here. It is assumed that the satellite is a rigid body, so that the motion is described by the Euler equations [11]. The system is equipped with reaction thrusters which provide control torques about the three principal axes. The attitude control problem has no complete general solution [29], it is highly nonlinear, exhibits chaotic behaviour under certain circumstances [27] and therefore is a challenging problem for testing the suitability of GP for the design of complex plant controllers.

## 5.2 The Euler equations

The Euler equations describe the rotational motion of a rigid body about its mass centre. They are given by the following equations [11]:

$$\begin{aligned} L &= I_x\dot{\omega_1} - (I_y - I_z)\omega_2\omega_3 \\ M &= I_y\dot{\omega_2} - (I_z - I_x)\omega_3\omega_1 \\ N &= I_z\dot{\omega_3} - (I_x - I_y)\omega_1\omega_2 \end{aligned} \tag{1}$$

where $I_x, I_y, I_z$ are principal moments of inertia, $\omega_1, \omega_2, \omega_3$ are the angular velocities of the rigid body about the $x, y, z$ body axes and $L, M, N$ are the torques about the $x, y, z$ axes respectively. There are several cases where freely rotating rigid bodies can exhibit chaotic behaviour [6].

If the torque feedback matrix for the nonlinear system is denoted by $\mathbf{A}$, so that $\mathbf{G} = (L, M, N) = \mathbf{A}\boldsymbol{\omega}$, the equations (1) become

$$\begin{aligned} (\mathbf{A}\boldsymbol{\omega})_1 &= I_x\dot{\omega_1} - (I_y - I_z)\omega_2\omega_3 \\ (\mathbf{A}\boldsymbol{\omega})_2 &= I_y\dot{\omega_2} - (I_z - I_x)\omega_3\omega_1 \\ (\mathbf{A}\boldsymbol{\omega})_3 &= I_z\dot{\omega_3} - (I_x - I_y)\omega_1\omega_2 \end{aligned} \tag{2}$$

It has been observed [27, 32] that for certain choices of $I_x, I_y, I_z$ and $\mathbf{A}$ equations (2) exhibit both strange attractors and limit cycles, therefore the high-nonlinear dynamic system is very complex (can exhibit different modes of motion).

## 5.3 The GP approach

Plain GP was applied for the detumbling of a satellite with moments of inertia $I_x = 23000$, $I_y = 23300$ and $I_z = 24000$. The goal was to discover a control law which detumbles the satellite about the $x, y, z$ body axes, i.e. $\omega_1 = \omega_2 = \omega_3 = 0$.

The fitness function used in the GP run was:

$$F = \omega_1^2 + \omega_2^2 + \omega_3^2 \tag{3}$$

with the following terminal and function sets:

Terminal set: $T = (\omega, \dot{\omega}, I, R, e)$

Function set: $F = (\sin, \cos, +, -, *, /, pow)$

where $R$ is the random number generator, $/$ is protected division as described earlier in Section 2.2, and *pow* is the power function taking as arguments the base and the exponent. The $*$ operator is the standard multiplication function when applied between two numbers, or a number and a vector, but is defined as an element-to-element multiplication when both of its arguments are vectors. In this initial approach, the angular velocity $\boldsymbol{\omega}$, the angular acceleration $\dot{\boldsymbol{\omega}}$ and the moments of inertia $\mathbf{I}$ were treated as vectors. If an acceptable solution was not found, then the two vectors would be expanded so that the terminal set would contain the individual elements of $\boldsymbol{\omega}$, $\dot{\boldsymbol{\omega}}$ and $\mathbf{I}$, i.e. $\omega_1, \omega_2, \omega_3, \dot{\omega_1}, \dot{\omega_2}, \dot{\omega_3}, I_x, I_y, I_z$.

The parameters shown in Table 4 were used for the results mentioned in this section:

| | |
|---|---|
| population size | 1000 |
| crossover probability | 0.90 |
| reproduction probability | 0.10 |
| mutation probability | 0.00 |
| probability of choosing function node for crossover | 0.90 |
| maximum depth of individuals in initial population | 6 |
| maximum depth of individuals in evolved population | 17 |
| generative method | ramped half-and-half |
| selection method | fitness proportionate |

Table 4: GP Parameters for the Satellite Detumbling control problem.

In one of the runs the following solution was found in generation 61:

$$\mathbf{G} = -1.2 \cdot \mathbf{I} \cdot \boldsymbol{\omega} \tag{4}$$

According to the definition of the $*$ operation above, the solution (4) can be expanded to the following:

$$\begin{aligned} L &= -1.2 \cdot I_x \omega_1 \\ M &= -1.2 \cdot I_y \omega_2 \\ N &= -1.2 \cdot I_z \omega_3 \end{aligned} \tag{5}$$

The control law (5) was applied to the satellite (using computer simulations) and the results are shown in Figures 3, 4 and 5.

In [5] it was proved that for the control law given by (5) a Lyapunov function exists, therefore the stability of the system is guaranteed from theory. This was the first (and the only so far) proof of stability for a controller derived by the genetic programming approach. Further research is required for the incorporation of the stability criteria in the fitness function, i.e. the fitness function could be constructed to serve as a Lyapunov function.

