GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms Final Report

Project title: GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms

Document title: GA-MINER: Parallel Data Mining with Hierarchical Genetic Algorithms Final Report

Document identifier: EPCC-AIKMS-GA-MINER-REPORT 1.0

Document history:

Personnel	Date	Summary	Version
Ian W Flockhart	17th July 1995	First draft	0.1
Ian W Flockhart	20th October 1995	Included NJR sections	0.2
Ian W Flockhart	16th November 1995	Approved	1.0

Approval List: *Project Manager*
Professor Nicholas J Radcliffe

Distribution List: *AIKMS, EPSRC, Industrial Collaborators*
Dr Mark Birkin (GMAP Ltd), Phil Brown (Barclays Bank Plc),
Professor Simon Lavington (AIKMS Technical Coordinator), Catherine Barnes (EPSRC)

Abstract

Many organisations now routinely gather vast and ever-increasing amounts of data in the ordinary course of their business. While much of this information is collected for day-to-day operational reasons, many businesses are now realising that this data has much additional value for improving operational processes. Large databases can form the basis of decision support systems, often based around a *data warehouse*. Such systems may then be used for a variety of applications such as trend spotting, pattern recognition, behavioral modeling and customer worth assessment.

Against this backdrop, the term *data mining* is used to refer to the process of searching through a large volume of data to discover interesting and useful information. The authors have traditionally sought to divide data mining into three types or levels—*undirected or pure data mining*, where the system is left almost entirely unconstrained to discover patterns in the data free of prejudices from the user; *directed data mining*, where the user may specify some constraint to "steer" the system through its search; and *hypothesis testing and refinement*, where the user poses some hypothesis and the system first evaluates the hypothesis and then seeks to refine it.

The system specified and built during this project—GA-MINER—is unique in being applicable to all three levels of data mining. Although undirected data mining has been the defining goal, the system has been deliberately built to enable also directed data mining and hypothesis refinement. The project has succeeded in its initial goal to implement a parallel genetic-algorithm based data mining tool for rule discovery and has maintained the principle that discovered knowledge must be scrutable and understandable. Performance results have shown that the system is scalable on both Symmetric Multi-processor (SMP) and Massively Parallel Multi-processor (MPP) systems.

Fast and flexible development of GA-MINER was made possible by its implementation in the *Reproductive Plan Language* (RPL2), an extensible language, interpreter and run-time system for the implementation of stochastic search algorithms, with a special emphasis on evolutionary algorithms such as genetic algorithms.

A central component of GA-MINER is the *pattern template*, which defines the general form of the patterns of "interest" to the user and restricts the search space to those patterns which are consistent with this form. A prototype graphical user interface has also been implemented to allow users to manipulate the pattern template and to visualise the results of the search. One of the biggest challenges in this project has been to produce a system that can differentiate between interesting and uninteresting patterns. The project considered several evaluation functions for estimating pattern interest, mainly based on statistical measures, and our experience has shown that many of these evaluation mechanisms give highly similar results. While falling short of providing fully satisfactory definitions of interesting patterns, the evaluation functions have been sufficiently successful for the system to discover several useful patterns in the databases provided by our industrial collaborators, GMAP Ltd. and Barclay Bank Plc.

Many of the insights, ideas and results of GA-MINER are also being exploited by Quadstone Ltd in its Decisionhouse product. The commercial market for data warehousing and data mining tools is expanding rapidly and increasing competition is forcing all businesses to put their customers at the centre of their operation. They are therefore collecting and trying to exploit data that they can obtain on these customers. Although hardware and database software now supports very large stores of data, there is no software currently available to analyse large data volumes efficiently, or to perform effective data mining operations in acceptable timescales. GA-MINER has provided many of the underlying architectural and analytical insights that have enabled the imminent product release of Decisionhouse, which aims to fulfill this need for scalable data mining software.

Contents

Chapter 1

Motivation and Objectives

1.1 Business Trends

The globalisation of business, together with increasing levels of competition, is combining with the falling cost and increasing power of information technology (IT) to change fundamentally the way that businesses operate. On the one hand, pressures for cost reduction, rationalisation and greater consistency have led companies to replace many human functions with computer-based processes, both behind the scenes and at the point of customer contact. These changes have been particularly pronounced in the financial services and insurance sectors, together with retailing and telecommunications, and have brought many benefits to the companies that have embraced them. On the other hand, businesses are increasingly finding that despite the masses of data they collect, process and (sometimes) store, they seem ever more remote from their customers, This has two primary effects. The first is that gross mistakes are often made because computers operate only those sanity checks explicitly programmed into them, and will often fail to "notice" bizarre conditions that no human would be likely to ignore. The second is that marketing and customer worth assessment become much harder, because the business "knows less" about its customers.

The central plank of IT relevant to customer relationship management is, of course, the database. Database systems are now central to running most customer-facing businesses, and are typically seen to support several classes of core functions:

- transaction processing
- batch processing
- *ad-hoc* querying
- Management Information System (MIS) and Executive Information System (EIS) functionality.

Transaction and batch processing are the critical functions that "run the business", whereas the *ad hoc* querying and MIS/EIS functions simply give primitive access to the data and provide management with the information it needs in summary form.

Over recent years, many businesses have started to realise that databases have the potential, in principle, to re-establish a closer link with their customers. *Data warehouses* have been conceived and are being built to collect together large volumes of business data—often at some level of aggregation above transaction level—with the aim of fulfilling a *decision support* function. A *decision support system* (DSS) is supposed to help businesses to turn large (and by this token necessarily incomprehensible) volumes of data into much smaller quantities of high-quality business information that can be understood by human beings. Decision support systems can offer this functionality through a wide variety of techniques and applications, including

- visualisation of and visual navigation through large data volumes, typically presented hierarchically

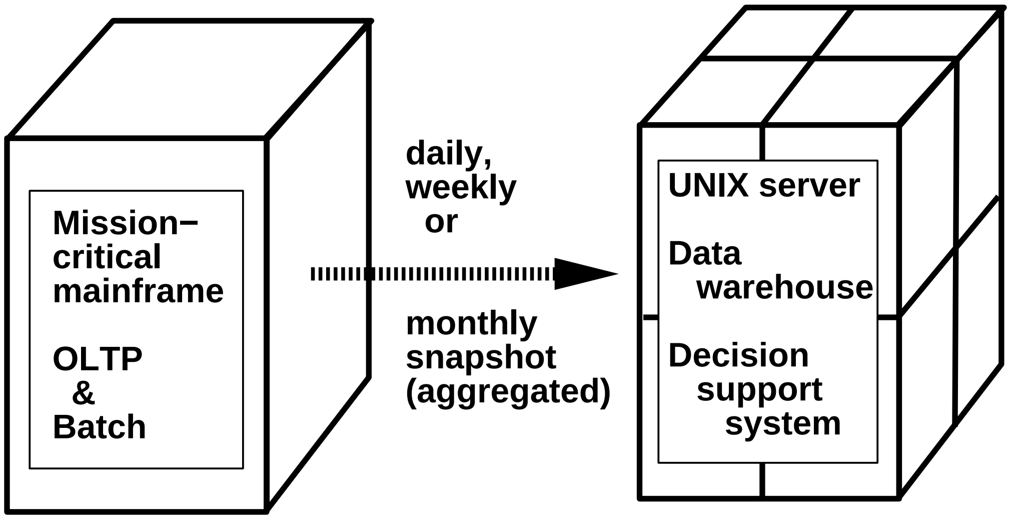


Figure 1.1: Decision support systems are typically high-end Unix servers, taking periodic snapshots from business-critical mainframes.

- trend spotting
- outlier detection
- cluster analysis
- induction, pattern recognition and classification
- behavioural modelling
- customer worth modelling.

Against this backdrop, the term *data mining*—precise definitions of which will be discussed in section 1.3—is used to refer to the process of searching through a large volume of data, typically initially resident in a database, to discover interesting, useful, information. This is the context in which the current project was conceived, though its focus is much narrower than some current uses of the term ‘data mining’ might suggest.

1.2 Trends in Business Computing

Business-critical function such as transaction and batch processing are still, today, typically run on traditional mainframes in most (large) businesses. The machines on people’s desks are typically PC’s or (less often) dumb terminals. However, the companies investing in decision support systems usually build these on very different platforms from mainframes. A typical situation is illustrated in figure 1.1 A typical data warehouse will be will be built on a high-end Unix server, typically a shared-memory architecture with 8–16 nodes. Examples of such platforms include the Sun Sparccentre 2000, SGI Challenge, Sequent Symmetry, and Cray CS6400. Slightly less commonly, a distributed-memory Unix platform such as an AT&T (“Teradata” / NCR) 3600, a Meiko Computing Surface or an IBM SP2 will be used.

A striking characteristic of these servers is that they are almost invariably parallel, and in many cases the databases used on them—typically Oracle, Informix, Sybase or DB2—actually exploit this parallelism. (Indeed, it could be argued that databases are the most powerful and complex examples of parallel applications in mainstream use.)

The traditional distinctions between parallel architectures were between “single-instruction-stream, single-data-stream ” (SIMD) machines, which consist of large numbers of simple processors executing a common intruction stream in lockstep (typically 1,024–65,536) and “multiple-instruction-stream, multiple-data-stream" (MIMD) machines which have smaller numbers of more powerful processors (typically 2–1,024), executing (potentially) different programs. However, except for specialised applications such as image processing, MIMD architectures have established near-complete domination. The main distinction today is between shared- and distributed-memory architectures. Here, however, boundaries are blurring fast.

Traditional distributed-memory platforms, now usually called MPP ("massively parallel processing") or "shared nothing" architectures, used message-passing programming paradigms to communicate data along physical links between processors. The memory of the entire machine was therefore partitioned into distinct address spaces. In contrast, traditional shared-memory architectures (now more commonly called "symmetric multiprocessors", or even merely "high-end servers") used a single address bus and offered a programming model based around a single address space. Mechanisms such as semaphores were used to control write access when necessary (though shared-memory machines, even more than their distributed-memory counterparts, have been used primarily as throughput engines for sequential applications). The distinctions are blurring fast as:

1. Kendall Square Research first introduced "virtual shared memory" machines, which offered a shared-memory programming model on distributed hardware.

2. Other manufacturers have quietly followed suit, so that today even the Cray T3D, one of the most scalable distributed-memory machines in existence, uses a single address space offers direct read and write facilities to locations on other processors.

3. As memory sizes increase beyond the 32-bit limit (4Gb theoretical, 1–2Gb in practice), 32-bit *shared*-memory architectures no longer offer a single address space. While this will change as 64-bit processors become more common, note that even the DEC Alpha—sold as a fully 64-bit processor—has fewer than 64 physical address pins.

4. Some manufacturers are now discussing supporting shared-memory programming models over distributed networks of workstations or servers (as opposed to integrated parallel machines).

At one level, therefore, architectural trends are converging. Nevertheless, building a scalable parallel application remains an extremely subtle business, requiring considerable attention to data locality, caching, contention and so forth. Even more critically, the demands of databases and related data-intensive applications such as decision support systems, are very different from those of the traditional scientific and technical applications that spurred the development of the early parallel architectures. In the context of commercial applications, it tends to be memory bandwidth and I/O bandwidth that limit performance more than raw processing power (in the sense of flops or MIPS). It is these pressures that are arguably driving some of the more radical architectural developments today.

1.3 Data Mining

Since this project was proposed, the term "data mining" has become a key marketing "buzz word" in many software and related markets, including those of databases, data warehousing, marketing, machine learning, pattern recognition and image processing. As suppliers have rushed to attach the label to many existing products and processes, the term has inevitably become somewhat debased, so it is important to revisit what we have meant by the term.

The authors have traditionally sought to divide data mining into three types or levels as follows:

- *Undirected or Pure Data Mining.*
  With undirected data mining, the concept is that the user simply asks of the data miner: "Tell me something interesting about my data". The key point here is that the user is *not* specifying what kind of rule is desired. The importance of this is that the system is left completely unconstrained (at least by the user) and is therefore given the greatest "freedom" to discover patterns in the data free of prejudices from the user. It seems likely that in these circumstances there is the greatest

scope for finding completely unexpected patterns in the data, which has been one of the "promises" of data mining.

In fact, arguably the user could be a little more specific and still leave enormous freedom to the data miner, by specifying, for example, a field of interest. Here the (conceptual) challenge might be: "Tell me something about my female customers".

- *Directed data mining.*
  The user may ask something much more specific, such as: "Tell me about links between income and expenditure" or "Characterise my highest-spending customers". Clearly here a much stronger "steer" is being given to the system. In this case, the problem usually changes from a general pattern-detection problem to a rather better defined *induction* problem. (Induction is most naturally thought of as the problem of reasoning or inference under partial information, so that *deduction* is not possible.) Induction problems have been extremely well studied using a variety of statistical and machine-learning techniques including various forms of regression, linear (or "additive") scorecarding, a range of decision tree methods, classification neural networks (typically "feed-forward" networks, also known as "multi-layer perceptrons) and various forms of cluster analysis and nearest-neighbour methods.

- *Hypothesis testing and refinement.*
  The final form of data mining we have traditionally considered involves the user conceptually saying: "I think that there is a positive correlation between sales of peaches and sales of cream: am I right?". Now the idea is that the system first evaluates the hypothesis but then—if the evidence for it is not strong—seeks to refine it. Depending on what scope for variation the system is allowed, this may make the task even more directed than "directed data mining", as described above, or almost as open as "undirected data mining", if all parts of the original hypothesis are allowed to be varied.

The system—GA-MINER—that we have specified and built is unusual in being applicable to all three kinds of data mining. It is undirected data mining that has been our defining goal, but we have deliberately built a system which allows directed data mining and hypothesis refinement also to be tackled. Directed data mining is achieved by fixing certain parts of the pattern over the course of the run, and hypothesis refinement is achieved by "seeding" the system with the hypothesis but then allowing some or all parts of it to vary.

It should be emphasized that the mainstream use of the term data mining is slightly different from ours. Most systems that we are aware of are really induction engines, tackling what we would call "directed data mining". There is a community that has existed for a while specialising in what it has called "Knowledge Discovery in Databases (KDD)", and indeed there has been a series of workshops attached to IJCAI on this theme, and a mailing list (KDD-nuggets) exists for this community. In 1995, the authors registered for the workshop "Knowledge Discovery in Databases (KDD)", but by the time it was held, KDD had been redefined to stand for "Knowledge Discovery and Data Mining (KDD)". The standard usage of the terms adopted at that conference is that "knowledge discovery" is the complete process of finding patterns in databases, corresponding roughly to the union of our three levels of "data mining", while the term "data mining" is used for the *search* component of a larger process that involves data collation, extraction, cleansing, pre-processing, post-processing, presentation and comprehension.

The authors suspect that the KDD terminology will gain wider popularity, but that the term "data mining" itself will for some time be used to mean very different things to different people.

In the remainder of this report, we will use data mining in the sense of the three levels introduced above, usually with a focus on "undirected data mining", but specialising where appropriate to the other forms.

1.4 What Makes a Pattern "Interesting"?

We have so far discussed data mining in terms of the search for interesting patterns in databases, but have as yet given little regard for what makes a pattern interesting.

In a highly interactive data mining system, users may have extensive control over the search process, guiding it towards patterns which they themselves regard as interesting. However, as the level of automation in a data mining system increases, there is an ever more important need for some numeric measure of the relative goodness of patterns. Before proceeding further, we must first attempt to quantify the subjective concept of "interesting" in order that "good" patterns may be distinguished from the "bad" automatically during the search.

We may identify some features as being common to interesting patterns. First and foremost, an interesting pattern must summarise some correlation which exists in the data. This is not to say that the pattern must be entirely accurate, but we must be able to calculate some numeric measure of the degree to which it is accurate. General patterns, which apply to larger portions of the database are, other things being equal, preferred, though generality and accuracy are often in conflict. The pattern should also avoid expressing trends that can be explained by chance variation and fluctuations in the data, and should therefore include some concept of *statistical significance*.

Patterns must capture non-trivial correlations, not simply "truisms". For example, the fact that all male customers in a database have the title "Mr" may well be entirely correct, but cannot be regarded as interesting or useful. Defining "truisms" can in itself be problematic. We could regard them as statements expressing *functional dependencies* about the database, known collections of fields whose values functionally determine the values in another field, or the term could also be widened to include known correlations between variables. Fortunately, we may bypass the provision of a definition for "truisms", as they represent just one example of a more general problem, which is easier to define, though unfortunately, more difficult to solve. That is, how a data mining system handles patterns already known to exist in the database, and uses them not only for the purpose of avoiding re-discovery or re-presentation of known relationships, but also as domain knowledge within an ever-growing knowledge-base which can be used to assist in the derivation of new patterns. This problem is further compounded in databases which may change over time, rendering at least some patterns out-of-date.

Patterns must be approachable, in the sense that they be formulated and presented in a form easily digested and understood by humans. It may be be acceptable to simplify a pattern, thereby reducing its accuracy and significance, if such simplification results in more readable and understandable knowledge. This *scrutabilty* of patterns is particularly important if data mining systems are to be accepted by the business community. Also important, is configurability towards a particular application domain or user, since the same database may be mined by many different people, all of whom have an interest in very different patterns.

From the discussion here, it is clear that the definition of "interesting" represents an immensely difficult task and indeed most of the considerations outlined above are the subject of ongoing research (e.g. Silberschatz & Tuzhilin, 1995). We have yet to resolve the problem of defining "interesting" to our satisfaction, but pragmatism requires that we construct at least some measure of this quantity. In general, GA-MINER restricts itself to measures of statistical significance, combined with heuristics for pattern simplification. The details connected with the evaluation a pattern's goodness are presented in section 4.4 along with the various pattern definitions supported by the system.

1.5 Scrutability of Patterns

For a data mining system to be usable by businesses, it must present its findings in a form which is understandable by non-scientists. Some techniques such as neural networks can produce clusterings of the database which implicitly represent useful knowledge. However, this knowledge is not immediately understandable by a typical end user of a decision support system and as such, it will receive little interest from business users.

Perhaps the most direct and understandable form in which a pattern may be expressed is as an explicit rule, perhaps expressed as predicates (*if x then y*,). The rule need not be strictly correct in order for it to be useful and indeed in general this is a stronger requirement than one would wish to impose. Picking out trends and correlations that are true to some degree is the more typical aim, because data is generally noisy in the true sense of containing errors, and more importantly because correlations do not have to hold perfectly in order to constitute useful, exploitable information. Thus the predicate *if x then y*, which translates formally to *x implies y*, could be more usefully replaced in the present context by *x tends to imply y*, or even *x increases the likelihood of y* (which could itself be formalised as $\mathcal{P}(y|x) > \mathcal{P}(y)$, where $\mathcal{P}$ denotes probability). Similar rule patterns are found in a variety of data mining systems. e.g. EXPLORA (Hoschka & Kloesgen, 1991; Kloesgen, 1992, 1994a; Gerbhardt, 1994), the Knowledge Discovery Workbench (Piatetsky-Shapiro, 1992) and Data Surveyor (Holsheimer & Kersten, 1994) (see section 3).

Other pattern forms are also of interest. Some systems, for example, Forty-Niner (Zytkow & Baker, 1991; Zytkow & Zembowicz, 1993) present results in the form of cross-tabulations or contingency tables, while EXPLORA provides a whole range of patterns for use in a variety of knowledge discovery tasks. Although the original goal of GA-MINER was to develop an explicit rule-generating data mining system using genetic algorithms, we have gone beyond this to consider other pattern forms when they have been helpful in maintaining the property of scrutability.

1.6 Ease of Use

Given the nature of a general purpose data mining system, we may reasonably expect that users of the system will come from a variety of disparate backgrounds, will use the system for a diverse range analysis tasks and will be equipped with varying levels of technical expertise. The importance of an appropriate user interface to such a system is clear.

A general purpose system must be approachable to all these users. The system should therefore be configurable in nature, both to particular application domains and to particular users. An appropriate graphical user interface should be provided, giving control over the nature and extent of the search. More advanced features of the system, for example, the ability to modify the search method, should be restricted to expert menus.

Ideally, the system should also interface cleanly with a variety of standard database management systems, allowing users to access data from systems with which they are familiar. It is unlikely that data mining systems reliant upon specific database hardware platforms will be widely accepted in the business community, partly because of commitments to previously installed systems, but also because of the increasing trend towards open systems. However, current database management systems may well adapt to incorporate the facilities required for the efficient support of data mining software.

Clearly, the presentation of patterns to users is of vital importance. Graphical presentations are likely to be easier to understand, and are particularly useful when they can be incorporated into reports. As data mining systems become more sophisticated, facilities for automatic report generation will be required, although such reports will undoubtedly still require some level of manual editing to reconcile the system's and user's respective views of what is interesting. Some ideas for pattern presentation are discussed by Klösgen (Kloesgen, 1992).

1.7 Objectives

The objective of GA-MINER is to investigate the utility of genetic algorithms as a software architecture for large scale parallel data mining. The emphasis is largely on the less-directed forms of data mining (see section 1.3) and on the search for scrutable rules which are directly comprehensible. The work also aims to consider any implications of the software model, or indeed of the data mining task itself, on the underlying hardware architecture required for its efficient support.

The detailed objectives of GA-MINER project may be summarised as follows:

- Determine a suitable representation for patterns. The representation must be capable of expressing sufficiently interesting knowledge about the database, while retaining enough simplicity to ensure scrutability. The emphasis is on patterns which express easily assimilable knowledge. The representation is also of vital importance given the selection of genetic algorithms as the pattern search method, since it will define the heritable units which may usefully be propagated through the genetic operators.