# 6 Speeding up GP

Whilst GP has been shown as able to solve difficult problems, an unfortunate characteristic is the considerable computation which is sometimes required by the process. The maintenance of the population, and the simulation of the evolutionary process is usually a very small proportion of the total run time. The area where most execution time is spent is in the evaluation of the fitness of the individuals in the population. Execution time can be particularly long when a complex simulation is required. This must, of course be carried out for every individual in the population. Difficult problems, with computationally expensive simulations inevitably demand larger populations, thus the problem is exacerbated. Whilst the ever increasing performance of modern CPUs will make more and more problems amenable to GP, in the immediate term, parallelisation of the technique offers the most effective means by which GP can be made to produce more timely results.

The idea of parallelising GP is by no means new. The GA community have recognised some time ago that parallel computing could be used to improve performance in their field. It is a natural progression to apply similar techniques to GP. A good example of applying parallel

GP to a complex problem is provided by Bennet III *et al*[2]. In this approach GP was used to evolve analogue electronic circuits. To evaluate the fitness of the circuits, a patched version of the freely available SPICE (Simulation program with Integrated Circuit Emphasis) simulator was used [33]. The run was on a system consisting of 64, 80MHz PowerPC 601 processors the architecture of which is described in Andre and Koza [1]. Although it is reported that this is an efficient means of parallelising GP, the equipment required is not going to be freely available to most researchers. Another parallel GP implementation is reported by Juille and Pollack [17].

Those seeking to enter the field of GP with a view to evaluating its suitability for use with their specific application need a simple means of producing a parallel GP system which will work on a network of entry level workstations. An obvious restriction to this method of parallelisation is the relatively slow communication afforded by a typical Ethernet network. The Bulk Synchronous Parallel (BSP) [37] model of parallel computing offers an easy entry to parallel computing, and does not require great skill in parallel computing. It is a SIMD (Single Instruction Multiple Data) parallel model which is easily implemented through a library such as that developed by Miller and Reed [30]. The library contains only six operations as shown in Table 5.

| | |
|---|---|
| `BSP_START` | start of the BSP program |
| `BSP_FINISH` | end of the BSP program |
| `BSP_SSTEP_START(n)` | start of superstep `n` |
| `BSP_SSTEP_END(n)` | end of superstep `n` |
| `BSP_STORE(to, from_data, to_data, length)` | store from local to remote processor |
| `BSP_FETCH(from, from_data, to_data, length)` | fetch from remote to local processor |

Table 5: The Oxford BSP Library basic operations.

Using BSP, a custom written GP application has been adapted to allow it to run in a parallel mode. Different versions of the library allow the same code to communicate using shared memory on a single machine, or communicate over a network using TCP/IP. Further versions of the library allow researchers to extend their work to a massively parallel machine such as a Cray if they so wish. However, the emphasis for this work was for the implementation on a network of workstations. A particularly important requirement for this was that communication should be kept to a minimum, as this was likely to be where a bottleneck would occur.

In order to minimise communication a loosely coupled approach was adopted, whereby the individual processes in the BSP machine are able to proceed independently, with communication occurring periodically. It was noted in Section 2.5 that the use of demes, or sub-populations have been shown to be beneficial for maintaining genetic diversity in GP. It is logical to locate a sub-population or deme at each processing node. In this way each processing node can be considered to be an island. This island model has been previously adopted in the GA field [12] and in parallel GP [23]. The extreme case of loosely coupled system would involve the simultaneous execution of a number of differently seeded runs as completely independent processes [15], but this does not take advantage of the use of demes.

The standard GP process is modified by the addition of a migration operator as shown in Figure 6.

The processing islands are arranged according to a topology, examples of which are shown in Figure 7. Every 10 generations, the top 10% of individuals from each island were migrated to neighbouring islands consequently displacing the lowest performing individuals, thereby distributing the best genetic material throughout the global population.

### 6.1 The Artificial Ant Problem

The problem chosen to test the BSP GP implementation was the Artificial Ant problem as used by Koza[20]. The problem involves moving a robot ant along a trail of food which lies on a grid. This trail contains 157 pieces of food as shown in Figure 8. In this figure, a black square represents a piece of food, and a grey square represents a gap in the trail. The figure shows the top left corner of the full grid which is 100 × 100 squares. The tableau for this problem is shown in Table 6.

| | |
|---|---|
| Objective: | Evolve a program to guide a robot ant along a trail of 'food'. |
| Terminal set: | `LEFT, RIGHT, FORWARD` |
| Function set: | `IF_FOOD_AHEAD, BRANCH2, BRANCH3` |
| Fitness: | Number of pieces of food collected before a timeout of 3,000 moves is made. |
| Fitness Case: | Single case — a grid containing a trail of food |
| Parameters: | population = 2000, generations = 50 |

Table 6: Tableau for Robot Ant Problem

The ant is capable of a simple set of actions, these being:

- Move forward one grid square
- Turn Left $90^{\mathrm{O}}$
- Turn Right $90^{\mathrm{O}}$

The ant also has a limited decision making ability through a function:

- If food in square immediately ahead then action $a$ else action $b$

The function set is augmented by the addition of two unconditional branching functions. This allows sequences of ant actions to appear in the evolved programs. The 3,000 move timeout is to prevent the ant finding all the food on the trail simply by randomly wandering over all 10,000 grid squares.