- Construct a parallel genetic-based data mining system, capable of finding patterns within real-world data. This requires the design and implementation of suitable recombination and mutation operators, as well as operators for collecting sets of rules, and for rule simplification.
- Investigate the relative merits of a variety of measures of goodness for patterns, with a view to formulating the notion of "interestingness" for a rule.
- Investigate the scalability of the genetic data mining system on both distributed memory and shared memory hardware platforms, and td to determine any aspects of the particular genetic algorithm method chosen which may have an impact on the performance or architectural design of database systems.

1.8 Review of Progress

All of the original stated objectives of the project have been met, with the construction of a parallel genetic data mining system to discover probabilistic rule patterns in large databases. The system has been demonstrated to be scalable (see section 5.3) on both shared-memory and massively parallel distributed memory platforms. Furthermore, it has been possible to engage in development of the system beyond that originally envisaged, with the inclusion of several pattern forms in addition to the main explicit rule form, and a prototype graphical user interface has been constructed.

Clearly, there is still much useful research to be done, many ideas for which are detailed in section 7.2.

Chapter 2

Background

Before considering GA-MINER in depth, we first provide some background on the chosen approach and methods used by the system. Section 2.1 gives an introduction to stochastic search and optimisation, followed in section 2.2 by an overview of evolutionary algorithms. We then briefly introduce the *Reproductive Plan Language* (section 2.3), an extensible language, interpreter and run-time system for the implementation of stochastic search algorithms, which has been used for the implementation of GA-MINER. Finally, in section 2.4 we consider issues concerned with database interfacing, including sampling, preprocessing and a description of the databases used during the course of this research.

2.1 Background on Stochastic Search and Optimisation

An optimisation problem is one in which the goal is to find the best possible way to perform some task or configure some system. In the present case the goal is to find the best possible rule. In considering such an optimisation problem, it is helpful to think in terms of a *search space*, which is the set of all the possible *solutions* to the problem at hand, where by a 'solution' we mean any *possible* rule, not just the best one, i.e. a solution is *any* way of performing the task at hand.

In order to perform an optimisation, it is necessary to have a way of measuring the quality of any possible solution from the search space. This is achieved by employing an *objective function* (also known as an *evaluation function*), which, when presented with a solution assigns to it a numerical value which in some way reflects its quality. In the present case, the objective function is the rule quality.

The goal of the optimisation is to find the solution in the search space that has the highest—or sometimes the lowest—value of the objective function. If more than one solution has the same maximum value for this, the goal is usually to find one of these solutions, though occasionally it is to find all of them. A solution having the maximum value of the objective function across the whole search space is called a *global optimum* (figure 2.1).

It should be noted that unless the form of the objective function is particularly simple, or the search space is very small, it is generally impossible to construct optimisation techniques that are guaranteed to find a global optimum. This is because, in general, the only way to be certain that a particular solution is the best possible is to generate and evaluate every possible solution to the problem at hand—a process called *exhaustive search* or *enumeration*. Bearing in mind that the universe has only, on most estimates, been in existence for some 10^{19} seconds, even if a computer could search a collosal one trillion solutions a second (10^{12}), and had been processing since the start of time, it would not have made a noticeable dent in many problems, in which search spaces regularly exceed 10^{100}. Typical problems in the present context, assuming a resolution of a mere 10^6 possible values per clause, have search spaces of order 10. For this reason, most optimisation problems are tackled using non-guaranteed *search* techniques, which explore the space of possible solutions but do not guarantee to find a global optimum. Good search techniques, neverthe-

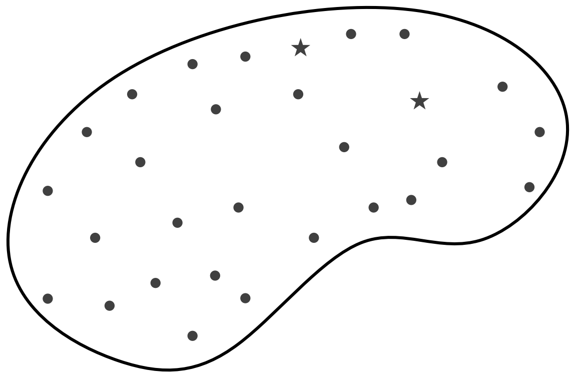


Figure 2.1: The *search space* consists of all potential "solutions" to the problem. In general, there will be one or more *optimal* way (shown here as stars), which maximises the value of the objective function and represents the best way to "solve" the problem.

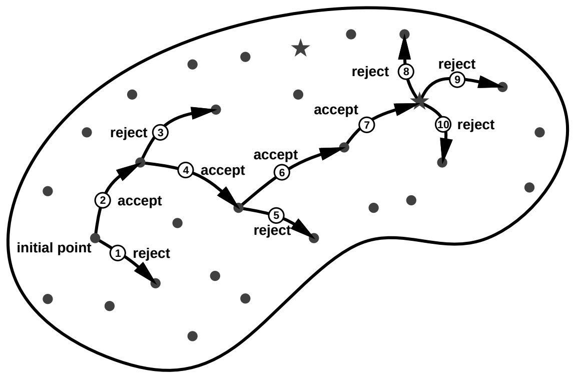


Figure 2.2: The diagram illustrates a search beginning at the initial point shown and repeatedly trying out various moves, accepting some (i.e. moving the "current point") but rejecting others.

less, normally produce near optimal solutions relatively quickly, and in many cases do, in practice, locate a global optimum.

Most search techniques begin by choosing one point (solution) from the search space as a starting point. The choice of this point may be either random or informed. (If non-random, it may be the currently best known solution, or a solution generated by some other method.) Thereafter, the process typically involves maintaining a notion of a *current point,* which is initially set to the starting point. The search normally proceeds by repeatedly generating a trial *move* within the search space by modifying the current point (solution) through the application of a *move operator,* which modifies the current solution in some way. The quality of the new point generated is then measured using the objective function and a decision is then made about whether to replace the current point with the new point. In the simplest case, the "move" to the new point is made if it is better than (or at least as good as) the current point. Such techniques are known generically as *hill climbers.* A general search process of the type discussed in this section is illustrated in figure 2.2.

The selection of a move operator to use for the search effectively imposes a *structure* or *connectivity* on the search space. This is because normally a move operator will be *stochastic,* i.e. it will choose from one of a number of possible moves it could make from the current point. One of two cases will usually arise. In the first case, the move operator will be capable of moving from any point to any other point, but will generate some moves much more often than others. In this case, the move operator effectively induces a *distance*

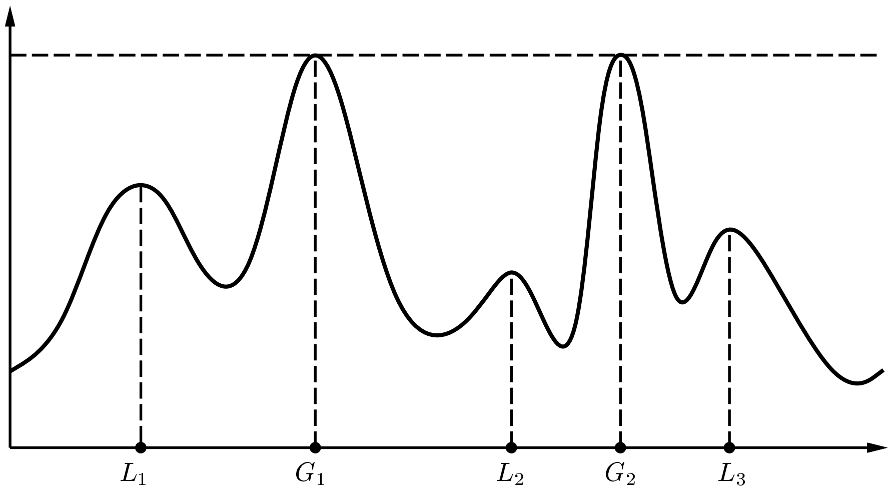


Figure 2.3: The diagram shows a search space with a one-dimensional structure along the horizontal axis and objective function value on the vertical axis. Assuming that the two highest peaks are of equal height, and that the objective function is to be maximised, this search space has two global optima (G_1 and G_2) and three local optima (L_1, L_2 and L_3). (Technically, global optima are also local optima.)

between each pair of points, which is the likelihood of moving from one to the other. Alternatively, the move operator may only move from each point to certain other points. In this case, it effectively defines a *connectivity* over the space, with each solution being connected to those other solutions than can be generated by a move from it. It then becomes meaningful to talk about the *neighbourhood* of a point in the space, which will either be all the other points to which it is connected (and itself), or all the points within some distance ε of itself. A second kind of optimum can now be introduced, known as a *local optimum.* A local optimum is a point in the search space which is better than (or at least as good as) every solution in its neighbourhood (figure 2.3). Most search spaces have many more local optima than global optima.[1] Finding local optima is often relatively easy, whereas finding global optima is very difficult. In real-world optimisation, we often settle for the best local optimum that we can find.

2.2 Evolutionary Algorithms

2.2.1 Motivation

Evolutionary algorithms are a class of stochastic search techniques inspired by natural evolution. Crudely, this inspiration is that, over time, organisms adapt to their environment, becoming *fitter.* Although this improving fitness manifests itself in many ways (as animals run faster and increase their intelligence, and viruses develop ever more effective ways of penetrating their hosts' defences) the ultimate measure of evolutionary fitness is simply the success of an organism in passing on its genes to viable offspring—of reproducing. Darwin famously summarised this guiding principle of evolution as the "survival of the fittest". Clearly the mechanisms of natural evolution constantly demonstrate an impressive "problem-solving" facility, if the problem is defined as producing organisms of increasing fitness. The aim, with evolutionary algorithms, is to simulate key aspects of natural evolving systems on a computer, and in so doing to harness their problem-solving ability to own search and optimisation problems.

[1] Strictly, the local optima are only defined with respect to some particular structure on the space, in this case induced by a particular move operator, as was explained in the main text.

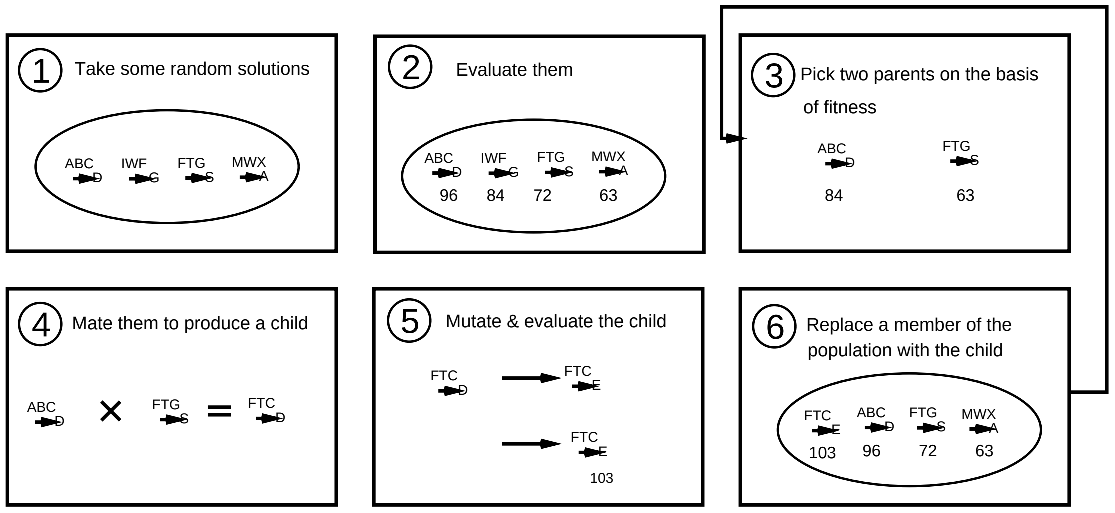

1 Take some random solutions
ABC → D IWF → G FTG → S MWX → A
2 Evaluate them
ABC → D IWF → G FTG → S MWX → A
96 84 72 63
3 Pick two parents on the basis of fitness
ABC → D FTG → S
84 63
4 Mate them to produce a child
ABC → D × FTG → S = FTC → D
5 Mutate & evaluate the child
FTC → D
FTC → E
FTC → E
103
6 Replace a member of the population with the child
FTC → E ABC → D FTG → S MWX → A
103 96 72 63

Figure 2.4: A simple evolutionary algorithm for evolving good rules.

2.2.2 Outline

Although following very broadly the outline of stochastic search given in section 2.1, *evolutionary algorithms* modify the general pattern in a number of important ways. First, instead of using a single *current point,* most evolutionary algorithms maintain a *population* of points from the search space. The size of this population is usually fixed, most often between about 50 and 100,000. The initialisation phase therefore consists not of choosing a single starting solution, but rather a set of such solutions. As in the general case, this choice can either be random or informed.

The second key difference between a general stochastic search algorithm and an evolutionary algorithm concerns the choice of the next point to test. Instead of basing this choice on a single *parent* solution, evolutionary algorithms typically use two parent solutions to determine the next solution generated. Thus, in addition to a *unary* move operator, as discussed above, evolutionary algorithms employ *binary* move operators, known as *recombination* or *crossover* operators, which take a pair of parent solutions and combine them in some way to produce a new child solution (or perhaps more than one child). Recombination is the analogue of sexual reproduction in nature, and the idea is to produce a child which “inherits” some of its properties from one parent solution, and some from the other. In producing a new solution, the normal approach is first to pick a pair of parents for recombination, and after recombination to apply a conventional unary move operator to the child solution they produce before it is evaluated. The unary move operator is normally called a *mutation* operator in the context of evolutionary algorithms, because it is analogous to the random genetic changes that occur in natural evolving systems.

It is important to note that in the standard case, both the recombination and the mutation step are *undirected,* i.e. they do not “try” to produce a child that is better than its parents. Rather it is the process of *selection* that models Darwin’s “survival of the fittest”. There are two opportunities to apply *selection pressure* in evolutionary algorithms. The first is in the choice of parents, which may be biased towards better (“fitter”) members of the population. Alternatively, when children are produced, the choice of which members of the population they should replace may be biased toward the less good members of the population. Some algorithms use both of these devices. While it is not particularly important *where* in the algorithm selection pressure is applied, the *level* of selection pressure is important. If it is too high (because, for example, only the very best members of the population are allowed to reproduce) the search will tend to get stuck quickly in local optima. If, on the other hand, it is too low, the search may be unduly slow, or may even fail to generate solutions of improving quality. A simple evolutionary algorithm is illustrated in figure 2.4.

2.2.3 Representation and Terminology

The choice of representation of solutions can have an important influence on the effectiveness of any search method, and this is nowhere more true than in evolutionary algorithms. This is perhaps unsurprising, because in nature the genetic operators—on which the recombination and mutation operators used in evolutionary algorithms are loosely modelled—operate at a level far removed from the observable characteristics of organisms. Because of the central importance of representation in evolutionary algorithms, it is worth spending a little time examining the issue, and introducing some terminology borrowed from biologists.

Physical organisms are known to biologists as *phenotypes* to distinguish them from their representation in the genetic code on their *genome.* In the current context, a solution itself (a rule) is a *phenotype*, whereas the data structures that we use to *represent* this solution form the *genome.*[2] The information content of the genome is known as the *genotype* of the corresponding organism, though distinction between the genome and the genotype is usually not important.

The reason that the distinction between the solution itself (the phenotype) and its representation (the genotype) is important is that the "genetic" move operators, both recombination and mutation, are most commonly defined as *syntactic* operations on the genotype. In a typical case, the genome consists of a number of variables, called *genes*, each of which takes on a value from a discrete or continuous range.[3] In the simplest cases, recombination consists of taking some gene values from one parent and the rest from the other, while mutation involves altering one or a small number of gene values.

Ultimately, the success of any search method depends on the order in which it generates points in the search space. A good search technique uses information that it gathers during the course of the search to inform its choice of future points sensibly, usually leading to the discovery of ever better solutions, whereas a poor one makes less effective use of the information gathered. In the case of an evolutionary algorithm, the population acts as the memory of the system, containing solutions having co-adapted characteristics. For this reason, it is important when designing move operators for a problem to have regard not only to the *syntactic* changes that they result in at the level of the genotype, but also the *actual* moves that they generate in the search space of solutions. In terms of the ideas introduced in section 2.1, good move operators create appropriate neighbourhoods in the search space, thus making search easier.

2.2.4 Population Structure

In the foregoing discussion, the population has been assumed to be unstructured, in the sense that the likelihood of any pair of solutions mating has been assumed to depend at most on their quality. It is becoming increasingly common, however, to impose a spatial *structure* on populations, so that the likelihood of two solutions mating depends not only on their performance, but also on their locality. The original motivation for this was partly to facilitate the exploitation of parallel and distributed computing hardware, which it does. In many cases, however, it appears that population structure actually improves the effectiveness of search as well, in terms of the number of points visited to achieve a given quality of solution.

There are two principal forms of structured population. In the *island model,* the population consists of a number of distinct *islands.* An evolutionary algorithm runs on each island without reference to the others except that there is occasional *migration* of solutions between islands (figure 2.5). This scheme has a number of attractions, but its most important benefit is that different characteristics tend to emerge, over time, on the different islands. When migration occurs, recombination will sometimes have the effect of combining a characteristic developed on one island with one found on another. In unstructured populations, it can be more difficult for different characteristics to emerge simultaneously as any small difference in relative fitness will normally cause one to out-compete the other.

The second main kind of structured population is known as the *diffusion model,* or the *fine-grained population model* (figure 2.6). Here, solutions occupy the sites of a grid, and mating is restricted to local,

[2]The human genome consists of 26 *chromosomes* (parts), but in evolutionary algorithms there is usually only one chromosome per genome, so terms "genome" and "chromosome" are often used interchangeably.

[3]These values are known as *alleles.*

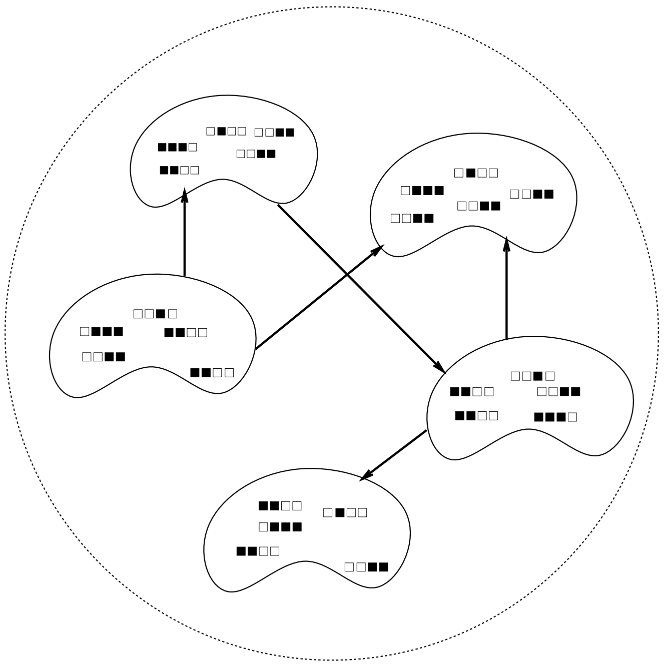


Figure 2.5: In the *island model* a separate evolutionary algorithm runs on each of a number of islands, and solutions occasionally *migrate* between islands, bringing in new genetic material.

overlapping neighbourhoods known as *demes*. As in the island model, the restrictions on mating tend to result in different characteristics evolving on different parts of the grid, but because of the overlapping nature of the demes, information is able to "diffuse" slowly across the grid. Patches of similar solutions tend to emerge, and interesting recombination events tend to occur at the boundaries of these patches.

2.2.5 Handling Constraints

In many cases, including the present case, simple syntactic move operators are inadequate because of *constraints* on valid solutions, which will tend to be violated by merely cutting and splicing existing solutions. There are three principal methods for handling constraints in evolutionary algorithms—building more sophisticated operators that "understand" (ensure compliance with) the constraints, using repair operators to "correct" infeasible child solutions and employing penalty functions. Of these, the last is the crudest, and is normally only adopted when neither of the first two methods can be used.

When repair operators are employed, an interesting decision arises as to whether to use the repaired version of the child only for the purposes of computing the objective function (placing the infeasible child in the population), or to replace the infeasible child with its repaired counterpart. While, on the face of it, it seems more sensible to place the repaired version of the child in the population, there are some circumstances in which this may be unhelpful. Consider the case shown in figure 2.7. Here, a solution is on the edge of a large infeasible region, while the global optimum is on the "other" side. It may well be that any likely move from the solution in question towards the global optimum, the new point will be in the infeasible region, and would be repaired back to the far side from the global optimum. In this case, allowing the genome to represent an infeasible point, while actually evaluating a repaired counterpart solution, may be the best strategy. In practice, following a suggestion of Davis & Orvosh (1993), it is often a good idea to make the choice probabilistically, "reverse transcribing" the repaired phenotype to the genome (say) 5% of the time.