### 6.2 Results

A perfect individual can be evolved using GP, however this was already known, and the purpose of these experiments was to investigate the speedup achievable on a simple problem.

The workstations used to run the software were a number of SUN SPARCstation 5 machines with 70MHz microSPARC-II processors, and running SOLARIS 2.4, each with 32Mb of RAM. These machines were connected on a common subnet with ethernet cabling. Communication between processors was achieved using TCP/IP.

The two implementations were run for 50 generations, and the mean elapsed time was recorded. The results are shown in Table 7.

The recorded speedups are shown in Figure 9.

| processors | elapsed time/s | |
|---|---|---|
| | ring | star |
| 1 | 4980 | 5100 |
| 2 | 5280 | 5400 |
| 4 | 6180 | 6120 |
| 8 | 7020 | 6430 |

Table 7: Elapsed time for runs of 50 generations for parallel GP implementations

Although the size of the problem was not very large, it can be seen that a significant speedup of the GP run is achieved. The differences between the star and the ring topologies are negligible. As the size of the problem increases, it can be expected that further speedups will be seen as the delays resulting from communication will be smaller compared with the actual parallel execution time. It is therefore beneficial to implement a simple parallel implementation of GP within the BSP model. By making use of demes or sub-populations, the diversity of the population is maintained during the run. This island model approach has been shown to be a better method of GP parallelisation than trying to parallelise a single GP population, as the latter method may suffer from premature convergence problems of standard GP, and the increased communication requirements do not yield such great speedups. A similar parallel approach using PVM was implemented in [8], tested on the truck backer upper problem.

# 7 Conclusions

The number of applications in the genetic programming area are now quite substantial, and new journals appear which specifically deal with evolutionary computation, presenting more and more GP applications in a number of disciplines. In this work, GP has been summarised and some of its more important facets described. Two applications in prediction and control theory have been presented which demonstrate the capabilities of GP in these very significant tasks. The idea of parallel GP has been demonstrated, as it is felt that this will be of particular importance to the future of GP.

At the moment industrial applications of GP seem quite limited. This is attributed to the youthfulness of the field rather than to its limitations. It is strongly believed that the field of genetic programming will expand its industrial applications significantly, similarly to the exponential growth of the number of GP publications. For this, further research is needed to overcome the problems addressed here and to build upon the strengths of GP.

# Acknowledgements

We are grateful to the Eastman Dental Institute for providing the data for the oral cancer diagnosis project. This work was partly supported by EPSRC award number 95700741.

# References

[1] David Andre and John R. Koza. Parallel genetic programming: A scalable implementation using the transputer network architecture. In Peter J. Angeline and K. E. Kinnear,

Jr., editors, *Advances in Genetic Programming 2*, chapter 16, pages 317–338. MIT Press, Cambridge, MA, USA, 1996.

[2] Forrest H Bennett III, John R. Koza, David Andre, and Martin A. Keane. Evolution of a 60 decibel op amp using genetic programming. In *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96)*, Lecture Notes in Computer Science, Berlin, Germany, 1996. Springer-Verlag.

[3] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Communications Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastraße 3, 8092 Zurich, Switzerland, December 1995.

[4] Charles Darwin. *On the Origin of Species.* John Murray, London, 1859.

[5] Dimitris C. Dracopoulos. Evolutionary control of a satellite. In John R. Koza, Deb. Kalyanmoy, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, San Francisco, CA, July 13–16 1997. Morgan Kaufmann.

[6] Dimitris C. Dracopoulos. *Evolutionary Learning Algorithms for Neural Adaptive Control.* Springer Verlag, August 1997.

[7] Dimitris C. Dracopoulos. Genetic algorithms and genetic programming for control. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 329–344. Springer Verlag, 1997.

[8] Dimitris C. Dracopoulos and Duncan Self. Parallel genetic programming. In *Proceedings of UK Parallel 96.* Springer Verlag, 1996.

[9] Ciaran Elliot. The use of inductive logic programming and data mining techniques to identify people at risk of oral cancer and precancer. Master's thesis, Brunel University, 1996.

[10] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1989.

[11] Herbert Goldstein. *Classical Mechanics.* Addison Wesley, second edition, 1980.

[12] V. Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, San Francisco, CA, USA, 1993. Morgan Kaufman.

[13] Simon G. Handley. The prediction of the degree of exposure to solvent of amino acid residues via genetic programming. In *Second International Conference on Intelligent Systems for Molecular Biology*, Stanford University, Stanford, CA, USA, 1994. AAAI Press.

[14] Simon G. Handley. Classifying nucleic acid sub-sequences as introns or exons using genetic programming. In Christopher Rawlins, Dominic Clark, Russ Altman, Lawrence Hunter, Thomas Lengauer, and Shoshana Wodak, editors, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology (ISMB-95)*, pages 162–169, Cambridge, UK, 1995. AAAI Press.

[15] Christopher Harris and Bernard Buxton. GP-COM: A distributed, component-based genetic programming system in C++. Research Note RN/96/2, UCL, Gower Street, London, WC1E 6BT, UK, January 1996.

[16] T. Higuchi, M. Iwata, and L. Weixin, editors. *Evolvable Systems: From Biology to Hardware*. Springer Verlag, 1997.

[17] Hugues Juille and Jordan B. Pollack. Parallel genetic programming and fine-grained SIMD architecture. In E. S. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 31–37, MIT, Cambridge, MA, USA, 10–12November 1995. AAAI.

[18] J.A. Jullien, M.C Downer, J. Zakzrewska, and P.M. Speight. Evaluation of a screening test for the early detection of oral cancer and pre-cancer. *Communications of Dental Health*, 12:3, 1995.

[19] John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, volume 1, pages 768–774, San Mateo, CA, USA, 20-25August 1989. Morgan Kaufman.

[20] John R. Koza. *Genetic Programming: on the Programming of Computers by means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[21] John R. Koza. *Genetic Programming II*. MIT Press, Cambridge, MA, USA, 1994.

[22] John R. Koza. Future work and practical applications of genetic programming. In T. Baeck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages H1.1–1–6. Oxford University Press, 1997.

[23] John R. Koza and David Andre. Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Stanford University, Department of Computer Science, January 1995.

[24] John R. Koza and David Andre. Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 8, pages 155–176. MIT Press, Cambridge, MA, USA, 1996.

[25] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 123–131, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[26] John R. Koza, Forrest H Bennett III, Jeffrey L. Hutchings, Stephen L. Bade, Martin A. Keane, and David Andre. Rapidly reconfigurable field-programmable gate arrays for accelerating fitness evaluation in genetic programming. In John R. Koza, editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, pages 121–131, Stanford University, CA, USA, 13–16 July 1997. Stanford Bookstore.

[27] R. B. Leipnik and T. A. Newton. Double strange attractors in rigid body motion with linear feedback control. *Physics Letters*, 86A:63–67, 1981.

[28] Weixin Liu, Masahiro Murakawa, and Tetsuya Higuchi. Evolvable hardware for on-line adaptive traffic control in ATM networks. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 504–509, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[29] George Meyer. On the use of Euler's theorem on rotations for the synthesis of attitude control systems. Technical Report TN D-3643, NASA, 1966.

[30] Richard Miller and Joy Reed. The Oxford BSP Library users' guide. Technical report, University of Oxford, 1993.

[31] David J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, March 1994.

[32] George E. Piper and Harry G. Kwatny. Complicated dynamics in spacecraft attitude control systems. *Journal of Guidance, Control and Dynamics*, 15(4):825–831, July-August 1992.

[33] Thomas Quarles, A.R. Newton, D.O. Pederson, and A Sangiovanni-Vincentelli. *SPICE 3 Version 3F5 User's Manual.* Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA, March 1994.

[34] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press/Bradford Books, 1998.

[35] Gilbert Syswerd. Uniform crossover in genetic algorithm. In J.D.Schaffer, editor, *Proceedings of the third international conference on genetic algorithms.* Morgan Kaufmann, 1989.

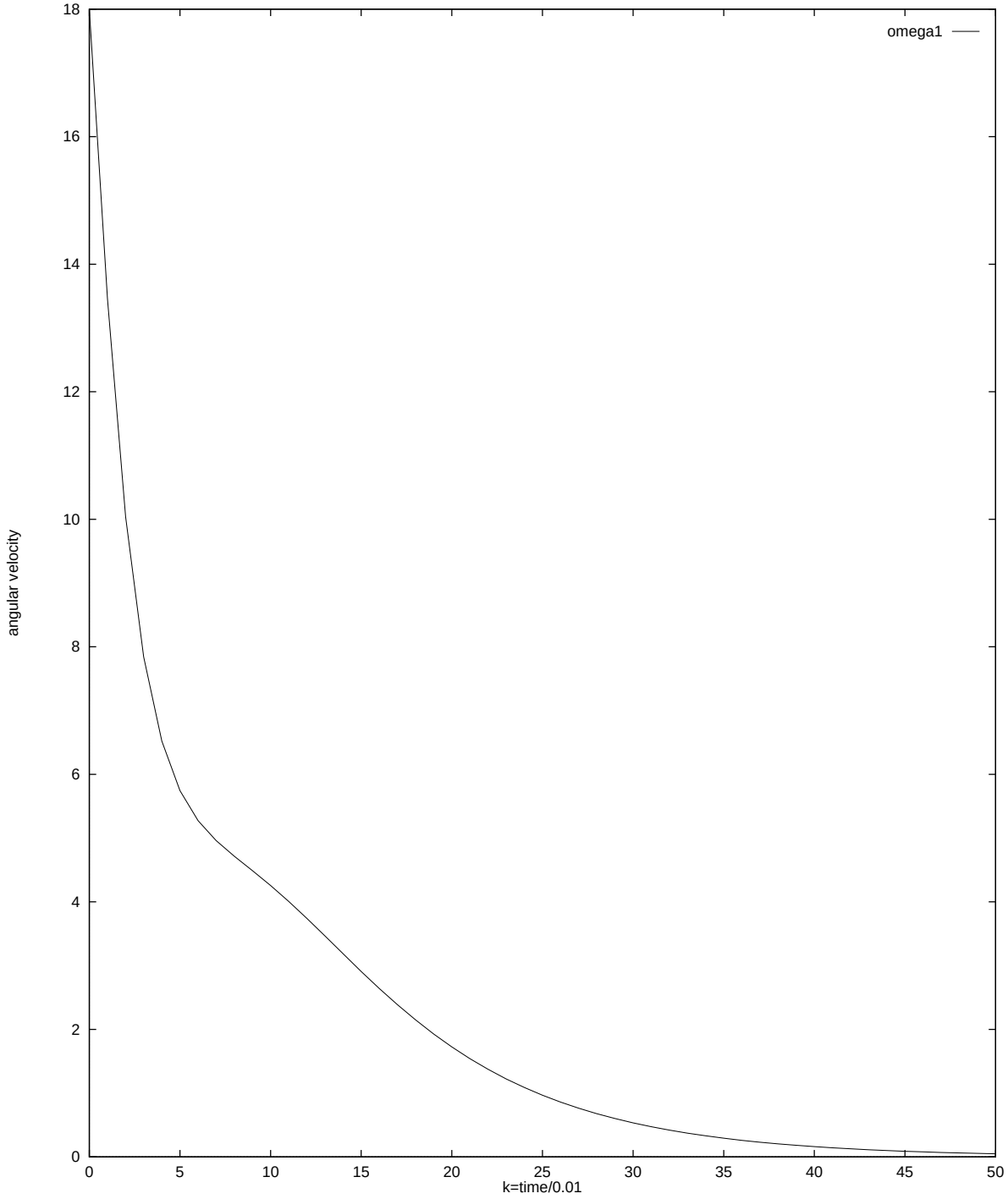
[36] Adrian Thompson. Silicon evolution. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 444–452, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[37] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the Association for Computing Machinery*, 33(8):103–111, 1990.