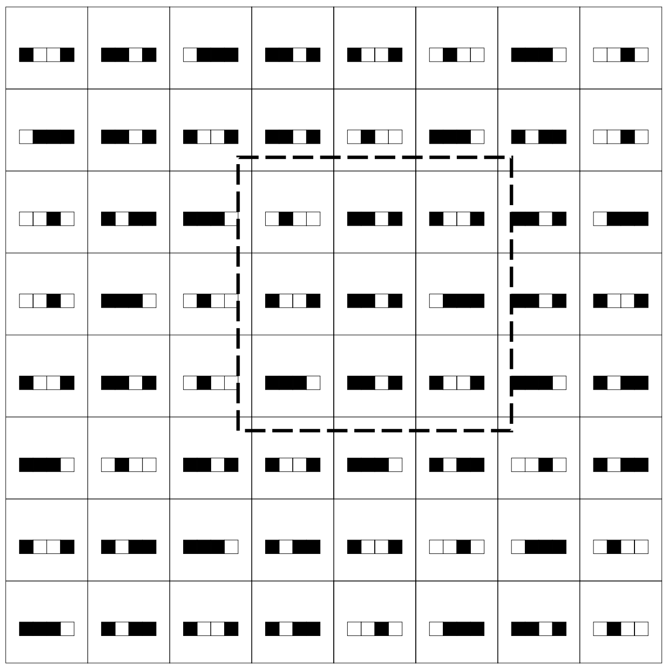


Figure 2.6: In the *diffusion model*, each solution has a unique location on a grid. Mating only occurs within local neighbourhoods (*demes*). As different characteristics emerge on different parts of the grid, the overlapping nature of demes allows information to "diffuse" across the grid, with "interesting" recombination happening at boundaries between the different regions.

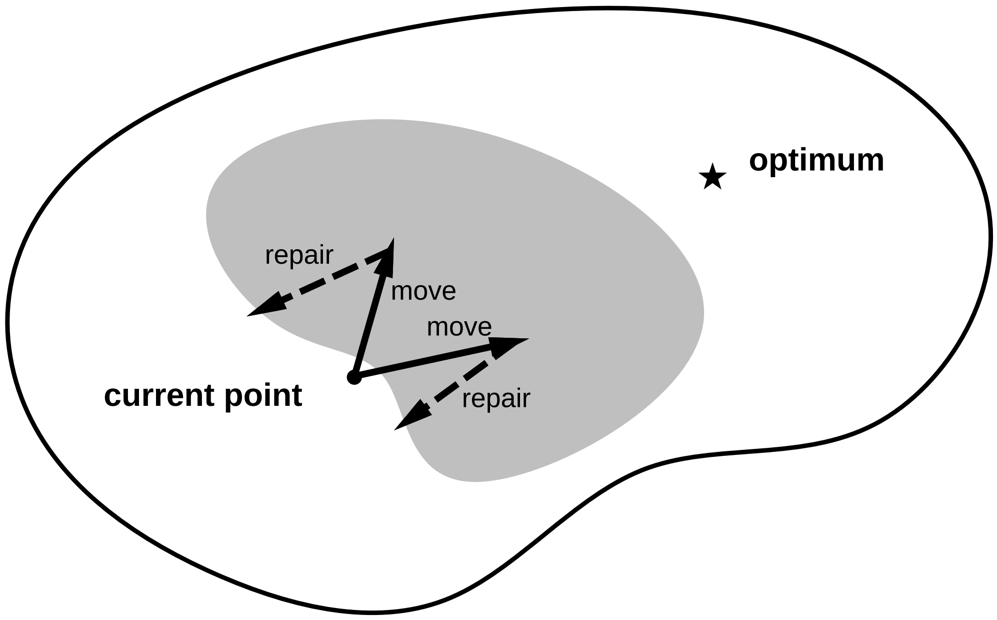


Figure 2.7: The shaded region of the search space is infeasible. If the solution is always repaired, it may be very difficult to traverse the large infeasible region, even though the global optimum is on the "other side". In these circumstances, allowing the genome to remain infeasible, while actually evaluating a phenotype corresponding to a repaired version of it, may be more satisfactory.

2.2.6 Schools of Evolutionary Computing

There are a number of different "schools" of evolutionary computing. The best known of these call their methods *genetic algorithms,* and build on work pioneered by John Holland (1975). In their original form, genetic algorithms tended to be used for combinatorial (discrete) optimisation, and to concentrate on binary string representations, selection pressure applied through the choice of parents and an emphasis on recombination as the main search operator. Over time, however, all of these tendencies have been modified, and many "genetic algorithms" have none of these features.

The second principal school of evolutionary algorithms was developed in Germany by Rechenberg and Schwefel (Baeck & Schwefel, 1993), and is known as the *evolution strategies* community. Evolution strategies have traditionally been applied to continuous numerical optimisation problems, emphasised mutation as the primary search operator, and tended to apply selection pressure through the choice of which offspring were allowed to survive. They also have a large number of internal control parameters, which are themselves subject to adaptation during the course of the run. Like genetic algorithms, over time evolution strategies have broadened, though perhaps less so, and the distinctions between the different schools are now more in their history and traditions than their practices. Nevertheless, the approaches can be distinguished, and this study has investigated both approaches.

Two further schools modes of evolutionary computing are used. *Genetic programming,* (Koza, 1992), and is essentially based on the application of genetic algorithms to LISP parse trees to evolve computer programming. Finally, *Evolutionary programming* (Fogel *et al.*, 1966) was originally based on the idea of evolving finite-state automata to perform tasks, though has evolved to be very similar to evolution strategies.

2.3 RPL2

All the search algorithms implemented for this project were constructed using the *Reproductive Plan Language* RPL2 (Surry & Radcliffe, 1994b, 1994a; Radcliffe & Surry, 1994c). RPL2 is an extensible language, interpreter and run-time system for the implementation of stochastic search algorithms, with a special emphasis on evolutionary algorithms such as genetic algorithms. It was developed at Edinburgh Parallel Computing Centre using a variety funding from SERC, EPSRC, DTI, British Gas and Cray, and has been further developed by Quadstone Ltd, which now owns RPL2.

For details of RPL2, the reader is referred to the papers cited above, together with papers describing its application to a range of challenging optimisation problems, including Boyd *et al.* (1994), Radcliffe & Surry (1994b), Radcliffe & Surry (1994a), Surry *et al.* (1995) and Harding *et al.* (1995). The main features of RPL2 pertinent to the current project are:

- *Automatic Parallelism.* RPL2 interpreters and run-time systems exist in both serial and parallel form, and the RPL2 language contains constructs specifically designed to allow the system to exploit parallelism. The programming paradigm used to support this is message passing, allowing efficient execution on distributed-memory platforms. In most cases, extremely efficient execution on shared-memory platforms is also achieved by parallel implementations of RPL2. RPL2 has been used efficiently on as many as 256 processors on Cray T3D platforms.

- *Support for Arbitrary Representations.* RPL2 supports arbitrary representations, rather than being restricted to simple string-based representations. This is important in the current context as the rule forms used are not string-like.

- *Large library of supplied functions.* RPL2 is supplied with a large number of standard functions which may be used in any reproductive plan ('program'). These massively reduce the amount of work required in constructing a new evolutionary algorithm, and in the present case allowed almost the entire project to be devoted to an exploration of data mining itself, rather than merely coding up ideas.

2.4 Database Interfacing

It is clearly helpful for a full data mining system to include the ability to interface to mainstream databases, for example through SQL. In many cases, the mode of operation of the data miner will be two-phase. The first phase (which may be likened to the training phase of, for example, neural network models) would involve hypothesis formulation and refinement, probably using only a sample of the database (see section 2.4.2). After useful patterns have been generated there would typically be a second phase (analogous to the classification phase of neural and related systems) in which the patterns are tested against the full database or some other larger database with comparable characteristics, either for verification or for application. During the first phase, depending partly on the quantity of data being used, it is likely that “training” data would be extracted *en masse* and then used without further reference to the database. This is typically the phase during which most of the computing power is required and most of the work is done. Even in cases when the entire database is to be used for hypothesis formulation, there are advantages in using extracted tables stored in an efficient format.

GA-MINER currently works on an extract from the database stored in main or virtual memory. This limits the size of database which may be effectively handled by the system to around the size of the main memory available on each processing element used. [4] However, since the large scale parallelism which is achievable by genetic algorithms makes them particularly suitable for larger databases, work is presently underway to integrate GA-MINER into a suite of database interfacing tools developed by Quadstone Ltd.

These tools, designed to run on Symmetric Multi-Processing (SMP) platforms, implement efficient virtual tables by storing an extract from the database in a column-oriented format within memory-mapped files across multiple disks. This will allow GA-MINER to access much larger databases more efficiently and exploit the parallel I/O facilities available on such platforms.

We expect that, at least in the short to medium-term, such extraction and reorganization of data will continue to be required if large scale data analysis is to be efficient supported (see section 6.5).

2.4.1 The Trend towards Data Warehouses

In the previous section we discussed the use of extracts to obtain the data required for analysis purposes. Here we have assumed that all the required data actually exists within the database, and is available for extraction, however this task is in itself far from trivial.

The nature of database systems is changing as a result of fundamentally different business requirements. Inmon & Osterfelt (1991) identify three phases in the adoption of computers within businesses. In the first phase, computers were simply used to automate repetitive and time consuming activities, such as payroll administration, while the second phase saw the introduction of transaction processing systems to automate further the day-to-day running of businesses. The third and current phase, *data pattern processing* involves the large scale pattern search of databases for key insights into the data, essentially a definition for what is now generally known as data mining.

It is clear to see how these changing business needs have influenced database technology. The original first-phase computer databases required relatively few updates by today’s standards, perhaps changing only when new employees joined and old employees left, or when a batch run calculated salaries and tax returns. Second-phase systems on the other hand, typified by the banks’ use of Automated Teller Machines (ATMs), required frequent and fast on-line updates and often much larger storage capabilities. The most recent database systems, known as *data warehouses* and generally used for decision support and data mining, have altered this data access and storage pattern once again.

It is now widely recognised that databases used by organisations to fulfill specific operational needs, can also be used to provide insight and knowledge that may be used for decision support purposes, particularly when many such databases are combined together. For example, a supermarket chain may wish to com-

[4] Performance degrades by a factor of around three when virtual memory is used

bine electronic point of sale (EPOS) data with a demographic breakdown of each of its store's catchment areas, to look for trends in its customer base.

To support such analysis, data warehouses must have the capacity to store vast quantities of data and manage this data in an efficient and cost-effective manner. The sheer scale of the data storage requirements means that a hierarchy of storage mediums is necessary, ranging from fast but expensive memory and data caches, to slower and moderately priced bulk storage devices for infrequently used historical data. The data is often wide ranging and collected from a number of disparate sources, usually requiring translation and compilation into an appropriate format before inclusion in the data warehouse. The system must therefore be able to interface with other widely used database systems. It may also require the ability to trace data back to its original source and time of extraction, perhaps because further analysis is later required on the same data, or because analysts wish to check the integrity of the original data source after some interesting patterns have been discovered. This is particularly difficult when data is being continuously changed and updated. Some systems may also provide improved performance by running in a "read-only" mode, removing the overhead of the lock manager.

Clearly, the emphasis of decision support data warehousing systems is distinctly different from that of systems for on-line transaction processing (OLTP). This has been reflected by the introduction of a new TCP-D benchmark for decision support (TPC-D, 1994). For these reasons, businesses have generally kept their decision support data warehouses separate from their OLTP systems, a trend which we expect to continue, as in the medium term at least, it is unlikely that OLTP systems will be capable of efficiently satisfying the needs of both on-line updates and decision support-style queries. Furthermore, the current trend towards "outsourcing" is likely to result in businesses buying data from external sources rather than collecting it themselves, once again emphasising the need to handle disparate data sources effectively.

2.4.2 The Rôle of Sampling

One of the main problems in mining very large databases is the time cost associated with evaluating the goodness of a pattern. Since evaluation normally involves scanning over all records in the database, an obvious way to reduce this time penalty is to use a sample.

Sampling may be applied in a number of ways, perhaps the most common being *uniform random sampling*, in which each record in the database has the same probability p of being selected as part of the sample. The inference techniques associated with uniform random sampling are relatively straightforward and methods for estimating the degree of significance for a finding are widely known (though see section 6.3 for some cautionary words on their application). Other sampling techniques, such as *stratified random sampling* and *cluster sampling* are also possible, though these are more common during the data acquisition process rather than during the model building stage.

If applied correctly, sampling can significantly reduce the amount of work required to evaluate patterns, and will not affect the quality of the evaluation of discovered patterns to a significant degree. Indeed, sampling may actually result in the construction of a better model for the data, since the use of the full database can result in over-fitting (Cortes *et al.,* 1995).

Despite its potential utility, sampling is now treated with some suspicion by many in the data mining community, and even more by many business users. We believe that the reason for this is that to date, a push to reduce the data volumes used for model building has driven sampling much more than a desire for generalisation ability in the models. This has in many cases led to the use of very small samples, wholly inadequate for the model building tasks to which they have been applied. This is perhaps inevitable, given that many of the tools used today are available only on PCs.

We do not argue here that sampling should not be used, merely that it must be used with caution and that its implications for particular applications are considered in advance.

2.4.3 The Rôle of Pre-Processing

Pre-processing of data can be defined, perhaps rather imprecisely, as the process of preparing data before use in an analysis.

Much of the work which may be regarded as pre-processing can be performed by the database or data warehouse which stores the data, for example, the relational joins, projections and selections needed to build the required table. Pre-processing may also be performed to reorganise the storage of an extracted table for more efficient access by the data mining system.

There are a number of other useful and widely applied pre-processing techniques which may be applied to the data. Summary statistics may be calculated in advance for each field. These may useful as reference values to compare against when searching for outliers or for changes in a distribution, or they may be used to assist the search and discovery process.

Derived attributes traditionally play an important rôle in conventional statistical analysis and can work in combination with the pattern representation to give more expressive knowledge representation. Derived attributes are generally introduced to express quantities which it is believed will have a significant effect on the analysis, but which the method will be unable to introduce on its own. For example, we may add non-linear transformations of the fields in a database if we have a pattern representation which can express only linear relationships between two variables.

Unfortunately, choosing appropriate derived attributes can be a black art. A few systems (e.g. Forty-Niner, Zytkow & Baker, 1991; Zytkow & Zembowicz, 1993) include “equation finders” which attempt to discover functional relationships between fields, reducing the need for derived attributes. Another alternative is to allow the system to create new derived attributes automatically during the search process. Both these approaches, of course, are strongly limited by the kinds of dependencies envisaged by the systems’ creators, and it is hard to see that a strong general methodology for automatic pre-processing can ever be obtained.

Some methods of analysis require particular pre-processing stages to be performed in order for the algorithm to produce good results. For example, transformations on variables are often used to reduce correlations between fields in regression analysis, since these often result in the regression coefficients having high standard errors.

Such transformations may also be useful in data mining. For example, if records contain aggregate data for individuals within particular geographical areas which vary dramatically in population, it may be appropriate to transform the fields to express the figures “per head of population”. This abstracts away from the strong “more people means higher values” relationship, and is likely to make other patterns easier to find.

Attribute elimination is often performed to reduce the number of attributes which need be considered by an analysis or data mining technique, perhaps for the purpose of making the analysis a tractable problem. Usually the attributes are selected for removal on the basis of lack of “predictive power” for the particular analysis taking place, usually measured by a correlation coefficient. However, this is difficult to determine, as an attribute may have no predictive power in isolation, but have significant predictive power when combined with other attributes. Attribute removal must therefore be applied cautiously, and perhaps only when the analysis or data mining technique would suffer badly from the inclusion of too many attributes.

Although data mining systems are performing increasing amounts of data pre-processing automatically, at the time of writing it remains largely a task requiring significant human input.

2.4.4 Data Provided by Industrial Collaborators

Our industrial collaborators both provided data of commercial interest which could be used by GA-MINER for data mining purposes.

GMAP Ltd. provided UK census data from the 1991 census, together with several databases containing car sales data from the same year. These databases were then aggregated and combined into a single relational table, from which some redundant attributes were dropped and to which some additional derived attributes were added.

In more detail, the databases provided were:

- UK census data for 1991. This data consisted of a geo-demographic breakdown of the UK at postal district level and contained aggregate data associated with both individuals and households.
- Ford car sales data, collected from individual garages.
- Sales data for all manufacturers at a postal district level.
- Geographical locations of all major UK car dealers.

The final pre-processed database consisted of 2257 records, one for each postal district, and 54 attributes, including demographic information and fields relating to the sales and market share of Ford cars in each postal district. It was hoped that data mining would provide insight into the factors affecting the sales of Ford vehicles, and discover patterns relating sales or market share to demographic features of particular areas. The fields in this database are detailed in full in appendix D.

During performance testing, this data was replicated some 20 times to form a larger database of 51140 records and 58 fields (see section 5.3), to evaluate GA-MINER's performance on databases on a realistic size.

Barclays Ltd. provided two customer databases, both of which are of a confidential nature and are therefore not discussed in detail in this report. In the first database, which consisted of 8669 records and 62 attributes, the rôle of data mining was to discover factors relating to which customers would default on credit repayments. The second and larger database (80 Mbytes) consisting of 50730 records and 185 fields, was a general customer database, within which the data mining task was to discover general trends in customer behavior.

Several public domain databases from the machine learning repository at UCI [5] were also used during the course of the project, as were a number of large artificially generated test data sets with known properties, used during refinement of the genetic operators.

[5] http://www.ics.uci.edu/AI/ML/Machine-Learning.html, or anonymous ftp from ics.uci.edu in directory pub/machine-learning-databases.

Chapter 3

Survey of Related Work

3.1 Related Work on Genetic Algorithm-Based Learning

Work on genetic algorithm learning systems has traditionally been grouped into one of two general approaches, named after the universities in which they originated. The *Pitt* approach most resembles the traditional genetic algorithm. Here each entity in the population is a set of rules, representing a complete solution to the learning problem. Crossover and mutation are applied in the usual way to create new generations of such populations. The *Michigan* approach, however, has generally used a distinctly different evolutionary mechanism. The population here consists of individual rules, each of which represents a partial solution to the overall learning task. Only through co-operation with the other rules in the population is the overall problem solved. Complex population dynamics and credit-apportionment schemes, such as Holland's *Bucket Brigade* (Holland, 1985), are often required to ensure that collections of rules which work well together are rewarded. Since the GA-MINER system bears most resemblance to the Pitt approach, we restrict our attention here to other systems of this nature.

LS-1 (Smith, 1980, 1983, 1984) is an early example of a machine learning system using the Pitt approach. A knowledge base (population) of production rule sets is manipulated by the genetic operators and at any instant, the best rule set in the knowledge base (as evaluated by a problem specific component of LS-1 called the *critic*) is regarded as LS-1's current hypothesis of the solution to the learning task.

The left hand side of production rules are a combination of state variables reflecting the specifics of the particular problem domain and a number of internal message variables, stored in LS-1's working memory. These are binary encoded and simple pattern matching is used to determine which rules match a particular set of inputs. The right hand side of production rules consist of a message to be posted in the working memory, and optionally, a problem specific operator to be applied in reaction to the current state, which represents the system's response to the outside world. When no operator is specified, the message posted by the production rule is for internal communication purposes only. LS-1 therefore acts as a finite state machine, whose production rules (governing its state transitions and its response to the environment) are evolved by the genetic algorithm.

Smith uses a revised set of genetic operators which manipulate the representation at different levels of granularity reflecting the semantics of the representation and shows that theoretical results for genetic algorithms still hold.

GABIL (DeJong *et al.*, 1993) also uses the Pitt approach for evolving concept descriptions. A concept description is defined as a collection (disjunction) of possibly overlapping classification rules. Left hand sides of classification rules consist of conjunctions of attribute/value-sets, while the right hand sides specify the class to which the rule will assign an example matching the left hand side. Thus the representation is in disjunct normal form (DNF), but with internal disjunction. Only discrete attributes are handled and the value sets are encoded in binary.

The focus of the work is on learning a single concept (or class) descriptions. Here, examples not covered by any production rule in the evolved concept description are regarded as not belonging to the concept. The problems specific to multi-concept learning, where more than one class may exist and examples may be covered by several conflicting production rules, are left as future work.

A modified two-point crossover is used, extended to handle variable length rule sets, together with a standard mutation operator. The goodness of a concept description is measured as the square of percentage of examples correctly classified.

COGIN (Greene & Smith, 1993, 1994) addresses multi-class problem domains by introducing competition for coverage of training examples, encouraging the population to work together to solve the concept learning task. In this respect it is similar to a Michigan approach; however, its evolutionary mechanism is distinctly Pitt-like.

Each rule is a conjunction of attribute/value-sets, encoded in binary. Unlike traditional genetic algorithm implementations, rules are chosen for recombination at random. The newly created rules, together with the existing population of rules, are then ranked in order of fitness and are inserted one by one in this rank order into the next generation of the population, provided they cover some example in the training set which has not already been covered by a previously inserted rule. Any such redundant rules are discarded. The population size thus changes dynamically according to the number of rules required to cover the entire set of training examples.

Fitness is based on an entropy measure, modified according to classification accuracy and both single point and more recently, uniform crossover have been used. Recombination is applied to the left hand sides of rules only. The right hand side of a rule is assigned to be the majority class found within training examples covered by the rule.

REGAL (Neri & Giordana, 1995; Giordana *et al.*, 1994) uses a similarly coverage-based approach for multi-concept learning. A population of conjunctive descriptions for concepts is maintained, forming a redundant set of partial solutions and the overall concept induction task is performed by the entire population of rules. Each rule is evolved in its own right as a partial solution to the problem, however, as in COGIN, a traditional Pitt-like genetic algorithm approach is used to evolve new rules and competition for coverage is used as a means of encouraging co-operation within the population. This is introduced via a new *Universal Sufferage* selection operator, resulting in a collection of rules which tend to cover different training examples and work together to provide a complete solution. The system is distributed in nature and uses first order logic for rule representation. Both two-point and uniform crossover are used, together with task-specific *generalising* and *specialising* operators.

Cui *et al.* (1993) use a parallel genetic algorithm for the identification of "good" and "bad" customers in a credit-scoring application. Solutions are evaluated using either a generic classification accuracy measure, or an application-specific measure of profitability, introduced to reflect the fact that profit is reduced by a larger amount due to an incorrectly classified bad risk than it is increased due to a correctly classified good risk. Results were compared with several other classifiers, including Bayes, k-nearest neighbours and ID3, and while the genetic algorithm was no better in terms of classification accuracy, it outperformed the other methods on the profitability measure.

3.2 Related Work on Data Mining

There are many tools which fall into the broad category of data mining systems, however they are too numerous to describe here. Rather, we restrict ourselves to describe just a few from which GA-MINER has drawn ideas.

The Knowledge Discovery Workbench (KDW) (Piatetsky-Shapiro, 1992) is a database exploration system including components for data visualisation, clustering, summarisation and classification. It has a strong emphasis on the integration of user interaction with automated discovery.

The system incorporates knowledge of the particular domain in three forms, a *data dictionary* which includes descriptions of the fields' respective types and possible values, *field-value taxonomies*, which group field values into classes reflecting features of the domain, and *functional dependencies*, functional relationships which are known to exist within the database.

Each of the discovery features of the system is implemented using its own specialised algorithm. Classification, for example, is performed by constructing a decision tree similar to Quinlan's ID3 (Quinlan, 1986), while summarisation is performed by scanning all records which satisfy a user-defined concept, incrementally building a conjunctive rule which describes these records, and evaluating its goodness using a statistical test.

The visualisation of data is achieved through the use of public domain tools such as gnuplot, which are integrated into the KDW environment. Extensions of the system to allow discovery of changes are also discussed.

EXPLORA (Hoschka & Kloesgen, 1991; Kloesgen, 1992, 1994a, 1994b; Gerbhardt, 1994) is an interactive statistical analysis tool for discovery in databases. A number of *statement types* (or patterns) are defined, and users may select the most appropriate for their particular analysis purposes. For example, patterns exist to detect sufficient and necessary conditions for belonging to a user-defined target group and also to detect shifts in the mean or in the distribution of dependent variables within a particular subset. The subsets of the database are selected using propositional logic constructs, consisting of conjunctions of conditional tests on attribute and values. Users may also specify the variables which may be used in the search and those which will be used as target (dependent) variables and explanatory (independent) variables.

Evaluation of the goodness of patterns is performed mainly using statistical significance tests, and the search of the pattern space is performed using a general graph search, combined with a redundancy filter to ensure that only true and non-redundant statements are generated.

A rule refinement stage is applied to select a moderately sized subset of statements which are sufficiently different from one another and a variety of means are provided for presenting discovered rules to the user.

Holsheimer & Kersten (1994) present an architecture for database mining consisting of a parallel databases server and a data mining tool, Data Surveyor. The authors argue that in many cases, data mining on an entire database is preferable to using a sample, and present a system which efficiently supports the search for classification rules in large databases.

The left-hand sides of classification rules consist of conjunctions of attribute/value-set conditions, while the right hand sides specify the assigned class. The quality of each rule is evaluated by considering the ratio of the number of positive examples covered by the rule to the number of positive and negative examples covered by the rule.

An iterative search strategy is used in which conjunctions are incrementally added to the current rule. Initially the left hand side of the rule is a tautology and at each iteration, the quality of all possible extensions to the rule (i.e. the addition of conjunctions) is evaluated. The best *w* of these are then selected for further exploration, a greedy beam search strategy, and the process continues until no further improvement in quality is achieved or until the length of the rule exceeds a user-defined limit. The algorithm has a *zooming in* behaviour due to conjunctive nature of the extensions, and hence calculating the possible extensions for a rule requires consideration only of the records presently covered by the rule. This continuous reduction in the coverage of rules is exploited by the database server to improve the efficiency of the search.

Freitas & Lavington (1995) also consider data mining from an architectural perspective and argue for an integrated knowledge discovery and database approach. Rather than run the knowledge discovery algorithm on an extract from the database, a *snapshot*, or if this functionality is not provided by the database server, a single relation, is created containing the data upon which the mining task will be performed. This allows the exploitation of database facilities such as indexing and query optimisation for efficient evaluation, and the utilisation of domain knowledge derived from integrity constraints and attribute statistics contained in the database.

Rules are of the form “if D then C” where D is a conjunction of attribute-value conditions and C is a value of a goal attribute. Discretisation of values is required, as the data mining task is reduced to the production of an $m \times n$ contingency table. This is supported by a *Count by Group* database primitive, which can be implemented using a standard SQL *group by* query. Results are compared on three architectures and suggestions are made for a modified version of the primitive, requiring extensions to SQL, to support increased efficiency.

Forty-niner (Zytkow & Baker, 1991; Zytkow & Zembowicz, 1993) searches for regularities in databases, that is, a pattern and the range within which it holds. The representation of a range is as a conjunction of attribute/value-sets, while a pattern is either a function (an equation relating the attributes) or a contingency table. The search algorithm initially concentrates on regularities between attribute pairs in subsets of the database, and allows user interaction to then focus a more specific (and more costly) search, to refine the most promising of the discovered regularities.

3.3 The GA-MINER Approach

GA-MINER attempts to bring together research on both genetic algorithms and data mining and as such, draws on the results and experience of much of the work described above.

We present a system for pattern discovery in databases which searches for patterns along the lines of those used in EXPLORA and Forty-Niner, but which uses a Pitt-like genetic search method. The ultimate aim of GA-MINER is to perform unsupervised knowledge discovery, and while the system falls some way short of this ambitious goal, it does operate with the minimum of user interaction to perform an undirected pattern search of the database. Facilities are also provided to allow the user to specify particular areas of interest, should they so desire, or to suggest rules to the system which GA-MINER may then refine into better rules.

We exploit the parallelism inherent in the GA approach using RPL2 (Radcliffe & Surry, 1994c; Surry & Radcliffe, 1994b) an interpreted and extensible language for the development of parallel genetic algorithms. We demonstrate that although the system is capable of pattern search on a stand-alone desktop PC, it is also capable of scalable performance to exploit the parallel resources of powerful supercomputers.

Chapter 4

Specification of the GA-MINER System

In this section, we present a specification of GA-MINER in some detail. Section 4.1 presents a general overview of GA-MINER, considering its integration with RPL2 and outlining its data model and main features. Subsequent sections consider the component parts of the system in more detail, starting in section 4.2 with a description of the genetic algorithm currently used by GA-MINER. Sections 4.4, 4.5, 4.6 and 4.7 describe the patterns and evaluation functions supported by GA-MINER, while section 4.8 describes the pattern templates which are used to select patterns and influence the scope of the search by constraining the population to be of a particular form. The operators written for the RPL2 rule library are specified in section 4.9, while section 4.10 describes the data format used by GA-MINER. Section 4.11 briefly considers how domain knowledge may be incorporated into a genetic algorithm and finally, in section 4.12, we describe the prototype graphical user interface to the system.

4.1 Overview of GA-MINER

The GA-MINER software is implemented as a new representation and library of operators for RPL2. As such, it is able to exploit the interpreted environment of RPL2 for fast interactive development and evaluation of new genetic data mining techniques, while at the same time utilise the parallelism provided by RPL2.

During development, RPL2 plans (section 4.1.3) are used to configure the application and set the control parameters of the genetic operators, while *pattern templates* (section 4.8) are used to control the form of patterns present in the population.

A graphical user interface is provided to hide the developers interface from users, and present discovered patterns in a visual form.

The architecture is highly amenable to the incorporation of domain-specific, non-genetic local search algorithms, which can exploit available domain knowledge to locally improve patterns found by the genetic algorithm. Such hybrid algorithms, termed *memetic* algorithms (Moscato & Norman, 1992; Radcliffe & Surry, 1994b), have been shown to be extremely powerful in other application domains, often obtaining higher quality solutions than either the genetic or local search algorithms could obtain on their own.

4.1.1 Integration of New Operators Into RPL2

The integration of new operators into RPL2 is straightforward. An *interface header file* is written for each new library (Surry, 1993b), and this contains an entry for each function in the library, specifying the name

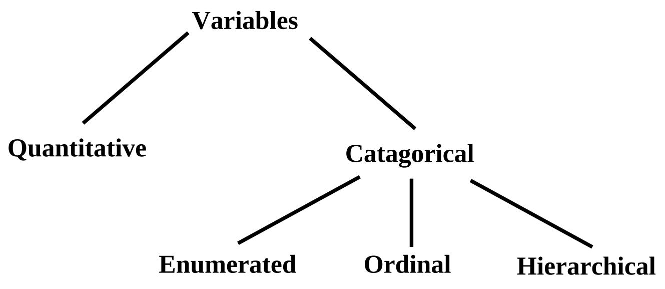


Figure 4.1: Types of variable.

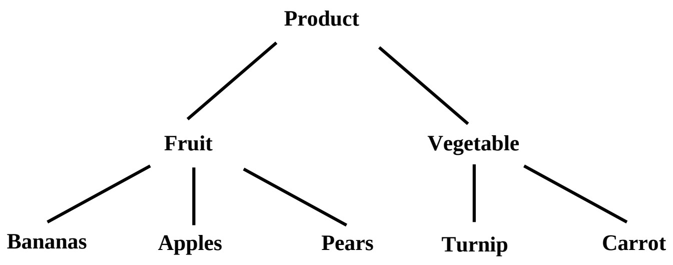


Figure 4.2: A variable hierarchy may contain useful domain knowledge.

which is to be used for the RPL2 operator corresponding to that function. Users then compile a new RPL2 executable, and the compilation process automatically links in the new functions and binds them to the specified operator names.

If the parallel features of RPL2 are to be made available, two additional functions must be written and included in the library, namely a *pack* and an *unpack* function. The pack function is used to flatten the new genome representation into a sequence of consecutive bytes, in order that it may be transfered between processes[1]. The unpack function performs the reverse operation, translating the byte sequence back into a genome representation. These two functions must be registered in the interface header file as before.

Operators are made available for use within RPL2 plans by including the library name in the *use directive* at the start of the plan. For an example, see section 4.9.1.

All the base functionality of RPL2 remains available within the modified executable.

4.1.2 Data Model

Data is currently read into GA-MINER using the `ReadData` function (see section 4.9.2). GA-MINER uses the relational model (Ullman, 1988) for data storage and, as is common in machine-learning circles, this is restricted to a single relational table containing all required data.

Variables (relational database fields) are assumed to be either categorical or quantitative. Categorical variables may be unordered (enumerated), ordinal (ranked) or hierarchical (see figure 4.1).

The purpose of allowing variable hierarchies or orderings in the underlying model is to allow the expression of domain knowledge which may be useful in the search. For example, if a rule within a retail sales database applies to both *bananas* and *apples*, the knowledge that *fruit* is a generalisation of these concepts may be useful in deriving new rules (see figure 4.2).

Variables are labeled as either *dependent* or *independent*. Independent variables are generally those whose value is determined by factors outside the database. Dependent variables are those whose value is believed to be affected by the independent (explanatory) variables in the database.

[1] A rule specification is a compound structure rather than a simple string

The designation of fields as “dependent” or “independent” is really a further expression of domain knowledge which allows the search to ignore certain patterns. For example, in a retail sales database, the weekly hours of sunshine may be included along with sales figures in the expectation that the weather may effect sales of certain products. Here, the sales of various items are dependent variables, while the weekly hours of sunshine is an independent variable. In this context it does not make sense to make the weekly hours of sunshine a dependent variable, as we do not expect that sales of a particular product will affect the weather.

Data elements within quantitative fields may be integer or real, while data elements with qualitative fields may be integers or strings.

4.1.3 RPL2 Plans

The genetic algorithm used by GA-MINER is expressed as an RPL2 plan (see appendix A for an example plan), and is therefore easy to modify. The plan most used during the course of this project is described in section 4.2 below, however, it is important to note that other algorithms are easy to implement by modifying the plan. Indeed, completely new genetic algorithm schemes, such as the coverage-based schemes used by REGAL and COGIN are relatively easy to implement by simply integrating any necessary additional operators and modifying the RPL2 plan as appropriate.

4.2 The Genetic Algorithm

The genetic algorithm used by GA-MINER is defined by its corresponding RPL2 plan, which uses a two-dimensional fine-grained structured population model, where each genome in the population is permitted to recombine only with other genomes within a finite local deme as described in section 2.2.4. Such a population structure helps prolong global diversity within the population, preventing premature convergence on a single pattern, and also allows local niching, which tends to result in exploration of several areas of the search space within the same population. This matches well with the goal of finding several patterns within the database, rather than just a single pattern.

The deme for the fine-grained diffusion model is set to a low value, usually 1 or $\sqrt{2}$, giving either four or eight possible reproductive partners for each population member at any particular generation. A generational update of the population is used. For each genome in the population, a reproductive partner is selected from its local deme using tournament selection.

The two genomes are then combined using the crossover operator. The newly created genome and the original genome are then collected together and tournament selection is used again to select one as the replacement genome for that location in the structured population.

The algorithm is parallelised using RPL2’s `structfor` construct, which performs a simple data decomposition of the population across processors. This is particularly suitable for a fine-grained population structure as the interaction between genomes is restricted to a local neighbourhood, and the evaluation of newly created genomes may be performed completely independently from each other.

The crossover operator is defined at a variety of levels, reflecting the structure of the representation. Within subset descriptions (see section 4.4.1) crossover at the disjunct clause level is based on uniform crossover, but enforces positional alignment of component clauses. Both uniform and single-point crossover are used at the clause level, while crossover at the term level is again based on uniform crossover. Details of the crossover functions are presented in section 4.9.5.

Mutation (section 4.9.6) is also defined at a variety of levels, with separate probabilities specified for mutating disjuncts, clauses, terms, attributes, values and ranges.

Clauses, terms and values are added or deleted with specified probabilities and can be regarded as a distinct operation from standard crossover or mutation (although they are actually implemented as part of the mutation operator). Again this reflects the nature of the application domain, with these operators either specialising or generalising a pattern.

GA-MINER collects sets of patterns or rules during the run of the algorithm for presentation to the user. The sets are of a user-specified maximum size and are initially empty. After each new generation of patterns has been created, the best *n* patterns in the new population (*n* is a user specified parameter) are considered for inclusion in the "rule set". GA-MINER currently uses a simple heuristic for updating the rule set, based on a strategy of maintaining the rule set at it maximum size and continually replacing either the lowest fitness rule in the set or the most similar rule in the set (if the similarity is over a threshold) by a higher fitness rule. The details are given in section 4.9.8.

4.3 Reasons for Abandoning the Hierarchical Genetic Algorithm

The original GA-MINER project proposal suggested the use of a hierarchical genetic algorithm consisting of two levels; a "low-level" genetic algorithm to evolve individual rules using niched competition and structured population models, and a higher level genetic algorithm to evolve sets of such rules which give good coverage of the search space.

However, early in the project it beacume clear that this two level scheme was unnecessary and over-complex. The concept of a good rule set is, like the concept of a good rule, hightly subjective in nature. Wheras we require a mathematical evaluation function to estimate the goodness of an individual rule, it seems unduly complex to require an evaluation function to estimate the goodness of a rule set, since all that we require from such a set in the initial version of GA-MINER is that each rule is sufficiently different from the others in the set.

Clearly, a straightforward goal such as this can easily be achieved through the use of simple heuristics, and at far lower computational expense than could be acheived by a second genetic algorithm. Thus, the creation of rule sets in GA-MINER is performed using an incremental set update strategy (see functions `RuleSetInit` in section 4.9.7 and `RuleSetUpdate` in section 4.9.8).

4.4 Specification of Rule Representation and Evaluation

In this section we specify the pattern forms supported by GA-MINER, together with their respective evaluation functions.

The basis for all supported patterns are the *subset descriptions*, which are used to pick out subsets of the database and form the main heritable units which are manipulated by the genetic algorithm. These are described in section 4.4.1. The main pattern forms supported by GA-MINER are then described, namely explicit rule patterns (section 4.5), patterns comparing distributions (section 4.6) and patterns describing correlations between attributes (section 4.7).

Much of this section, and section 4.8 is reproduced from Flockhart (1995b).

4.4.1 Subset Descriptions

Subsets descriptions are clauses used within the various patterns to pick out subsets of the database. The subsets are formed by selecting those data records which satisfy the constraints specified in the clause. Such a data record is said to *match* the clause.

The form of clauses is based on *disjunctive normal form* (DNF), with the slight extension to allow value sets rather than just single values at the *Term* level. This effectively adds a further layer of disjunction at the lowest level. A *Backus-Naur form* (BNF) (Aho *et al.*, 1986) description of the grammar is shown below:

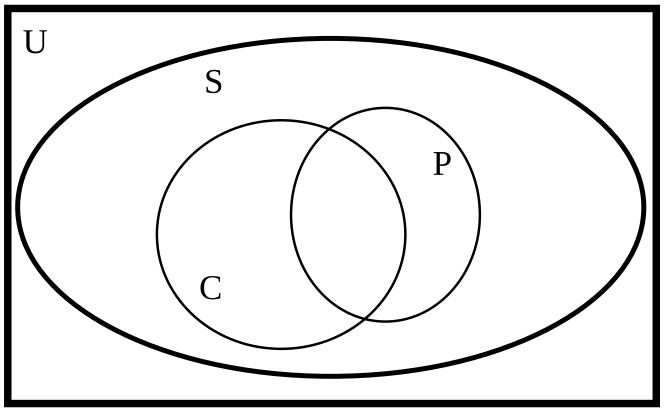


Figure 4.3: Venn diagram illustrating the selection of subsets from the database.

Disjunct Clause	::=	*Clause* [**or** *Clause*]
Clause	::=	*Term* [**and** *Term*]
Term	::=	*Attribute* = *Value Set*
	|	*Attribute* **in** *Range*

Operators '=' and '**in**' have highest precedence, followed by '**and**' and finally '**or**' which has lowest precedence.

The '*Attribute* = *Value Set*' form of term is referred to as a *value term*, or *attribute-value constraint*. The '*Attribute* **in** *Range*' form of term is referred to as a *range term* or *attribute-range constraint*. Value terms are restricted for use with categorical variables, while range terms are used for quantitative variables.

4.5 Explicit Rule Patterns

These patterns are formed from three subsets extracted using subset descriptions which we will call *S*, *C* and *P* (*specificity*, *condition* and *prediction* respectively) . The action of selecting subsets can be visualised using a Venn diagram (see figure 4.3). The *S* clause is used to restrict the domain of the rule to a particular subset of records within the full database *U*. Often the *S* clause will be a tautology, in which case the rule applies to the entire database. Within this selected subset, we then evaluate the goodness of the rule "**if** *C* **then** *P*" (one-way implication) or "*C* **if and only if** *P*" (double implication). In many cases, evaluation functions may be slightly adjusted to score either of these two rule forms and we therefore consider them together.

The *accuracy* of a rule is defined as the number of records satisfying *C* **and** *P* divided by the total number of records satisfying *C*. i.e.

$$\text{accuracy} = \frac{|S \cap C \cap P|}{(|S \cap C \cap P| + |S \cap C \cap \overline{P}|)}$$

The *coverage* of a rule is defined as the number of records satisfying *C* **and** *P* divided by the total number of records satisfying *P*. i.e.

$$\text{coverage} = \frac{|S \cap C \cap P|}{(|S \cap C \cap P| + |S \cap \overline{C} \cap P|))}$$

The *support* of a rule is defined as the number of records satisfying *C* divided by the total number of records. i.e.

$$\text{support} = \frac{|S \cap C|}{|S|}$$

When rules are presented by GA-MINER, all of these quantities are displayed.

Rule patterns for single implication take the conceptual form:

> **when** *S*, **if** *C* **then** *P*.

As noted precisely, a more precise interpretation would be

> Within S : $\mathcal{P}(P|C) > \mathcal{P}(P)$

Example

> **when** *Country* = ’UK’,
> **if** *Month* **in** $\{$‘November’$\}$ **and** *Rainfall* $>$ 30
> **then** *Sales of Umbrellas* $>$ 100.
>
> Accuracy is 82%
> Coverage is 55%

Rule patterns for double implication take the form:

> **when** *S*, *C* **if and only if** *P*,

or more precisely,

> Within S : $\mathcal{P}(P|C) > \mathcal{P}(P)$and $\mathcal{P}(C|P) > \mathcal{P}(C)$

Example

> **when** *Country* = ’UK’,
> *Month* **in** $\{$‘December’$\}$ **and** *Rainfall* $>$ 20
> **if and only if** *Sales of Umbrellas* $>$ 100.
>
> Accuracy is 77%
> Coverage is 65%

4.5.1 Interpreting Contingency Tables as Rules

In general we do not manipulate rule patterns in the form presented above. Rather, the basis for all rule patterns in GA-MINER is a 2×2 contingency table containing the number of data records that fall into each of the four subsets $S \cap \overline{C} \cap \overline{P}$, $S \cap \overline{C} \cap P$, $S \cap C \cap \overline{P}$ and $S \cap C \cap P$ respectively, commonly known as cell counts (see table 4.1).

Clearly, the table can be interpreted as a rule for either single or double implication, and is presented as such to users. However, the use of a contingency provides a unified structure for these patterns and also for other more general patterns.

For example, consider the pattern:

> **when** *S*, *C* and *P* are related.

More precisely, the term “related” can be taken to mean “not statistically independent”. For example, initially assuming independence of *C* and *P*, we may look for unusually high or low numbers of records falling into each of the four subsets.

To see how this can be interpreted a rule, consider an example where an unusually high number of data records fall into the $S \cap C \cap P$ subset and an unusually low number fall into the $S \cap C \cap \overline{P}$ subset. This means that the rule **if** *C* **then** *P* is true for an unusually high number of data records. Its accuracy may not be high, but it is higher than in the expected case.