[38] David A. White and Donald A. Sofge, editors. *Handbook of Intelligent Control.* Van Nostrand Reinhold, 1992.

[39] Darrell Whitle. The genitor algorithm and selection pressure: Why rank based allocation of reproductive trials is best. In J.D.Schaffer, editor, *Proceedings of the third international conference on genetic algorithms.* Morgan Kaufmann, 1989.

[40] Jing Xiao, Zbigniew Michalewicz, Lixin Zhang, and Trojanowski Krzysztof. Adaptive evolutionary planner/navigator for mobile robots. *TEC*, 1(1):18–28, April 1997.




(a)

Figure 3: The control law discovered by GP applied to the detumbling of the satellite. The satellite moments of inertia are $I_x = 23000, I_y = 23300$ and $I_z = 24000$ and the initial conditions $(\omega_1, \omega_2, \omega_3) = (18.0, 9.0, 9.4)$. angular velocity $\omega_1$

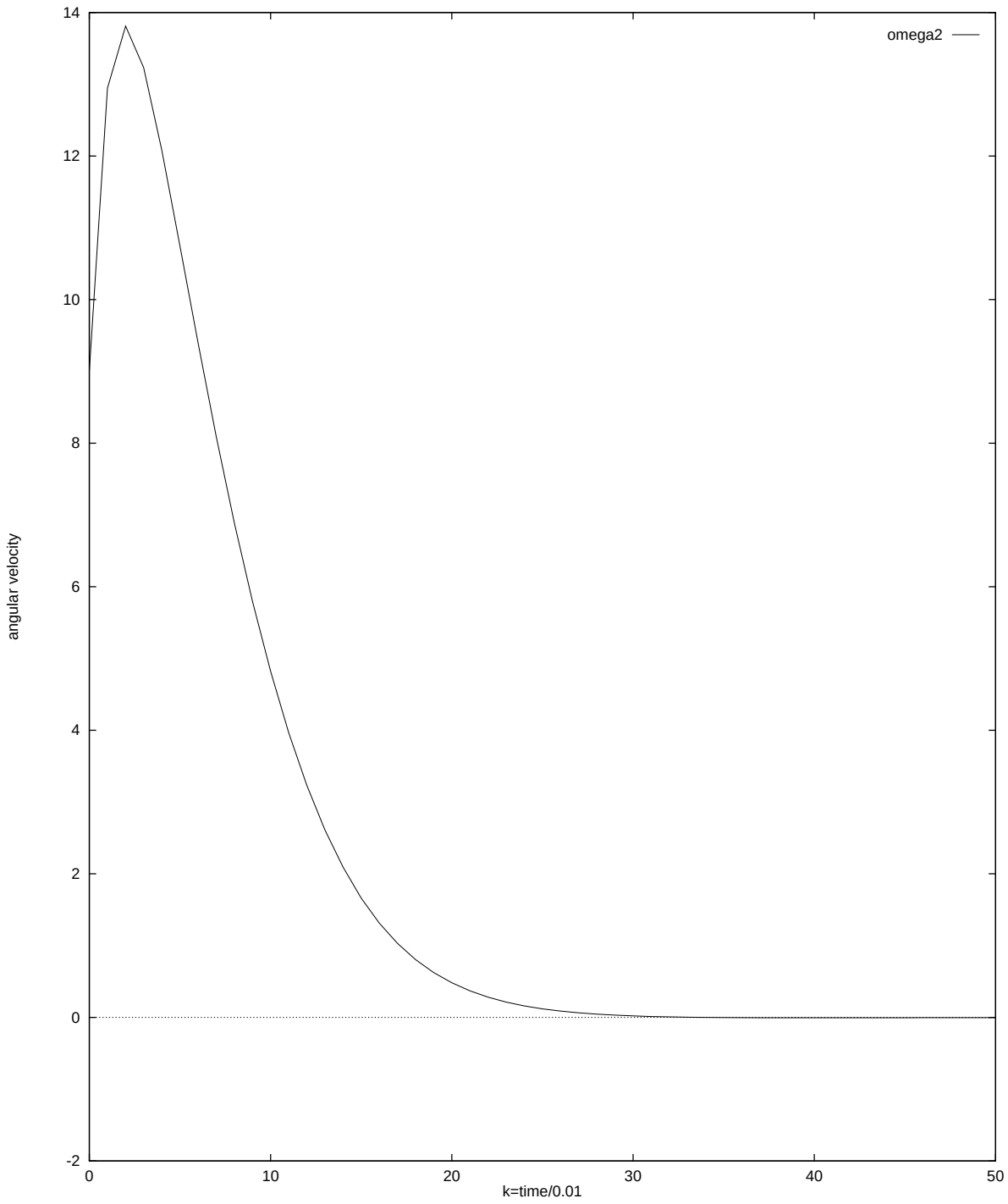



Figure 4: The control law discovered by GP applied to the detumbling of the satellite. The satellite moments of inertia are $I_x = 23000, I_y = 23300$ and $I_z = 24000$ and the initial conditions $(\omega_1, \omega_2, \omega_3) = (18.0, 9.0, 9.4)$. angular velocity $\omega_2$.

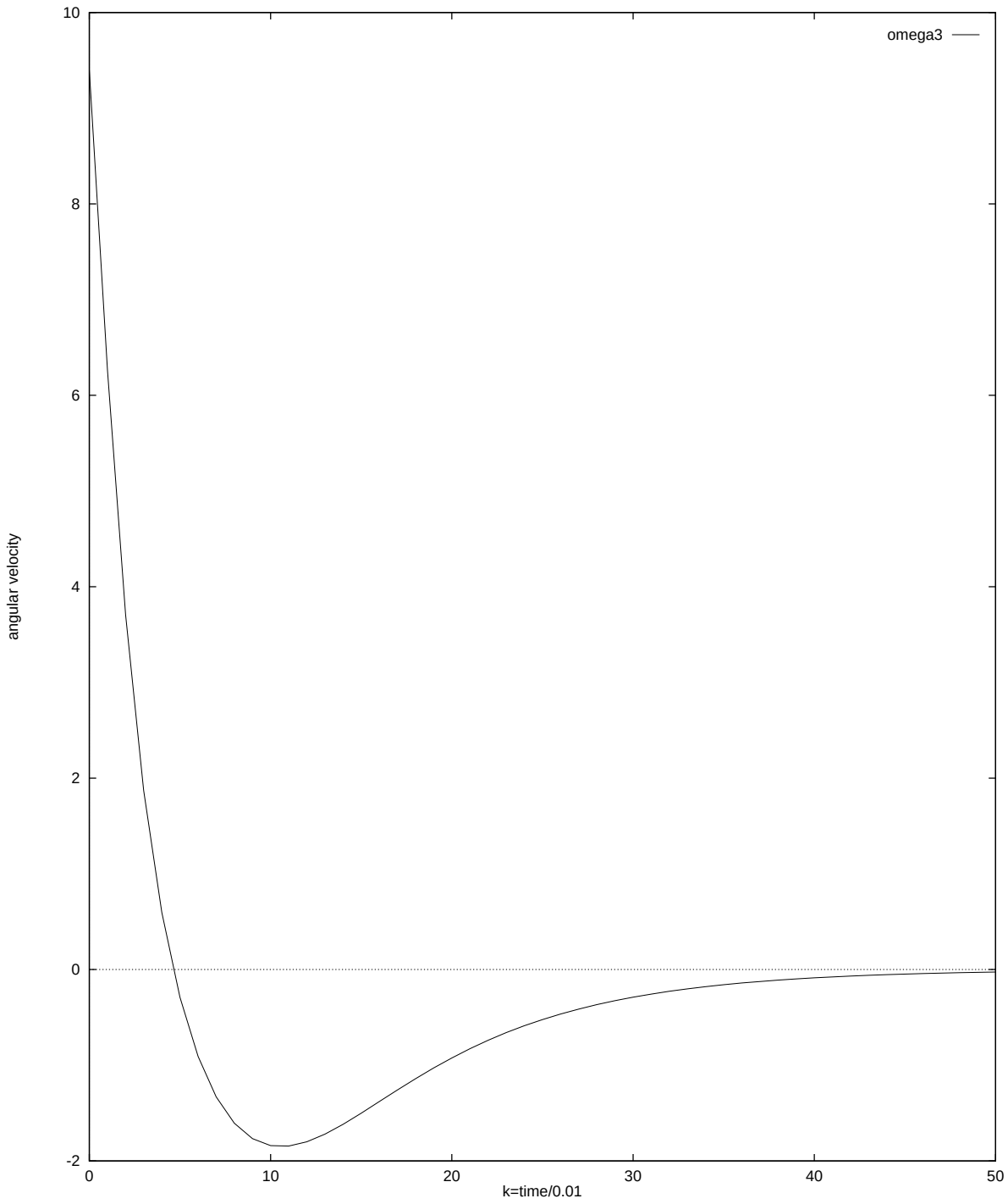



Figure 5: The control law discovered by GP applied to the detumbling of the satellite. The satellite moments of inertia are $I_x = 23000, I_y = 23300$ and $I_z = 24000$ and the initial conditions $(\omega_1, \omega_2, \omega_3) = (18.0, 9.0, 9.4)$. angular velocity $\omega_3$.