$S \cap$	$\overline{P}$	P
$\overline{C}$	$\lvert S \cap \overline{C} \cap \overline{P}\rvert$	$\lvert S \cap \overline{C} \cap P\rvert$
C	$\lvert S \cap C \cap \overline{P}\rvert$	$\lvert S \cap C \cap P\rvert$

Table 4.1: A 2×2 contingency table.

Similarly, if an unusually high number of records fall into the $S \cap C \cap P$ and $S \cap \overline{C} \cap \overline{P}$ subsets, together with an unusually low number of records in the $S \cap \overline{C} \cap P$ and $S \cap C \cap \overline{P}$ subsets, the rule *C* **if and only if** *P* is seen to apply to a greater extent than expected.

When these rules are presented by GA-MINER, the 2×2 contingency table is shown together with the expected cell counts.

Example

when *Country* = 'UK',
the subsets satisfying *Month* **in** {'March'} **and** *Rainfall* > 25
and *Sales of Umbrellas* > 100 are unusually related.

(Expected cell counts in brackets)

S **and**	**not** *P*	*P*
not *C*	70 (40)	40 (60)
C	30 (50)	60 (50)

Or interpreting this as a rule:

when *Country* = 'UK',
Month **in** {'March'} **and** *Rainfall* > 25
if and only if *Sales of Umbrellas* > 100.

Accuracy is 57%
Expected Accuracy was 20%

Coverage is 40%
Expected Coverage was 30%

Note that we present both the expected accuracy and coverage alongside the actual accuracy and coverage calculated for the rule.

4.5.2 Evaluation Functions

One may think that accuracy, as defined above may be used as an evaluation function. However, without including some notion of support or coverage, it is possible that this function will tend to select trivial knowledge (a rule that applies to one data record will be 100% accurate, but entirely useless). The inclusion of multiplicative factor to encourage larger C subsets addresses this problem to some extent, however, since the rule "**if** *C* **then** *true*" is 100% accurate, rules with large subsets P achieve undeservedly high fitness. If the right hand sides of rules are fixed however (which in not true in the general case, but is an important special case) accuracy adjusted by some multiplicative factor (e.g. $\log |C|$ may perform reasonably well.

In general, the accuracy, coverage and support of a rule are useful descriptive notions which are easily understood, however, they are not generally suitable as evaluation functions in an unmodified form. To be useful, evaluation functions must take into consideration, either implicitly or explicitly, the degree to which a rule departs from the expected.

The evaluation functions considered by GA-MINER are described below:

Rule interest : This evaluation function was suggested by Piatetsky-Shapiro (1991) and is essentially $|S \cap C \cap P| - (|S \cap C| \times |S \cap P|)/|S|$.

J-measure : Suggested by Smyth & Goodman (1991), this takes the general form:

$$\text{J-measure}(C, \{P_j\}) = \sum_j \frac{|C \cap P_j|}{|S|} \log_2 \frac{|C \cap P_j|\,|S|}{|C|\,|P_j|}$$

where $\{C_i\}$ and $\{P_j\}$ are partitions of S. In our case, $\{C_i\} = \{C, \overline{C}\}$ and $\{P_j\} = \{P, \overline{P}\}$.

Information gain : In the general case, when $\{C_i\}$ and $\{P_j\}$ are partitions of S, the information gain originally derived by Shannon (see Frawley, 1991) is:

$$\text{information gain } = \sum_{i,j} \frac{|C_i \cap P_j|}{|S|} \log_2 \frac{|C_i \cap P_j|\,|S|}{|C_i|\,|P_j|} = \sum_i \text{J-measure}(C_i, \{P_i\})$$

Product moment correlation : *C* and *P* are considered as binary variables *c* and *p* respectively, which take the value 1 if a data record matches and 0 otherwise. The product moment correlation, ρ, is then calculated as given in section 4.7.2 (substituting *c* and *p* for A and B respectively).

Chi-square, χ^2 : Given a 2 × 2 contingency table, containing the actual counts n_{11}, n_{12}, n_{21} and n_{22} of records falling into each of the four subsets, we may calculate the column sums c_1 and c_2, row sums r_1 and r_2, and total number of records n respectively as shown:

$S \cap$	$\overline{P}$	P	
$\overline{C}$	n_{11}	n_{12}	$r_1 = n_{11} + n_{12}$
C	n_{21}	n_{22}	$r_2 = n_{21} + n_{22}$
	$c_1 = n_{11} + n_{21}$	$c_2 = n_{12} + n_{22}$	$n = n_{11} + n_{12} + n_{21} + n_{22}$

Assuming the independence of the two variables *C* and *P*, the expected number of records in each subset e_{11}, e_{12}, e_{21} and e_{22} respectively may be calculated from the product of its respective row and column probabilities using the multiplicative law of probability. Some simple arithmetic shows that the expected number of records in each subset is therefore:

$$e_{ij} = \frac{r_i \times c_j}{n}, i, j = 1, 2.$$

It can be shown that under the assumption of independence of *C* and *P* the χ^2 test statistic,

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{(n_{ij} - e_{ij})^2}{e_{ij}}$$

follows a χ^2 distribution with a single degree of freedom. Thus, high values of χ^2 provide evidence that the variables *C* and *P* are not independent.

In a 2 × 2 table, the χ^2 simplifies to:

$$\chi^2 = \frac{|S|(\mathcal{P}(C \cap P) - \mathcal{P}(C)\mathcal{P}(P))^2}{\mathcal{P}(C)\mathcal{P}(\overline{C})\mathcal{P}(P)\mathcal{P}(\overline{P})} = \frac{|S|(\text{rule interest})^2}{\mathcal{P}(C)\mathcal{P}(\overline{C})\mathcal{P}(P)\mathcal{P}(\overline{P})}$$

Note that the χ^2 statistic depends on the number of records in the contingency table (i.e. the size of set S) and so cannot be used to directly compare patterns which differ in the magnitude of S. In such cases, *Cramer's V coefficient* (Zytkow & Zembowicz, 1993) may be used, which for a 2 × 2 contingency table is defined as

$$V = \sqrt{\frac{\chi^2}{|S|}}.$$

The χ^2 statistic is (up to a constant $|S| \log_e 2$) an estimate of the information gain when $\mathcal{P}(P_j|C_i) \simeq \mathcal{P}(P_j)$.

Gini index or Twoing value (Breiman *et al.*, 1984) : These two measures are in fact equal to a constant factor for a 2×2 contingency table and are hence included together.

$$\text{twoing value} \; = \frac{1}{2} \text{ gini index } = \frac{(|C \cap P| \, |S| - |C| \, |P|)^2}{|S|^2 |C| \, (|S| - |C|)}$$

For a 2×2 table, the twoing value is empirically a good approximation to the information gain.

4.6 Patterns Comparing Distributions

These patterns look for a difference in the distribution of a variable within two subsets. Two comparisons are supported; in both cases the pattern consists of a quantitative variable, A, together with a BNF description of two subsets, *S* and *C*. The variable A is often referred to as a *hypothesis variable*, since pattern instances are essentially hypothesis about A.

4.6.1 Mean Comparison

This pattern takes the form:

The mean of A when *S* **and** *C*
is significantly different from the mean of A when *S*.
i.e. $|\overline{\mu}(C \; \cap \; S) - \overline{\mu}(S)| \gg \; 0.$

Example

The mean of *Sales of Umbrellas*
when *Country* = 'UK' **and** *Month* **in** $\{$'March'$\}$ **and** *Rainfall* > 25
is significantly different from the mean of *Sales of Umbrellas*
when *Country* = 'UK'.

4.6.2 Evaluation Function

Student's t-test statistic : The most obvious evaluation function for the goodness of this pattern is Student's t-test (Mendenhall & Beaver, 1994), a widely used statistical test which may be used to compare two population means.

If $\bar{x}_1$ and $\bar{x}_2$ are the sample means of A in subsets S and $S \; \cap \; C$ respectively, $n_1 \; = \; |S|$, $n_2 \; = \; |S \; \cap \; C|$, and s is a pooled estimator of the standard deviation of A, it can be shown that under the assumption that $\bar{x}_1$ and $\bar{x}_2$ are equal, the test statistic t

$$t = \frac{(\bar{x}_1 - \bar{x}_2)}{s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

follows Student's t-distribution with $n_1 + n_2 - 2$ degrees of freedom.

Here, the pooled estimator of the standard deviation of A is calculated by

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{(n_1 - 1) + (n_2 - 1)}}$$

where s_1 and s_2 are the sample standard deviations of A in S and $S \cap C$ respectively.

Thus, values of t far from zero provide evidence that our original hypothesis (the mean of A is the same in both subsets) is false. Again, a significance level can be determined for this finding, however GA-MINER makes use only of the absolute value of the t statistic to evaluate the relative goodness of these patterns.

This test assumes that the underlying distribution of A is normal. When this can not be guaranteed, the more general location comparison pattern is more appropriate (see section 4.6.3).

4.6.3 Location Comparison

This pattern takes the form:

The distribution of A when *S* **and** *C*
is significantly shifted from the distribution of A when *S*.

Example

The distribution of *Sales of Umbrellas*
when *Country* = 'UK' **and** *Month* **in** {'March'} **and** *Rainfall* > 25
is significantly shifted from the distribution of *Sales of Umbrellas*
when *Country* = 'UK'.

4.6.4 Evaluation Function

Mann-Whitney z statistic : The Mann-Whitney U-test (Mendenhall & Beaver, 1994), which is equivalent to Wilcoxen rank sum test (Gibbons, 1985), is a non-parametric statistical test used to check for a shift in location of a distribution. It is based on rank orderings of the samples and being a non-parametric method, makes no assumptions about the underlying distribution of the variable A. We use the large sample formulation which is valid when the sizes of the subsets are both greater than 10 (Mendenhall & Beaver, 1994).

The data records in the two subsets are first jointly ranked based on the value of variable A, assigning 1 to the smallest observation, 2 to the second smallest, etc. Ties in the ordering are handled by taking the average of the ranks which would have been assigned to the records had they been arbitrarily ordered, and assigning this average rank to each of the tied records.

Let T_1 and T_2 be the sum of the ranks in subsets S and $S \cap C$ respectively. We then calculate U, which is the smaller of

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - T_1$$

and

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - T_2.$$

Finally we calculate the test statistic z:

$$z = U - (n_1 n_2)/2 \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

Under the assumption that the distribution of A is the same in the two subsets, it can be shown that z follows a z distribution, therefore, the further the value of z from zero, the greater the evidence that the two population distributions are shifted in location. Once again, GA-MINER does not calculate a significance level, but uses the z statistic directly as a measure of the goodness of the pattern.

4.7 Correlation Patterns

These patterns search for a correlation between pairs of variables that perhaps only hold within certain subsets of the database. Two correlation patterns are supported. In both cases the pattern consists of two quantitative hypothesis variables, A and B, along with a BNF description of a subset *S*.

4.7.1 Linear Correlation

This pattern has the general form:

when *S*, the variables A and B are linearly correlated

and expresses a linear relationship between two variables.

Example

when *Country* = ’UK’ **and** *Month* **in** $\{$‘October’$\}$,
the variables *Rainfall* and *Sales of Umbrellas* are linearly correlated.

4.7.2 Evaluation Function

Product moment correlation coefficient, ρ : The product moment correlation coefficient is defined as

$$\rho = \frac{Cov(A,B)}{\sqrt{Var(A)Var(B)}}$$

where $Cov(A,B)$ is the covariance of A and B and $Var(A)$, $Var(B)$ are the variances of A and B respectively.

Values of ρ near ± 1 indicate a linear relationship, hence the absolute value of ρ may be used as an evaluation function to indicate linear relationships.

$\rho \times$ **support** : The product moment correlation multiplied by the support to encourage larger subsets to be formed.

t-statistic for no linear correlation : It can be shown that under the assumption of no linear correlation between A and B, the statistic

$$t = \frac{\rho\sqrt{n-2}}{\sqrt{1-\rho^2}}$$

follows a t-distribution with $n-2$ degrees of freedom. Therefore, larger values of t provide evidence of linear correlation and we may use this as an evaluation function.

4.7.3 Rank Correlation (Monotonicity)

This pattern has the general form:

when *S*, the variables A and B are monotonically correlated

and expresses a monotonic relationship between two variables.

Example

when *Country* = ’UK’ **and** *Month* **in** $\{$‘October’, ‘November’$\}$,
the variables *Rainfall* and *Sales of Umbrellas* are monotonically correlated.

4.7.4 Evaluation Function

Spearman rank correlation coefficient : The Spearman rank correlation coefficient is essentially the product moment correlation coefficient between two sets of rank. We use a simplification of the formula which is applicable if the number of ties in rank are small compared to the number of data records, n in S. Hence, if a_i and b_i are the values of A and B in the ith data record, we define the Spearman rank correlation coefficient as

$$r_s = 1 - \frac{6}{n(n^2-1)} \sum_{i=1}^{n} d_i^2$$

where

$$d_i = a_i - b_i$$

$r_s \times$ **support** : The Spearman rank correlation multiplied by the support to encourage larger subsets to be formed.

t-statistic : as defined previously in section 4.7.2, but with r_s replacing ρ.

4.8 Pattern Templates

GA-MINER supports several pattern forms. Clearly, a mechanism is required for selecting which of the patterns should be used in the search. It is also useful to allow certain features of the pattern to be restricted, thus guiding the search in the direction desired by the user.

For example, in an *Explicit Rule Pattern*, it may be useful to fix the *S* subset description to narrow the domain of the search to people living in Scotland, rather than using the entire database which perhaps covers the entire UK. Alternatively, in a retail sales database, we may wish to find only rules concerning the sales of apples, and wish therefore to restrict the *P* subset description to permit only the variable *Sales of Apples* to appear.

For this reason, GA-MINER employs *pattern templates*. A pattern template can be regarded as an initial genome upon which all other genomes in the population are based. The component parts of the template genome are marked as either *initialised* or *fixed*. Fixed parts of the template genome are inherited by every genome in the population and remain fixed throughout (i.e. they are not modified by crossover or mutation). Initialised parts of the template genome are used to initialise all newly created genomes, however, these may be modified by the genetic operators. The pattern template also specifies the the set of variables from which each of the subset descriptions may be formed, and the maximum number of permitted clauses, terms and values which may appear in each of the subset descriptions.

The language used to describe the pattern template is described in the subsequent sections and an example pattern template description is given in appendix B.

4.8.1 General Form

A pattern template description consists of a *use directive* (see section 4.8.2), followed by optional *variable specifiers* (see section 4.8.3) and an optional *dependent variables specifier* (see section 4.8.4), followed by a *block statement* for each subset description (see section 4.8.5).

4.8.2 Use Directive

The *use directive* specifies the pattern to use in the search, and has the form:

use (**Rule** | **Mean** | **Median** | **LinCorr** | **RankCorr**) **pattern**;

The pattern specifiers indicate the following patterns:

Pattern Specifier	Pattern
Rule	Explicit Rule Pattern
Mean	Mean Comparison Pattern
Median	Location Comparison Pattern
LinCorr	Linear Correlation Pattern
RankCorr	Rank Correlation Pattern

4.8.3 Variable Specifiers

These optional specifiers allow users to initialise or fix any of the hypothesis variables used in patterns.

The basic form of the specifier is:

(**fix** | **init**) (A | B) = *Variable Name*

where *Variable Name* is the name of the desired variable in double quotes, and A and B correspond to the variables A and B in the patterns described in Flockhart (1995b).

When the **init** version of the statement is used, all randomly generated patterns (see section 4.9.3) are initialised with the specified hypothesis variable set to *Variable Name*. The **fix** version of the statement specifies that this variable should remain fixed throughout the search, and not be modified by crossover or mutation (see sections 4.9.5 and 4.9.6).

For example, in a Linear Correlation pattern, the specifier:

fix B = “Sales of Bananas”;

initialises and fixes variable B to be *Sales of Bananas* throughout the search.

4.8.4 Dependent Variables Specifier

The dependent variables specifier is an optional statement which changes the default specification of variables as *dependent* or *independent* The form of the statement is:

dependent *Variable List*;

where *Variable List* is as described in section 4.8.9. Each variable in the variable list is set to be a dependent variable, while all variables not in the list are set to be independent variables.

4.8.5 Block Statement

The block statement groups together the statements which define the template for subset descriptions. The form of the statement is:

begin ((**spec** | **cond** | **pred**) **)** *Statement List* **end (** (**spec** | **cond** | **pred**) **)**

The **spec**, **cond** and **pred** subset specifiers indicate the *S*, *C* and *P* subsets respectively, used in Flockhart (1995b). Blocks may not be nested and the subset specifier used in an **end** statement must match that used in the last unmatched **begin** statement.

The *Statement List* may consist of optional statements to set the maximum number of clauses, terms and values for that subset description (section 4.8.6), an optional statement to define the variables that may be used in that subset description (section 4.8.7), and a description of the template for clauses, terms and values (section 4.8.8).

4.8.6 Setting the Maximum Number of Clauses, Terms and Values

This statement has the form:

(**MaxClauses** | **MaxTerms** | **MaxValues**) = *Integer*;

The **MaxClauses**, **MaxTerms** and **MaxValues** specifiers are used to indicate the maximum number of clauses, maximum number of terms and maximum number of values respectively, which may be used in current subset description (i.e. the subset description specified by the current **begin** and **end** block statements).

Each of these statements is optional. If no statement to set the maximum is included, a default value is used.

4.8.7 Variable Set Specifier

This statement has the form:

select variables from *Variable List*;

and specifies that only variables in the *Variable List* may be used within the current subset description. *Variables List's* are specified as described in section 4.8.9.

4.8.8 Subset Description Specifier

Subset description specifiers allow the user to initialise or fix the clauses, terms and values within a subset description. The basic form of these descriptions is shown below:

Subset Description Specifier	:=	**(** *Disjunct Clause Description* **)** [**:** **fixed**]
Disjunct Clause Description	:=	**(** *Clause Description* **)** [**:** **fixed**] [**or** *Disjunct Clause Description*]
	|	**true**
Clause Description	:=	**(** *Term Description* **)** [**:** **fixed**] [**and** *Clause Description*]
Term Description	:=	(**fix** | **init**) *Variable Name* = *Rhs Description*
Rhs Description	:=	(**fix** | **init**) *Value*
	|	(**fix** | **init**) *Numeric* .. (**fix** | **init**) *Numeric*
Value	:=	*Integer* | *Quoted String*
Numeric	:=	*Integer* | *Real*

The optional **fixed** directive appearing after a *Disjunct Clause Description* fixes the number of clauses to the number of clauses appearing in the *Disjunct Clause Description* (i.e. the addition or deletion of clauses is prevented). The special case of an entirely empty disjunct clause may be specified using the '**true**' keyword. This is only useful when the **fixed** option is also selected, in which case the disjunct clause is fixed as a tautology.

The optional **fixed** directive appearing after a in *Clause Description* fixes the number of terms to the number of terms appearing in the *Clause Description* and also prevents the deletion of that clause. The optional **fixed** directive appearing after a *Term Description* prevents the deletion of the term.

The **init** or **fix** directives before the *Variable Name*, *Value* and *Numeric* fields allows the initialisation or fixing of variables, values and numerics respectively.

Example:

Consider the example subset description shown below:

```
(
     (
             ( fix ”Sex” = fix ”M” ) : fixed
             and ( fix ”Age” = init 0 .. init 25 ) : fixed
     ) : fixed
     or
     (
             ( fix ”Age” = init 45 .. init 70 ) : fixed
     )
) : fixed
```

Here, the subset description is appended with a **fixed** directive, indicating that the subsets must contain exactly two clauses. The first clause itself is appended with a **fixed** directive, indicating that it must contain two terms, and may not be deleted. The first term description within this clause is fixed and all components of the first term description are fixed, indicating that the term may not be modified in any way. The second term description is also fixed, indicating it may not be deleted, and its variable is also fixed. However, the range constants are only initialised, so they may be modified by crossover and mutation. Thus the first clause picks out records which have *Sex = "M"* and *Age* lying within some range, initially [0..25].

The second clause is not appended with a **fixed** directive, indicating that it may have additional terms included by the `RandomGenome` function (section 4.9.3) and may also have terms added or deleted. However, the term specified in the description may not be deleted (since it is appended with a **fixed** directive). Its variable component is fixed and so must not be modified, however, its range components are merely initialised so may be changed through mutation and crossover. Thus the second clause initially picks out records which have *Age* lying within the range [45..70], and also satisfy any other terms which are added when a new random genome is generated.

4.8.9 Variable List Descriptions

Variables lists are specified using any of the following forms:

- **Explicit Variable List**

  Variable Name [, *Variable Name*];

  Here the (quoted) variable names are simply given in a comma-separated list.

- **Derived List**

  all [**except** *Explicit Variable List*];
  all dependent [**except** *Explicit Variable List*];
  all independent [**except** *Explicit Variable List*];

  Here the variable list is derived from an existing known variable list, by excluding the variables named in the *Explicit Variable List*. The **all** , **all dependent** and **all independent** specifiers indicate that the new variable list is to be derived from all variables, all dependent variables, or all independent variables respectively. The current setting of dependent and independent variables is used, which may have been changed from the default using the *dependent variable specifier* (section 4.8.4).

  For example, the variable list specifier:

  all dependent except “Sales of Bananas”, “Sales of Apples”;

  indicates that the variable list should contain all dependent variables except *Sales of Bananas* and *Sales of Apples*.

4.9 Specification of RPL2 Rule Library Operators

RPL2 plans written for data mining use both standard RPL2 operators and a number of operators specific to data mining, which are implemented in the RPL2 Rule Library. These include data access operators, operators for initial random pattern generation, genetic operators to manipulate the pattern representations, evaluation functions and operators to create and update sets of patterns. This section, much of which is reproduced from Flockhart (1995a), describes the operators included in the Rule library. These operators are summarised in the table below, together with the relevant section number to which readers should refer for more detailed descriptions.