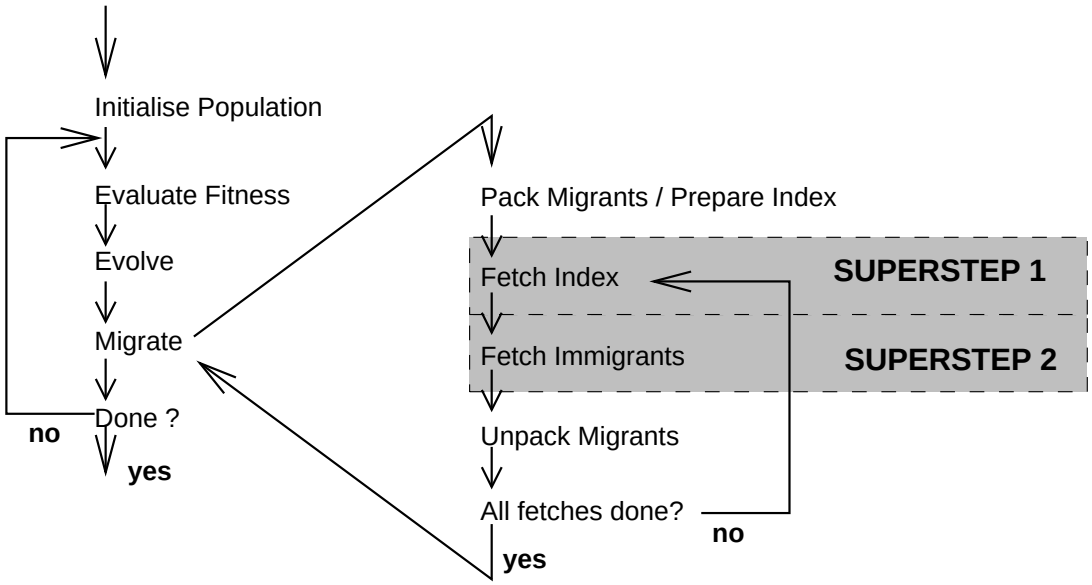



Figure 6: Parallel GP process

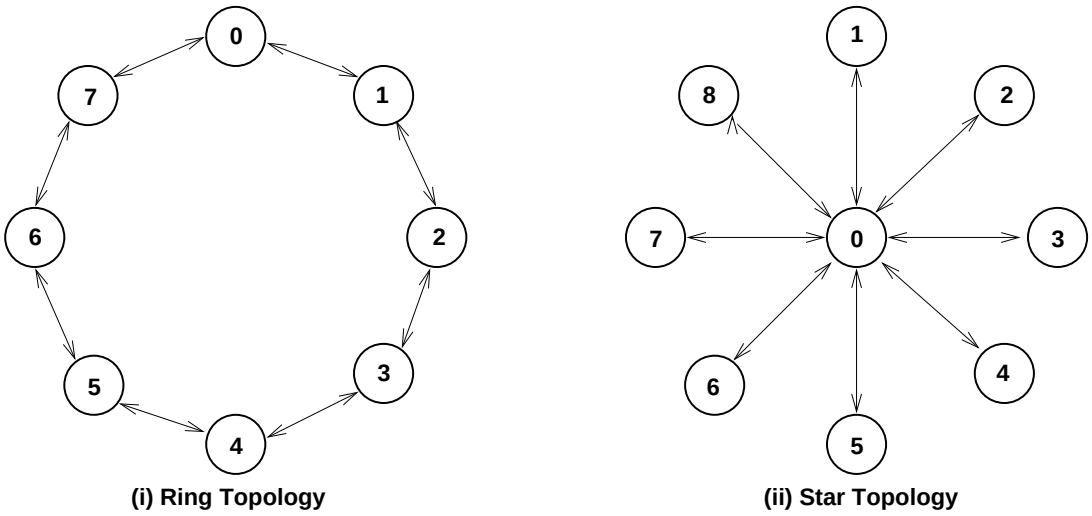



Figure 7: Topologies used with the island model GP implementation.