Function	Purpose	Returns	No. of Arguments	Section
`ReadData`	Read data and template description	`void`	2	4.9.2
`RandomGenome`	Generate a new random genome	`genome`	0	4.9.3
`EvalRule`	Evaluate the Rule genome	`void`	2	4.9.4
`Cross`	Crossover two Rule genomes	`genome`	4	4.9.5
`Mutate`	Mutate a Rule genome	`void`	11	4.9.6
`RuleSetInit`	Initialise the rule set	`void`	1	4.9.7
`RuleSetUpdate`	Add a new genome to the rule set	`void`	4	4.9.8
`PrintGenome`	Print a Rule genome	`void`	1	4.9.9
`RuleSetPrint`	Print the rules in the rule set	`void`	0	4.9.10

4.9.1 Initialisation

Before proceeding to describe the operators in more detail, we first briefly consider how the operators are used within an RPL2 plan. The functions in the Rule library are made available within a plan by including the Rule library in the parameter list of an RPL2 `use` statement (Surry, 1993b). For example,

```
    use StdInst, Rule;
```

specifies that both the standard library and Rule library are to be used.

An example RPL2 plan is shown in appendix A. In the extract below, we consider the use of the RPL2 Rule library operators to initialise a two dimensional fine-grained population structure of genomes and display the best one. For clarity, line numbers have been added and variable declarations and initialisations have been omitted.

Example

```
(1)  ReadData(<database file>, <template file>);
(2)  RuleSetInit(ruleSetSize);
(3)
(4)  structfor [*,*]
(5)          Randomize(seed);
(6)          g := RandomGenome();
(7)          EvalRule(g, evalFn);
(8)  endstructfor
(9)
(10) gBest := ReduceRawBest(g,bMaxIsBest);
(11) PrintGenome(gBest);
```

The `ReadData` statement (section 4.9.2) on line 1 must be the first Rule library function called, as this both reads the data into the system, performs normalisation and initialises the pattern template. The `RuleSetInit` statement on line 2 then sets the size of the rule set to be `ruleSetSize` and initialises it to be empty.

The `structfor/endstructfor` statements on lines 4 and 8 delimit the block of statements (from lines 5 to 7 inclusive) which may be executed for each population element in parallel (Surry, 1993b). The

RPL2 standard library function, `Randomize`, is first used to initialise RPL2's random number generator. The Rule library function `RandomGenome` is then called to generate a new random genome, and the `EvalRule` function is used to evaluate this genome using the specified evaluation function (section 4.9.4).

The `ReduceRawBest` function from RPL2's standard operator library then selects the best genome from the population, and finally the Rule library's `PrintGenome` function is used to display this rule to the user.

4.9.2 Function `ReadData`

```
void ReadData(
 string  sName       Base Name of the data and configuration files.
 string  sTemplate   Name of the template description file.
)
```

Synopsis: Reads data from the file *¡sName¿.dat*, using configuration information from the file *¡sName¿.cfg*. Also reads a pattern template description from the file *¡sTemplate¿*. Quantitative data values are normalised internally to have a mean of zero and a standard deviation of one, however, the original data values are used when rules are presented to the user.

4.9.3 Function `RandomGenome`

```
genome RandomGenome()
```

Synopsis: Generates a new genome, based on the pattern template, but with additional randomly generated parts where permitted.

Hypothesis variables are generated randomly from the set of permitted variables, unless such variables are initialised or fixed by the pattern template. The *S*, *C* and *P* subset descriptions are generated using as described in section 4.9.3.1.

4.9.3.1 Random Subset Description Generation

Any clauses appearing in the subset description of the pattern template are included in the randomly generated disjunct clause. These clauses may be modified as described in section 4.9.3.2.

In addition, a random number between the number of clauses already included from the pattern template and the maximum number of clauses permitted is generated. This number of additional clauses are newly generated using random clause generation (section 4.9.3.2) and included in the new disjunct clause.

4.9.3.2 Random Clause Generation

If the clause to be generated is to be based on a clause from the pattern template, then all terms appearing in the clause from the template are included in the randomly generated clause. These terms may be modified as described in section 4.9.3.3.

In addition, a random number between the number of terms already included from the pattern template and the maximum number of terms permitted is generated. This number of additional terms are generated using random term generation (section 4.9.3.3) and included in the new clause.

4.9.3.3 Random Term Generation

If the term to be generated is to be based on a term from the pattern template then the variable and values or range specified in the pattern template are used in the new term. For value terms, any values appearing in the template are included, together with a randomly selected number of additional values (up to the maximum number permitted).

Otherwise, if the term is to be newly generated, a variable is first selected at random from the set of permitted variables and the type of the selected variable determines whether a value or range term is then used. For value terms, a random number of values are selected (up to the maximum number permitted) and are included in the term. For range terms, the minimum and maximum range constants are generated as follows:

One range constant is selected from the range [0,1] using a uniform random number generator. A second constant is then generated by adding the first constant to an offset value (modulo 1.0), which is randomly generated from a normal distribution with parameters μ and σ. The resulting two constants, which lie in the range [0,1], are then scaled to lie in the appropriate range for the current variable. Finally, the minimum and maximum of the two scaled constants are assigned to be the minimum and maximum range constants respectively.

4.9.4 Function `EvalRule`

```
void EvalRule(
 genome  g        Genome which is to be evaluated
 int     evalFn   Integer specifying which evaluation function to use
)
```

Synopsis: Evaluates the given genome `g` using the evaluation function specified by the `evalFn` parameter. For some patterns, the `evalFn` parameter is not used, as only a single evaluation function exists. The evaluation function selected by the `evalFn` parameter for each pattern type is given below. These evaluation functions are described fully in Flockhart (1995b).

Pattern Type	`evalFn`	Evaluation Function
Explicit Rule Patterns	1	Rule interest
	2	J-measure
	3	Information gain
	7	Product moment correlation
	8	Chi-square
	9	Gini index or Twoing value
Linear Correlation Pattern	1	Pearson product moment correlation coefficient, ρ
	2	$\rho \times$ support
	3	t-statistic for no linear correlation
Rank Correlation Pattern	1	Spearman rank correlation coefficient, r_s
	2	$r_s \times$ support
	3	t-statistic for no rank correlation
Mean Comparison Pattern	—	T-test statistic
Location Shift Pattern	—	Mann-Whitney U test statistic

This function sets the RPL2 genome fitness (Surry, 1993a) according to the result of the applied function.

Before an evaluation takes place, the pattern is pruned as described in section 4.9.4.1.

The evaluation function keeps track of the number of records in the database which match each term in a pattern, and stores this information along with the term description. Any terms which were uniquely false for some record are also marked.

4.9.4.1 Pruning Patterns

Pruning patterns is performed in order to simplify the subset descriptions, to remove any contradictions contained within them and to ensure the representation constraints required by the genetic operators (i.e. a maximum of one term for each variable appearing in a clause) continue to hold throughout the search.

Pattern pruning of disjunct clauses is described in section 4.9.4.2.

4.9.4.2 Pruning Disjunct Clauses

Each component clause in the disjunct clause is first pruned as described in section 4.9.4.3. Any component clauses which contain no terms are then removed one at a time by moving the last clause in the clause list to overwrite the clause to be deleted. During recombination, a clause is recombined with the clause which shares the same position in other subset description's list of clauses, so this method is least disruptive to the limited alignment which exists.

4.9.4.3 Pruning Clauses

Each component term within the clause is first pruned as described in section 4.9.4.4.

Any value term containing no values is then deleted. Unlike in section 4.9.4.2, the particular method used for deletion need not be specified, since the crossover operator aligns terms according to the variable they contain.

If some of the terms in the clause were never uniquely false at the last evaluation (see section 4.9.4) one of them is selected randomly and removed. Furthermore, terms which are close to being uniquely false are considered uniquely false with some probability.

Finally, multiple terms concerning a single variable are merged into a single term. If this leads to a contradiction, then all terms are removed from the clause.

4.9.4.4 Pruning Terms

Range terms are not pruned further. Value terms have any duplicate values removed.

4.9.5 Function `Cross`

```
genomeCross(
 genome  gA        First parent genome to be used in crossover
 genome  gB        Second parent genome to be used in crossover
 real    rBias     Bias given to one of the parents during crossover
                   (takes values in the range [0,1]).
 real    rPUCross  Probability of using uniform crossover of clauses
                   (as opposed to single-point crossover).
)
```

Synopsis: Perform a recombination of the two parent genomes `gA` and `gB` respectively. Two forms of crossover are supported, based on the usual single-point crossover and uniform crossover operators. Because of the variable length representation used for subset descriptions, the operators used are somewhat different from the conventional single-point and uniform crossover. Hypothesis variables are recombined as described in section 4.9.5.1. Each of the subset descriptions are recombined as described in section 4.9.5.2.

Returns: A new genome which is the recombination of the two parent genomes.

4.9.5.1 Hypothesis Variable Crossover

In patterns which contain two hypothesis variables A and B (i.e. Linear Correlation and Rank Correlation), the child's A and B attributes are selected (independently of each other) from the first parent with probability `rBias` and from the second parent with probability (1–`rBias`).

4.9.5.2 Subset Description Crossover

Each clause in the first parent is crossed with the clause in the corresponding position in the second parent. For each clause, we use uniform clause crossover (section 4.9.5.3) with probability `rPUCross` and single-point crossover (section 4.9.5.4) with probability (1–`rPUCross`).

If the number if clauses in the two parents is different, the clauses which do not have a partner are each included in the child disjunct clause with probability `rBias` in the case of clauses from the first parent and (1–`rBias`) in the case of clause from the second parent, unless the clause is marked as fixed, in which case it is included with a probability of 1.

Example

Consider the disjunct clauses A and B below:

> **A** : *Clause A1* **or** *Clause A2* **or** *Clause A3* **or** *Clause A4*
> **B** : *Clause B1* **or** *Clause B2*

Disjunct clause crossover selects clauses $A1$ and $B1$ for crossover and uses uniform crossover or single-point crossover with the specified probabilities. Clauses $A2$ and $B2$ are then selected for crossover, and again uniform or single-point crossover is used according to the specified probabilities. Clauses $A3$ and $A4$ have no partner in disjunct clause B, so these clauses are independently included in the child with probability `rBias`.

4.9.5.3 Uniform Clause Crossover

Uniform clause crossover first performs an "alignment" of terms between the two parent clauses in order that terms concerning the same variable will be crossed with each other. Note that each clause can contain a maximum of one term concerning any particular variable, a condition which is enforced during clause pruning (see section 4.9.4.3).

Then for each aligned pair of terms, term crossover (section 4.9.5.5) is used to produce a new term for the child clause.

Terms without a partner are included in their entirety in the child clause with probability `rBias` in the case of terms from the first parent, and probability (1–`rBias`) in the case of terms from the second parent. Terms marked as fixed are handled specially, and are always included in the child.

Example

Consider the clauses A and B below:

> **A** : *Age = 20 .. 30*
> **B** : *Sex = M* **and** *Age = 0 .. 25*

The alignment of the terms according to variable gives:

> **A** : *Age = 20 .. 30*
> **B** : *Sex = M* **and** *Age = 0 .. 25*

The two terms concerning *Age* are crossed using the term crossover function, while the term concerning *Sex* is included in the child with probability (1–`rBias`).

4.9.5.4 Single-Point Clause Crossover

Single-point clause crossover first performs an alignment of terms between the two parent clauses in order that terms concerning the same variable are aligned. Again, clauses may contain a maximum of one term concerning any particular variable, enforced by clause pruning (see section 4.9.4.3).

A crossover point is then selected, and terms to the left of this point in the first parent and to the right of this point in the second parent are included in the child, provided the maximum number of terms is not exceeded in which case terms from the first parent are given priority. An exception occurs with terms marked as fixed, which are always included in the child clause.

Example

A : *Age = 20 .. 30* **and** *Height = 1.5 .. 2.0*
B : *Sex = M* **and** *Age = 0 .. 25*

The alignment of the terms according to variable gives:

A : *Age = 20 .. 30* **and** *Height = 1.5 .. 2.0*
B : *Sex = M* **and** *Age = 0 .. 25*

Selecting a crossover point between the *Sex* and *Age* variables, the child C then takes terms to the left of the crossover point in clause A, and terms to the right of the crossover point in clause B, giving:

C : *Age = 0 .. 25*

4.9.5.5 Term Crossover

Term crossover is used to combine two terms concerning the same variable. Crossover of *value* and *range* terms is handled differently.

For range terms, the minimum and maximum range constants are selected independently from the parents, using the constant from the first parent with probability `rBias` and the constant from the second parent with probability (1 - `rBias`). If the resulting range is not valid (i.e. the maximum of the range is less then the minimum) it will be repaired later by the term pruning (section 4.9.4.4).

For value terms, all values common to both parents are included in the child. Values unique to one parent are included with probability `rBias` in the case of values unique to the first parent and probability (1 - `rBias`) in the case of values unique to the second parent, provided the maximum number of permitted values is not exceeded, in which case values from the first parent are given priority.

Note that the term crossover propagates all values and range constants marked as fixed.

4.9.6 Function `Mutate`

```
void Mutate(
```

`genome`	`g`	genome which is to be mutated.
`real`	`rPAddDelCls`	probability of adding or deleting a clause.
`real`	`rPAddDelTerm`	probability of adding or deleting a term.
`real`	`rPMutHypAttr`	probability of mutating a hypothesis attribute.
`real`	`rPMutTerm`	probability of selecting a term for mutation.
`real`	`rPMutateAttr`	conditional probability of mutating a variable in a term, given that a term has been selected for mutation.
`real`	`rPMutValue`	conditional probability of mutating a value in a term, given that a term has been selected for mutation.
`real`	`rPChangeConst`	conditional probability of mutating a range constant in a term, given that a term has been selected for mutation.
`real`	`rConstHalfRange`	half range of the interval used for creep mutation.
`int`	`nConstSteps`	number of steps into which the half range should be equally divided.
`int`	`nMaxConstCreep`	maximum number of steps permitted by a single creep mutation of a range constant.

```
)
```

Synopsis Mutates the genome `g`. Hypothesis variables are mutated as described in section 4.9.6.1. Subsets descriptions are mutated as described in section 4.9.6.2.

4.9.6.1 Hypothesis Variable Mutation

For patterns containing hypothesis variables, each hypothesis variable is independently mutated with probability `rPMutHypAttr`, unless the variable is marked as fixed, in which case mutation is prevented.

Mutation replaces the hypothesis variable with another variable randomly selected from the variable set.

4.9.6.2 Subset Description Mutation

Each component clause is first mutated using clause mutation (section 4.9.6.3). A clause is then added or deleted with probability `rPAddDelCls` (clause addition and deletion are equally likely — each has a probability of 0.5).

Clauses are selected for deletion at random. If a fixed clause is selected, its deletion is prevented and another clause is not selected.

For clause addition, the clause is added provided the maximum number of clauses is not exceeded. The new clause is generated at random as described in section 4.9.3.2.

4.9.6.3 Clause Mutation

Each component term is first mutated using term mutation (section 4.9.6.4). A term is then added or deleted with probability `rPAddDelTerm` (once again, term addition and deletion are equally likely, each having a probability of 0.5).

Terms are selected for deletion at random, and if the selected term is fixed, its deletion is prevented and another term is not selected. For term addition, the term is added provided the maximum number of terms is not exceeded. The new term is generated at random as described in section 4.9.3.3.

4.9.6.4 Term Mutation

A term is mutated with probability `rPMutTerm`. Given that a term has been selected for mutation, the variable within a term is mutated with probability `rPMutateAttr`, though variables marked as fixed are not mutated. When a variable is selected for mutation, it is replaced by a randomly selected variable from the variable set. Since the type of this variable may be different from the original, the entire term is re-generated at random as described in section 4.9.3.3.

If the term variable is not selected for mutation, the value or range part of the term is considered for mutation.

For value terms, one of the values is selected at random and is mutated to a randomly selected value from the set of possible valid values for the current variable. Any duplicate values introduced will be removed during term pruning (section 4.9.4.4). Values marked as fixed are not mutated.

For range terms, each range constant is mutated with probability `rPChangeConst`. When a range constant is selected for mutation, a number of creep mutation steps is chosen at random (up to a maximum of `nMaxConstCreep`), and the constant is incremented or decremented (with equal probability) by the creep mutation interval for that number of steps. Range constants marked as fixed are not mutated.

4.9.7 Function `RuleSetInit`

```
void RuleSetInit(
 int  size   specifies the maximum size of the rule set
)
```

Synopsis: Initialises the rule set to be empty and fixes the maximum size to be `size`.

4.9.8 Function `RuleSetUpdate`

```
void RuleSetUpdate(
 genome  g            genome to consider for inclusion in the rule set.
 real    rSim         value of the similarity threshold.
 bool    bMaxIsBest   true if trying to maximise fitness, false otherwise.
 int     nSim         indicates the similarity function to use.
)
```

Synopsis: Considers the genome `g` for inclusion in the rule set. In general, GA-MINER does not search for a single good pattern within a data set, rather it attempts to find several, diverse, good patterns. GA-MINER collects patterns together in a *pattern set*, which is updated as new and better patterns are discovered.

In order to avoid the pattern set becoming full of extremely similar patterns, the inclusion or otherwise of a new pattern in the set must be a function of both the goodness and similarity of the pattern relative to those already in the set.

As a simple heuristic for collecting good and diverse patterns, GA-MINER uses a similarity measure to determine the closeness of patterns to each other. When considering a new pattern for inclusion in the set, we firstly determine the most similar pattern already in the set. If the similarity is over the threshold level, `rSim`, and the new pattern has better fitness, then it replaces the old pattern in the set. If the most similar pattern is below the threshold level, `rSim`, then the new pattern replaces the worst fitness pattern in the set, provided it has a higher fitness.

The similarity measure presently used is based on a simple Euclidean distance function, modified to allow range terms. Although such functions are known to have limitations in high-dimensional space, we have found such a function adequate for producing sets of sufficiently different rules.

4.9.9 Function `PrintGenome`

```
void PrintGenome(
 genome  g   genome to display.
)
```

Synopsis: Displays a pattern to the user.

4.9.10 Function `RuleSetPrint`

```
void RuleSetPrint()
```

Synopsis: Displays the contents of the rule set.

4.9.11 Other Required Functions

As with all RPL2 libraries, functions to implement the *pack*, *unpack*, and *free* operations (Surry, 1993a) must be written. The pack function is used on parallel hardware to convert the user-defined genome to a flat sequence of bytes so that it can be passed from one processor to another by the framework, while the unpack function performs the reverse translation. The free function is used to release memory which was allocated for a new genome, in order that the RPL2 framework can provide automatic garbage collection.

The functional specification of these functions is given in Surry (1993a).

4.10 Data Format

GA-MINER uses the relational data model with data stored in a single fully de-normalised table (Flockhart, 1995b).

Data is presented to GA-MINER in ASCII format, one record per line, with fields separated by a comma. Fields may contain integers, reals or character strings.

Associated with each data file is a configuration file, which specifies the names of the variables in the data set, their type and underlying storage-type, together with an indication of whether they are *dependent* or *independent* variables (Flockhart, 1995b). In the absence of further restrictions imposed by pattern templates, this indicator variable is used to eliminate certain meaningless patterns from the search.

The first two lines of the configuration file contain the number of records and number of fields in the database respectively. Then for each field, the variable name, type, storage-type and indicator variable appears on a separate line. A BNF description of the possible entries is shown below:

Variable Name	::=	*Character String*
Type	::=	`numeric` | `enum` | `ordinal` | `hierarchical`
Storage Type	::=	`int` | `dbl` | `str`
Indicator	::=	`ind` | `dep`

The values of the *Type* specifier are described more fully in Flockhart (1995b). The *Storage Type* values `int`, `dbl` and `str` are used to indicate integers, reals and strings respectively, while the *Indicator* values `ind` and `dep` indicate independent and dependent variables respectively.

Numeric data is normalised internally to have a mean of 0 and standard deviation of 1. However, the original data values are used when patterns are presented to the user.

4.11 Incorporating Domain Knowledge

Genetic algorithms are often characterised as a “weak” search method, in the sense that they are broadly applicable and make relatively few strong assumptions about the problem they tackle. Occasionally they are incorrectly classified as “blind” search methods, or “black box optimisers (Goldberg, 1989).

In fact, both of these characterisations are rather misleading, as a series of papers (Radcliffe, 1992, 1994; Wolpert & Macready, 1995; Radcliffe & Surry (1995)) have shown. It would be more accurate to say that genetic algorithms traditionally capture knowledge (or assumptions) about the problem as hand implicitly rather than explicitly, through the choice of representation and genetic move operators.

Explicit mechanisms for "strengthening" a genetic algorithm include non-random initialisation of the population (Ray, 1992, 1994), incorporation of extra ("non-genetic") local move operators and "hybridisation" with domain-specific heuristics (Davis, 1991). We have been unable in the time available to fully explore these methods of incorporating domain knowledge into GA-MINER. However, we believe that such explicit mechanisms, and in particular the use of local move operators, hold much promise for genetic-based data mining systems, and consider this as essential future work (section 7.2).

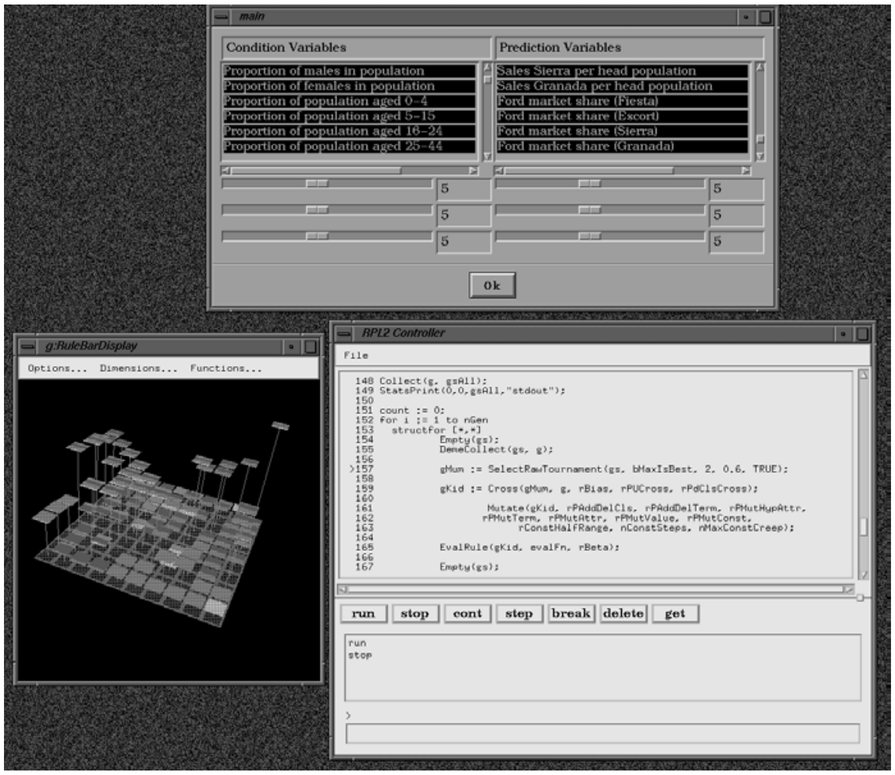
4.12 The Prototype Graphical User Interface

Sufficient time was available towards the end of the research period to construct a prototype graphical user interface for GA-MINER and to integrate this with the RPL2 visualisation software.

A screen shot of the GA-MINER interface and the RPL2 visualiser is shown in figure 4.4. The pattern template window (top) allows simple editing of the pattern template. Presently, users may select the variables which are permitted to appear on the left and right hand sides of rules and the maximum number of clauses, terms and values which may appear in rules. It is planned to extend the interface to allow manipulation of individual terms and values as time allows.

The RPL2 controller window (bottom right) shows the RPL2 plan which is presently running, and in many ways acts like a debugger, allowing breakpoints to be set in the plan and visualisation instructions to be placed on particular lines. Presently, users may not change the value of RPL2 plan variables, for example the mutation rate, during a run of the algorithm, however extensions are planned which will allow such interaction.

The visualisation window (lower left) shows the population of genomes (rules) in a fine-grained population structure. Rules are currently represented in this window by a three dimensional plot of their respective contingency tables, though alternative visualisation routines may be added with relative ease, and each rule's height above the baseplane is proportional to its fitness. Individual rules may be selected from this window and examined in more detail a further window (not shown), which displays the rules in textual form.



Condition Variables	Prediction Variables
Proportion of males in population	Sales Sierra per head population
Proportion of females in population	Sales Granada per head population
Proportion of population aged 0–4	Ford market share (Fiesta)
Proportion of population aged 5–15	Ford market share (Escort)
Proportion of population aged 16–24	Ford market share (Sierra)
Proportion of population aged 25–44	Ford market share (Granada)

```
Collect(g, gsAll);
StatsPrint(0,0,gsAll,"stdout");

count := 0;
for i := 1 to nGen
  structfor [*,*]
            Empty(gs);
            DemeCollect(gs, g);

            gMum := SelectRawTournament(gs, bMaxIsBest, 2, 0.6, TRUE);

            gKid := Cross(gMum, g, rBias, rPUCross, rPdClsCross);

                    Mutate(gKid, rPAddDelCls, rPAddDelTerm, rPMutHypAttr,
                   rPMutTerm, rPMutAttr, rPMutValue, rPMutConst,
                          rConstHalfRange, nConstSteps, nMaxConstCreep);

            EvalRule(gKid, evalFn, rBeta);

            Empty(gs);
```

```
run
stop
```

Figure 4.4: Screen shot of the Rule library GUI and RPL2 visualiser

Chapter 5

Results

Two factors complicate presentation of results from the GA-MINER project. The first, which was discussed at some length in section 1.4, is that our measures of rule quality are all surrogates for our actual objective, which is to find genuinely "interesting" rules. For this reason, showing graphs or statistical summaries of the objective function value as a function of time and patterns searched is of little value. (In any case, they look much as would be expected). For this reason, we find it more illuminating to focus on particular rules found by the system. In this section, a few such rules are presented in detail, illustrating some of the more useful rule types. Further examples are given in appendix E.

The second complication arises because of the commercial sensitivity of the data provided by Barclays. For this reason, rules concerning their data can only be shown in disguised form. Although some examples are included for completeness in appendix E, these are less than illuminating. However, Barclays is developing an ever-stronger interest in work on data mining and has been conducting a study which includes input from the GA-MINER research (see appendix C).

5.1 Rules and their Interpretation

In this section we examine some of the rules and patterns discovered in the data provided by GMAP Ltd., and consider their respective interpretation. Explicit rule patterns are first considered in section 5.1.1 followed by distribution shift patterns in section 5.1.2. No strong correlation patterns were found in the GMAP data, hence we do not show any examples of this pattern form.

5.1.1 Explicit Rule Patterns

Many rule patterns of varying strength were found within the GMAP data, however we restrict ourselves to show just two examples which are representative of the kinds of patterns discovered. In general, the largest number of rules appeared to be concerned with segment D sales and market share, within which Ford's representative vehicle is the Ford Escort, however patterns concerning other market segments were also commonplace.

Rule pattern 1

if Proportion of households with 1 child $\geq$ 0.12
(Approximate percentiles 36% - 100%)
(true: 1618 false: 939 unique false: 272)

and

Number of Ford Dealers $>$ 0
(Approximate percentiles 62% - 100%)
(true: 936 false: 1621 unique false: 809)

and

Proportion of households with 3+ cars **in range** 0.01 .. 0.07
(Approximate percentiles 4% - 86%)
(true: 2038 false: 519 unique false: 108)

then

Ford market share segment F (Sierra) **in range** 0.06 .. 0.75
(Approximate percentiles 30% - 100%)
(true: 1770 false: 787 unique false: 787)

Left hand side matches 19% of the database
Right hand side matches 69% of the database

	Expected	Actual
Accuracy:	69%	93%
Coverage:	20%	27%

An example of such a pattern found by GA-MINER is shown above (rule 1). The true and false count for each clause show the number of times that clause is true and false respectively, while the unique false count shows the number of times the clause is false when all the others are true (see section 6.2). This gives some indication of the relative importance of the terms.

This rule states that in postal districts where there is at least one Ford dealer and the proportion of households with 1 child is relatively high (that is in the top 64% of the distribution) and the proportion of households with 3 or more cars is neither very high nor very low, there is 93% probability that the postal district has a Ford market share of segment F (Sierra) in the top 70%. This compares to an expected probability of 69%, under the assumption of no relationship between the left and right hand sides of the rule.

The genetic algorithm which found this rule used a 10×10 fine-grained structured population model with the J-measure as an evaluation function. The fitness of the rule (0.0512818) compares to an initial mean population fitness of –0.0001256, with standard deviation of 0.0020194. Rules of similar quality were normally found within 100–150 generations, taking approximately 5 minutes wall clock time on an SGI Indy workstation using the 2557 record data set.

Rule pattern 2

if Proportion of households with 1 child ≥ 0.12
(Approximate percentiles 54% - 100%)
(true: 1169 false: 1388 unique false: 239)

and
Proportion of households — Council ≥ 0.23
(Approximate percentiles 70% - 100%)
(true: 732 false: 1825 unique false: 676)

then
Ford market share segment D (Escort) ≥ 0.19
(Approximate percentiles 44% - 100%)
(true: 1394 false: 1163 unique false: 1163)

Left hand side matches 19% of the database
Right hand side matches 56% of the database

	Expected	Actual
Accuracy:	55%	82%
Coverage:	19%	29%

Rule 2 above suggests that postal districts with a high proportion of council housing (top 30% of the distribution) and a high proportion of households with one child (top 46% of the distribution) are 84% likely to have a relatively high Ford market share of segment D (Escort) (in the top 66% of all postal districts).

Once again, a 10×10 fine-grained population structure was used, together with the J-measure evaluation function. The rule fitness is 0.047255, which compares to an initial mean population fitness similar to that given in the description of rule 1.

It is important to note that these rules do not imply causality. So for instance, we may not conclude that Ford market share of segment D is high in a postal district *because* there are a high number of council households and a high proportion of households with 1 child. There may be other factors not included in the database which are the cause of all of these effects. To establish causality is extremely difficult, requiring controlled statistical experiments designed to take into consideration all other possible causes [1]. Clearly such experiments can not be performed on a routine basis to evaluate each rule discovered by a data mining system. In practice therefore, we must attempt to use the rule in conjunction with common sense, perhaps evaluating other empirical evidence which may support the rule.

In cases where the rule seems counter-intuitive, or indicates a surprising relationship, it is perhaps best to engage in further investigation of the result. However, it should be recognised that discovery of unexpected rules is one of the main advantages of an automated data mining system.

[1] Consider, for example, the efforts over many years to prove that smoking causes cancer.

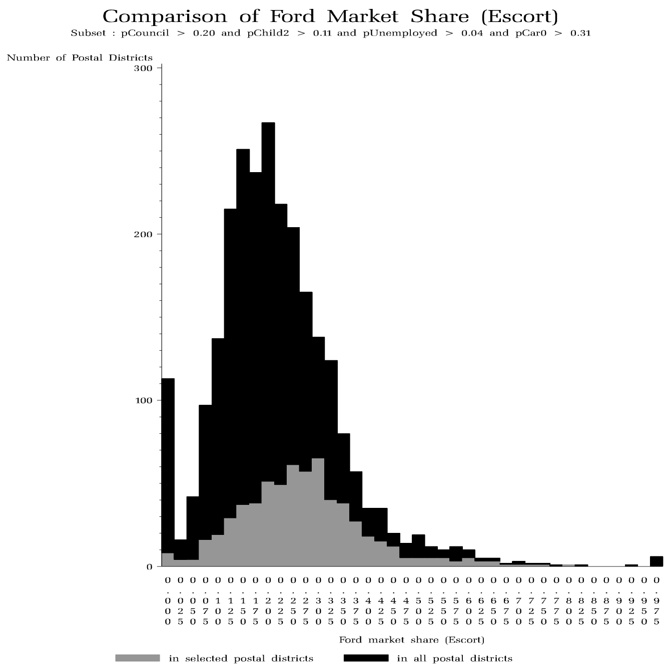


Figure 5.1: Distribution Shift Pattern

5.1.2 Distribution Shift Patterns

Distribution shift patterns can in many ways be used to provide similar knowledge to that imparted by explicit rule patterns.

Pattern 3 is another real example of a distribution shift pattern found in the data provided by GMAP.

Distribution shift pattern 3

The distribution of "Ford market share of segment D (Escort)" when

Proportion of households — Council ≥ 0.20
(Approximate percentiles 62% - 10%)
(true: 929 false: 1628 unique false: 123)

and

Proportion of households with 2 children ≥ 0.11
(Approximate percentiles 26% - 100%)
(true: 1854 false: 703 unique false: 233)

and

Proportion of unemployed in population ≥ 0.04
(Approximate percentiles 52% - 100%)
(true: 1183 false: 1374 unique false: 48)

and

Proportion of households with 0 cars ≥ 0.31
(Approximate percentiles 60% - 100%)
(true: 1015 false: 1542 unique false: 89)

has median 0.28 and is significantly shifted from the
distribution in the database as a whole which has median value 0.20.

The rule indicates that the subset of 408 records selected from the original database of 2557 records by the conjunction of the terms shown, generally have a higher Ford market share in segment D (represented by Ford Escorts). The terms themselves suggest that Ford market share in this segment is higher in postal districts which have a high proportions of council houses, relatively high unemployment, a high proportion of households with no car and relatively few households with 2 children. Figure 5.1 shows the distribution of Ford Market share of segment D (Escort) in both the database as a whole (dark shading) and the selected subset (light shading). One can see from this figure that the selected subset has both a higher mean and median value for market share, and also excludes a large number of records for which market share was near zero.

One can see how the knowledge contained in this pattern, or at least similar knowledge, might have been represented as a rule pattern, with a right hand side selecting the high end of the segment D market share distribution. Indeed, in many cases, similar rules emerged from both explicit rule and distribution shift patterns, linking particular features of the data to unexpectedly high or low sales or market share.

As a final example of distribution shift pattern, consider rule 4 which states simply that sales of Sierra per head population are higher in postal districts which contain at least one Ford dealer.

Distribution shift pattern 4

The distribution of "Sales Sierra per head population" when
Number of Ford Dealers > 0
(Approximate percentiles 62% - 100%)
(true: 936 false: 1621 unique false: 1621)

has median 0.00036 and is significantly shifted from the
distribution in the database as a whole which has median 0.00026.

5.2 Results on Classification Problems

GA-MINER is not specifically designed for classification problems but rather as a more general pattern search tool. We found it useful, however, to use GA-MINER for classification purposes during the process of refining the genetic operators, as we believe that successful classification is a prerequisite for a successful general data mining tool. This is not to say that a general tool such as GA-MINER should outperform more specialised classification methods, but they must at least give results which are comparable in quality.

GA-MINER may be used for classification by constraining the right hand side of rules to a single term involving the designated class variable, giving rules of the form "**if** C **then** Class $= c_i$". Of course, to form a true classification system, a firing mechanism is required to apply the rules to the target data. Also, in the absence of population mechanisms such as those used in COGIN and REGAL (see section 3) to encourage the population to develop co-operatively to solve the overall classification task, we are likely to develop a set of rules which do not fully cover the database or the possible classes, that is, there would be some records which do not match any of the classification rules and some classes for which there are no rules. Although it is relatively easy to add such mechanisms to GA-MINER in the form of new RPL2 Rule library operators, this was seen a diversion from our main purpose, which was to aid us in the refinement of the genetic operators. We chose instead to evolve rules for each class c_i separately, thus ensuring a rule existed for each class, and to apply those rules using a simple firing mechanism which selected the most accurate matching rule.

As an example of a classification problem which was considered, we present some results obtained on the soya bean dataset, available from the machine learning repository at UCI, where the task is to identify soya bean diseases. The training data consists of some 307 records and 35 attributes. However, there is some controversy over the last 17 records in the database, which it is argued, do not provide large enough samples of their respective classes to construct a sensible model. We therefore eliminated records in these classes from both the training and test data.

The genetic algorithm used a 10×10 fine-grained population model, and was run for 100 generations on each class.

Two example classification rules are reproduced below:

```
if      leaf-mild in {1}
        (true: 10 false: 280 unique false: 280)
then
        disease = powdery-mildew
        (true: 10 false: 280 unique false: 280)
Left hand side matches 3% of the database
Right hand side matches 3% of the database

              Expected   Actual
Accuracy:        3%       100%
Coverage:        3%       100%
```

```
if      fruit-pods in {3, <missing-value>}
        (true: 50 false: 240 unique false: 223)
and
        leaves in {1}
        (true: 263 false: 27 unique false: 10)
then
        disease = phytophthora-rot
        (true: 40 false: 250 unique false: 250)
Left hand side matches 14% of the database
Right hand side matches 14% of the database

              Expected   Actual
Accuracy:       14%       100%
Coverage:       14%       100%
```

While GA-MINER did not outperform specialist classification systems on the soy-bean data, our results of around 80–90% accuracy were at least comparable to those reported in the UCI repository of around 95%.

51140 records, 58 fields, 25 Mbytes of data Population size 400, 100 generations		
Number of Processors	Mean runtime (minutes)	Speedup
1	507	—
2	269	1.88
4	135	3.75
8	77	6.58

Table 5.1: Performance of GA-MINER on the Challenge XL.

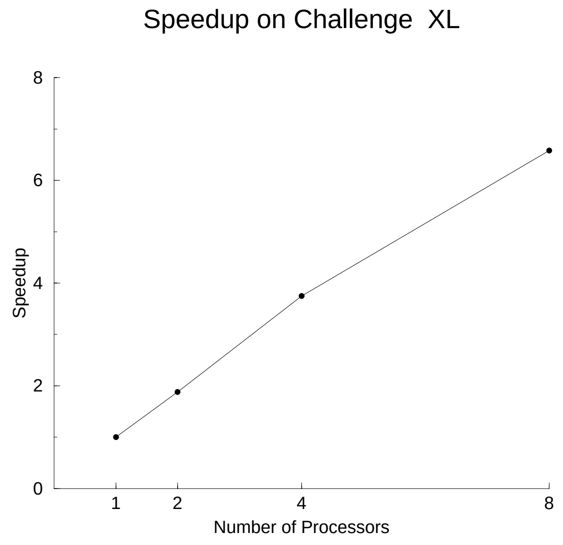


Figure 5.2: Speedup on the Challenge XL

5.3 System Performance and Scalability

In order to determine the scalability of the GA-MINER application as more processors are made available, we undertook some performance tests on an eight processor Silicon Graphics Challenge XL (representing SMP platforms) and a Cray T3D (representing MPP platforms).

The database used for the Challenge XL consisted of 51140 records and 58 fields, amounting to some 25 Mbytes of data. The population consisted of 400 members in a fine-grained structure of dimension 20×20 and was parallelised using a regular domain decomposition. The tests were performed in competition with other users, so some adverse effects may have been experienced. These do not appear to have seriously affected results however, as the variance in runtimes was low over some 7 runs.

The speedup graph for 100 generations of the algorithm on 1, 2, 4 and 8 processors respectively is shown in figure 5.2. We did not perform any optimisation to the code, since the main aim of the exercise was to determine scalability, so runtimes are somewhat higher than might be expected in a final optimised version of the code. It can be seen from these graphs that the scaling behaviour is relatively good, with 8 processors achieving a 6.58 times speedup and the tail-off being relatively slow.

On the Cray T3D it was unfortunately not possible to run this same large job because of a system administration policy which limits to one hour, the time allocated to jobs using a small number of processors. However, we did run several tests on the smaller GMAP database of 2557 records, using a fine-grained population structure of dimension 10×10 and 20×20 respectively. These results are shown in table 5.2 and the speedup shown in figure 5.3. Once again, we see that good speedup is achieved up to around eight nodes, and that further speedup is achieved on 16 and 32 nodes respectively, particularly as the popula-

2557 records, 58 fields Population size 100, 20 generations			2557 records, 58 fields Population size 400, 20 generations		
Number of Processors	Runtime (seconds)	Speedup	Number of Processors	Runtime (seconds)	Speedup
1	1570	—	1	6050	—
2	790	1.99	2	3210	1.88
4	430	3.65	4	1730	3.50
8	210	7.48	8	740	7.20
16	180	8.72	16	490	12.35
32	150	10.47	32	330	18.33

Table 5.2: Performance of GA-MINER on the Cray T3D.

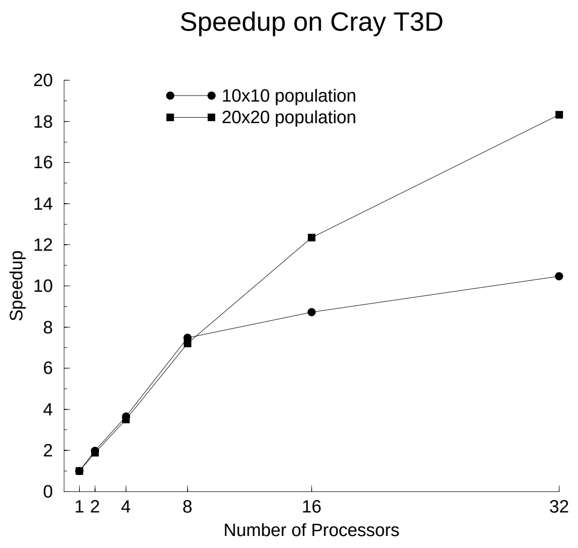


Figure 5.3: Speedup on the Cray T3D

tion size is increased since this will result in an increase in the ratio of computation with respect to the overheads of I/O and inter-process communication.

The evaluation of a pattern against the database is by far the largest processing stage in the algorithm. This involves a full pass over the database, gathering statistics and applying the conditions in each subset description to each record to determine its inclusion or otherwise in a particular set. Upon completion of this pass, the chosen evaluation function is called to compute the goodness of the pattern.

The runtime could be reduced by using lazy evaluation of the subset descriptions, returning `false` as soon as any term in a conjunction evaluates to `false` and similarly returning `true` as soon as any term in a disjunction evaluates to `true`. However, this would come at the expense of the statistics which are presently gathered for each clause in a rule and which are used by the pruning mechanism (see section 6.2).

The runtime could also be reduced by using a sample of the database, however as we mentioned in section 2.4.2, sampling can miss patterns which apply to only a small number of records, and a particular sample may not be representative of the database as a whole. The first of these problems is really inherent in the use of sampling, however, the second problem can be overcome to some degree by using a different sample upon each evaluation. This results in a noisy fitness function, that is, if a particular pattern were evaluated several times, it would appear to have varying fitness. However, one positive effect is that patterns which appear to be good in only a single freak sample are less likely to survive through successive generations. The final selected set of patterns can be evaluated against the whole database upon completion of the algorithm to determine their true goodness.

Chapter 6

Discussion

6.1 Evaluation Functions

In this section, we discuss the various evaluation functions which have been implemented in GA-MINER, and consider their relative merits in scoring patterns.