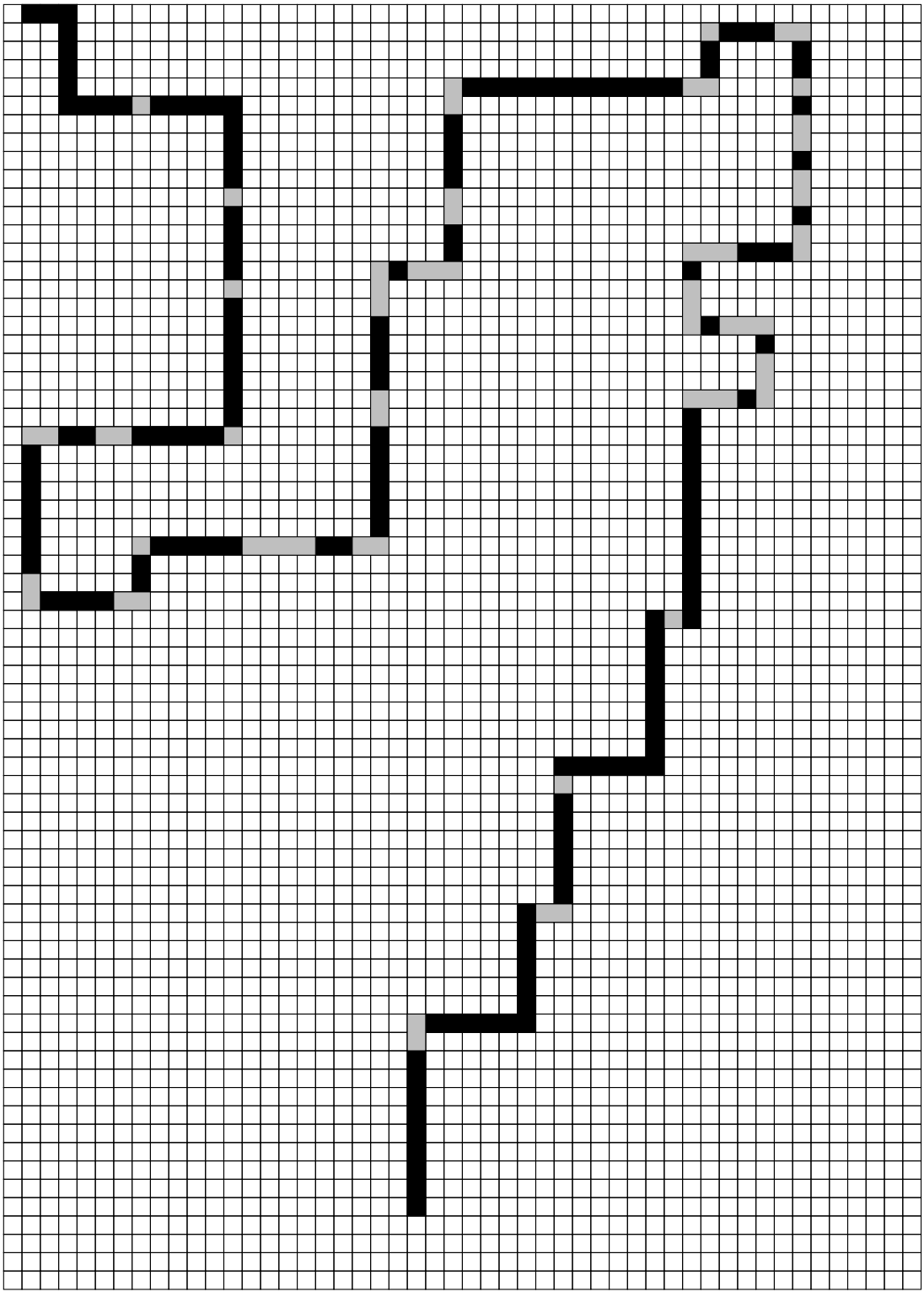


Figure 8: The Artificial Ant trail.

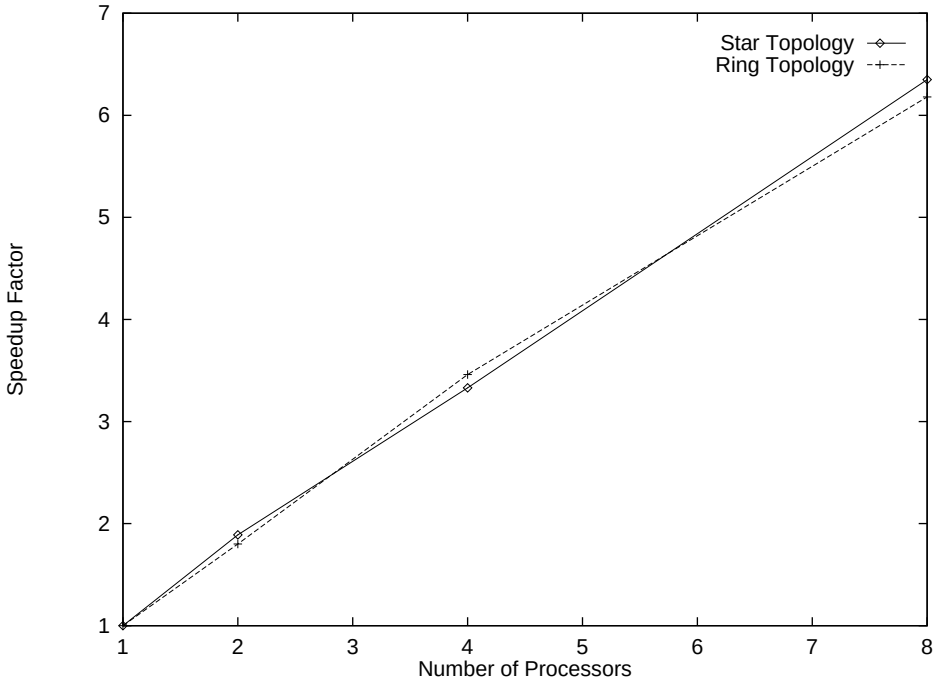



Figure 9: Graph of actual speedup achieved