All of the evaluation functions are based entirely on mathematical properties of the patterns, such as generality, accuracy, coverage and statistical significance and do not include any qualitative measures such as rule "comprehensibility". Rather we have chosen to introduce these other factors of interest into the algorithm separately. For example, rule simplicity is encouraged by a pruning mechanism which removes redundant or near redundant terms (see section 6.2).

Despite this limitation to quantitative measures, the evaluation function remains of vital importance to the algorithm. It is important to obtain some level of familiarity with the kinds of rules which may be produced by different evaluation functions, in order that an appropriate one may be selected for the task in hand.

6.1.1 Comparison with Evaluation Functions in Decision Trees

The evaluation functions used for explicit rules in GA-MINER are all functions of $|C|$, $|P|$, $|C \cap P|$ and $|\overline{C} \cap \overline{P}|$ respectively, that is, the number of records falling into each of the four partitions of the database induced by the subset descriptions C and P. If we regard P as a target population to be modeled using a decision-tree (Quinlan, 1986, 1993; Breiman *et al.*, 1984), and C as defining a proposed split of the database into two subsets, then the evaluation functions used in GA-MINER can also be used in decision-tree applications to determine the goodness of the proposed split.

Decision trees can be interpreted as a set of rules which partitions the database into mutually disjoint subsets. One rule exists for each leaf node, and it is formed as the conjunction of the conditions which must be met as the tree is traversed along the unique path from the root to that leaf node. It is interesting, therefore, to consider the effects of evaluation functions in decision tree methods with their effects in GA-MINER.

Simple decision tree methods often produce the same or similar decision trees no matter what evaluation function is used to determine the utility of a particular split (Breiman *et al.*, 1984, Murthy & Salzberg, 1995). This is because the construction of the tree is both incremental and greedy (so the best split at each node is always used) and many evaluation functions will give identical results for the best split. If the decision trees generated by various split functions are near identical, then the rule sets formed will also be nearly identical.

As the complexity of the decision tree increases, perhaps by allowing non-binary splits or by permitting splits to be defined by a function over several attributes, the possibilities for differences between the trees generated by different split functions increase, though perhaps not by as much as one might expect, since the split resulting in the maximum value of the respective functions remains the only relevant factor.

In the GA-MINER approach, rules are formed in a more immediate and less systematic way than in decision tree approaches, or for that matter, in any other deterministic rule building approach. That is, rules are not incrementally generated one term at a time. Rather, they may initially be randomly generated and are then modified through relatively disruptive crossover and mutation operators. This means that many rules are evaluated which would never even have been generated by more deterministic methods, resulting in a more extensive, albeit less directed search of the space.

Although this more widespread search can not be attributed directly to the evaluation function, it is clear that the evaluation function plays a more important rôle than for decision trees, since in GA-MINER, the whole range of function values are relevant to the algorithm and the complex population dynamics will lead to more unpredictable results.

6.1.2 Evaluation Functions for Explicit Rules

GA-MINER provides several evaluation functions for explicit rule patterns, the details of which are described in section 4.5. Here we attempt to draw some comparisons and general conclusions on the relative merits of these functions.

It is important to note that rank-based selection and replacement mechanisms are employed throughout GA-MINER, so only the rank ordering of rules is important.

In general, we found that it was not possible to conclude that any of the evaluation functions produced *better* rules than the others. Indeed, many of the evaluation functions were found to be similar in their rank ordering.

6.1.2.1 Similarity of the Evaluation Measures

We performed a number of experiments which exhaustively considered all possible contingency tables which could be formed from a set of N data records[1]. Essentially this involved generating all possible contingency tables which satisfied the following constraints.

1. $0 \leq |C| \leq N$
2. $0 \leq |P| \leq N$
3. $|C \cap P| \geq |C| + |P| - N$
4. $|C \cap P| \leq \min(|C|, |P|)$

Each table was then evaluated using the various evaluation functions and the results were compared on scatter plots. The results strongly suggest that for a 2×2 table, the rank ordering of many of the evaluation functions is similar.

In particular, information content, the twoing value and χ^2 (with or without Yates correction) were all strongly correlated, and had high rank agreement.

This group of evaluation functions were also strongly correlated with the absolute value of rule interest and with the absolute value of the product moment correlation evaluation function. These two functions are uni-directional in the sense that they do not score rules of the form "**if** C **then** $\overline{P}$" highly. However, they do assign to such rules a negative fitness equal in magnitude to that which would be accredited if P and $\overline{P}$ were interchanged. Thus the absolute value of these functions can be used to detect any association between C and P

[1] In our experiments, N=50.

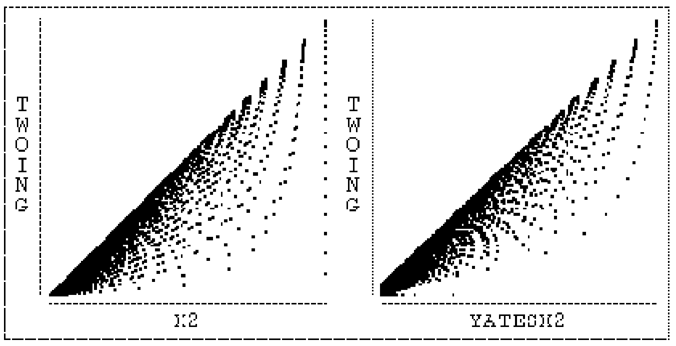


Figure 6.1: Scatter plot of twoing value against χ^2 and Yates χ^2

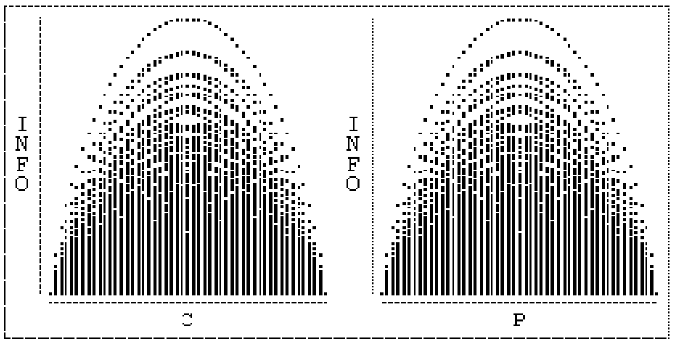


Figure 6.2: Scatter plot of information gain against C and P

The linear and rank correlations between these evaluation functions were all high (in the region of 0.95). However these figures do not present the whole picture. A scatter plot of the various measures is more revealing of the subtle relationships between the functions. For example, figure 6.1 shows the relationship between the twoing value, χ^2 and Yates χ^2 evaluation functions. The points in the lower right hand corner which are scored highly by χ^2 but rather less well by twoing value, correspond to those contingency tables which contain a number of records close to zero in any of their entries. It is widely known that the χ^2 tends to break down for such small quantities. However it is interesting to note that the χ^2 with Yates correction improves the agreement to some extent.

The χ^2 generally tends to assign higher fitness to large subsets C and P, which result in relatively small deviations from the expected cell counts. This is because statistical significance depends critically on the sample size (Elder & Pregibon, 1995) and even these small deviations attain high statistical significance. As a result, clauses in rules generated by the χ^2 tend to be relatively short in length, giving simpler rules than some other measures without the need for pruning. This is likely to be a side-effect of the tendency to select large subsets, which more easily achieved when the the number of terms is small. On the negative side, the rules generated by χ^2 can often be over-general.

The symmetries of the various measures are also revealing. The information gain, twoing value, and absolute value of rule interest are all symmetric in C and P and are all very similar (see for example, information gain plotted against C and P in figure 6.2). These indicate that subsets C and P selecting approximately half of the records in the database have the potential to achieve the maximum fitness.

The χ^2 is also symmetric in C and P (see figure 6.3), but it differs from the previous evaluation functions in that subsets of all sizes have the potential to achieve maximum fitness. Thus, the χ^2 does not differentiate between contingency tables with say $|C| = |P| = |C \cap P| = 1$ and $|C| = |P| = |C \cap P| = 20$. The absolute value of linear correlation also has a similarly shaped scatter plot to the χ^2 and shares this property. Indeed, χ^2 and the absolute value of linear correlation have perfect rank correlation and for our purposes are therefore equivalent. Note also that the χ^2 with Yates correction modifies the symmetries to look more like that for information gain.

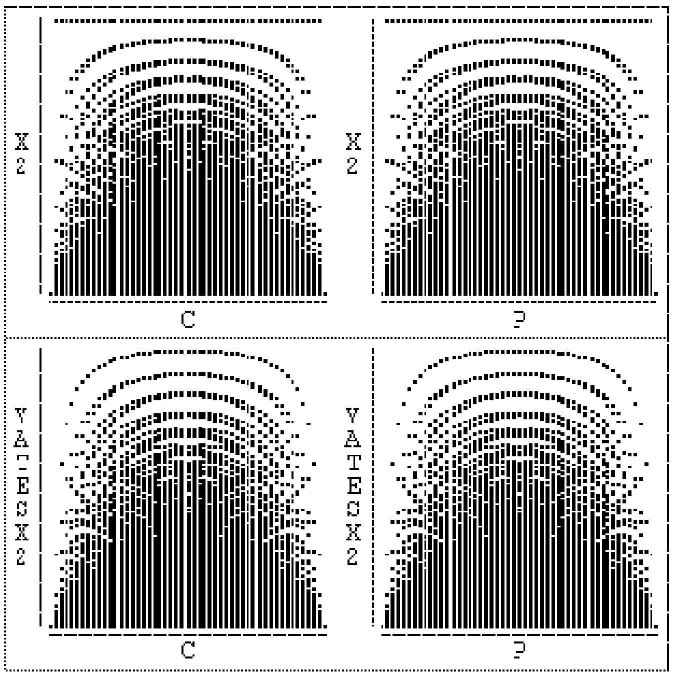


Figure 6.3: Scatter plot of χ^2 and Yates χ^2 against C and P

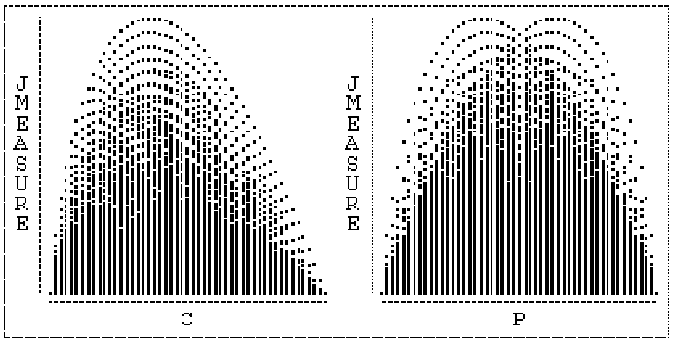


Figure 6.4: Scatter plot of J-measure against C and P

The J-Measure is perhaps the most different of the evaluation functions examined. J-Measure is not symmetric in $|C|$ and $|\overline{C}|$ and is bi-modal when plotted against P (see figure 6.4). In general, J-measure tends to slightly favour C subsets of a size slightly less then half of the total number of records. Similarly, P subsets of size slightly less than or slightly greater than half of the total number of records are favoured. So for example, in a database of 50 records, a rule with $|C| = |P| = |C \cap P| = 18$ will score slightly better than a rule with $|C| = |P| = |C \cap P| = 25$.

6.1.3 Evaluation Functions to Detect Shifts in a Distribution

Detection of differences in the distribution of a variable between particular subsets of a database can provide useful knowledge. For example, if the variable $sales$ is generally higher within a particular subset selected by the clause $price < 1.00$, then one might suspect that the lower price is having a positive effect on sales. Clearly, the concept “generally higher” is not the only assertion which could be useful. The terms “generally lower” or even just “different” could also be useful in different circumstances.

To make such statements more mathematically concrete, we must first decide upon what we are actually going to compare, and within which subsets (or populations).

One option is to compare the mean value of a selected variable within two populations. Student's t-test has been traditionally used as a test for comparing two means (see section 4.6). However, this test assumes that the distributions being compared are normal, and although it is relatively robust for small deviations away from this assumption, some of the subsets generated by GA-MINER will significantly depart from this assumption.

The central limit theorem tells us that when the subsets are large, the mean value within each will be approximately normally distributed regardless of the underlying distribution, allowing the use of a simpler test statistic. This method is used in GA-MINER together with a lower threshold on the size of subsets used, preventing inaccurate results for small population sizes.

In general we would expect comparison of means to suffer from a lack of robustness, that is, outliers in a distribution may seriously skew the calculation of the mean, perhaps giving misleading results. In our experiments, we did find some evidence that skewing was indeed occurring. However, many of the patterns discovered were still regarded as useful and the evaluation function was not irretrievable affected by this problem.

In cases where the mean comparison is badly by outliers, an alternative is to use a non-parametric test statistic based on the rank ordering of a variable. These are generally more robust, since the actual values of the variable are not important, tempering the effect of severe outliers. One such test, which is implemented in GA-MINER, is the Wilcoxen-Mann-Whitney test (4.6.3) which compares the median value in two populations. In our experience, we did find evidence that this statistic was more robust that a mean comparison, and in general it produced slightly better results. However, it did take longer to evaluate each solution, given the need to perform a sort to rank the records in order.

Yet another alternative is the Kolmogorov-Smirnov test. Unlike mean and median comparisons, this test is not location-based. Rather it is a more general distribution comparison, whose test statistic is the maximum difference between the cumulative distributions of the two populations. So for example, a symmetric bimodal distribution about 0, which will not show a mean or median shift when compared with a normal distribution about 0, will be detected by the Kolmogorov-Smirnov test. However, in general, the power of this test is weaker than the others when distributions are close to normal and unimodal, and in most cases, detecting a general difference in two distributions is perhaps less useful and more difficult to interpret than detecting a shift. For these reasons, the Kolmogorov-Smirnov test statistic has not been implemented in GA-MINER.

There are several options regarding the subsets within which variables may be compared. One possibility is to compare the distribution of the variable in a selected subset with its distribution over the database as a whole (i.e. a comparison of the distribution within sets S and $S \cap C$ respectively). In this case, records in the selected subset $S \cap C$ are counted as members of both populations. A second alternative is to compare the distribution of the variable in the selected subset with its distribution over records not in the subset (i.e. a comparison of the distribution within sets $S \cap \overline{C}$ and $S \cap C$ respectively). Either of these two options seems reasonable, though for the distribution patterns in GA-MINER, the first alternative is followed.

6.1.4 Evaluation Functions for Correlation Patterns

GA-MINER includes patterns to detect both linear and rank correlations between a pair of variables within a subset of the database.

Linear correlation attempts to fit a line $y = \beta x + \alpha$ to the data, where y is a dependent variable, x is an explanatory variable and α and β are constants. Perhaps the simplest evaluation function for linear correlation is the product moment correlation coefficient (see section 4.7) which gives a value ranging from -1 (perfect negative linear correlation) through 0 (no linear correlation) to 1 (perfect linear correlation).

However, this evaluation function tends to be badly affected by outliers, takes no consideration of the statistical significance of the discovered correlation and includes no encouragement to find larger subsets.

In practice we found that this measure tended to pick out small subsets of variables, often containing an outlier which resulted in an artificially high fitness.

The evaluation function may be modified a multiplicative factor $\mathcal{P}(C)$ to encourage larger subsets. We did find that this helped to some degree, but the selection of the multiplicative factor was crucial to control the balance the between subset size and correlations accepted as reasonable.

Another alternative is to use a statistical significance measure to evaluate the evidence against the slope of the best fit line being zero, which would be the case if there were no linear association between the two selected variables. Such a t-statistic for this purpose is given in section 4.7. This measure also increases with the subset size, providing an incentive to find larger subsets. However, once again, we often found that large samples (i.e. large subsets C) tended to give high significance to slight correlations.

The Spearman rank correlation coefficient between two variables is a measure of the degree of monotonicity between the variables, taking values between -1 (y increases or says the same as x decreases), through 0 (no monotonicity) to 1 (y increase or stays the same as x increases). Note that two perfectly linearly correlated variables will also show perfect monotonicity or inverse monotonicity. This is not surprising, since the Spearman rank correlation coefficient is just the product moment correlation between the rank ordering of the variables, but since it does not use the explicit values for each variable it is more robust against outliers.

The evaluation functions used for rank correlation closely mirrored those used for linear correlation. In general the rank correlation fitness functions showed similar tendencies to their corresponding linear correlation counterparts.

Our results with the correlation patterns on real world databases were in general disappointing. However, it may be that these databases simply did not contain relationships of this form, as we were repeatedly able to find planted correlations between variables within subsets in artificially generated datasets. Whether or not such patterns are typical in the real world requires further investigation.

6.2 Pruning of Rules

Rather than include the notion of rule simplicity in the evaluation function, GA-MINER uses a runtime pruning mechanism to encourage rules with fewer terms. The pruning mechanism is also used to remove redundancy in the rule representations.

When rules are evaluated, a count of the number of data records satisfying each term is maintained, together with a count of the number of times a single term alone evaluated false and hence caused the entire clause to evaluate false (that is, the number of times the term was *uniquely false*).

If a term was uniquely false zero times, then it is essentially redundant, as it was never the unique cause of a failure in the overall condition. Such terms may be pruned without detrimental affect to the evaluated fitness, provided only a single such term exists in a clause. In such cases, we chose to remove these redundant clauses with a probability of one.

If more than one uniquely false term appears in a clause, then only one of them may be removed immediately, since deletion of this term invalidates all the evaluation statistics gathered. In cases where more than one redundant term appears in a clause, we chose to select one for removal at random.

Near redundant clauses (that is, clauses which were close to being uniquely false) are pruned with a probability based on how close the uniquely false count was to zero.

A runtime pruning mechanism does have associated dangers, since it will remove genetic material from the population and reduce diversity. However, the alternative, which is to allow possibly very complex rules to form within the population and perform a post-pruning process, also has disadvantages. For example, in this case, there may be much “excess baggage” in a rule, with many overlapping and redundant terms, making it unclear which of these terms is contributing to the overall goodness of the rule.

The runtime pruning mechanism employed adds a slight pressure for shorter, simpler rules, while still attempting to prolong the life of useful genetic material in the population. Clearly, post-pruning may still be applied if thought necessary.

6.3 Assigning Significance Levels to Patterns

In GA-MINER, we have refrained from assigning an explicit level of significance to patterns and have instead used the respective test statistics as a measure of the relative goodness of patterns. This is because although hypothesis tests, such as the t-test and chi-square test are widely used to assign significance levels to patterns in data, such significance levels are only valid if the assumptions of the tests have been met. Whereas in controlled statistical experiments it is usually possible to ensure that this is the case (at least to a reasonable degree) in data mining we invariably find that test assumptions are violated.

For example, consider perhaps the most general assumption of hypothesis tests, which is that only a single test was performed on the data. In data mining hundreds or even thousands of hypothesis tests may be repeatedly performed on the data during the search. Significance levels obtained therefore overestimate the true significance of the discovered patterns. It has been noted that Bonferonni adjustments may be made to take into consideration the number of tests performed (Piatetsky-Shapiro *et al.*, 1993), however, the patterns being tested are often highly correlated, so the assumptions of this adjustment method are also violated, resulting in the true significance level of patterns being badly underestimated.

Jensen (1991) has suggested the use of randomisation testing to obtain significance levels. Jensen's application domain requires that a set of candidate classification rules be compared to determine whether any of them are better than the current best known solution. The candidate solutions are repeatedly evaluated on randomised data which is carefully constructed to maintain the classification accuracy of the current best known rule, but results in different records being mis-classifying. For each randomised data set, the best classification accuracy obtained by any of the candidate rules is recorded and is used to construct a distribution of best scores. The score of the best competitor on the true data is then compared to this distribution, and the percentage of the distribution falling below this level is an estimate of the probability that the best candidate solution is better than current best known rule on the test data.

6.4 Genetic Algorithms as a Data Mining Search Tool

In the light of our experiences during the development of GA-MINER, it is worthwhile considering both the merits and disadvantages of a genetic-based data mining system.

Perhaps the greatest strength of the GA-approach is its ability to find patterns in very large search spaces with little or no required background knowledge. The nature of genetic algorithms is such that the areas of the pattern space which are subject to the most intensive search will be those which appear to match the actual data. In this respect, the genetic algorithm is self configuring. The way in which patterns are formed through crossover and mutation also makes it possible to search a larger number of complex patterns than may normally be the case, since many greedy and heuristic methods must restrict the search to a limited number of terms or clauses in order to make the problem tractable. Genetic techniques appear therefore to be particularly suitable for databases containing many fields, where the number of possible patterns is large and other methods would require heavy pruning which could be detrimental to their effectiveness.

Genetic algorithms also appear well suited to undirected data mining, given their limited need for user direction and user interaction. The genetic algorithm process is stochastic in nature and therefore several runs of the same algorithm using a different seed for the random number generator can generate completely different rules. [2] This can often be advantageous, as subsequent runs of the algorithm offer the

[2] The random number generator included in RPL2 will generate the same sequence of numbers on all platforms, given the same initial seed. This usefully allows all results to be reproduced when necessary, provided all other parameters to the algorithm are restored to their original values.

possibility of discovering new patterns. So for example, a genetic-based data mining tool could be used to search unsupervised through a large database every night during quiet load times.

Genetic-based methods have also been demonstrated to be highly congruous to parallel implementation, providing a scalable system which can exploit the power of multiprocessing machines. A further advantage of genetic algorithms, and indeed of many of the heuristic approaches, over some other techniques such as neural networks, is that the patterns produced are directly comprehensible and understandable. This has clear advantages in terms of the usability of the system and improves the ease with which patterns may be interpreted.

Pattern templates were found to be extremely useful for the more directed data mining applications, guiding the system towards patterns of the desired form. Scrutability of patterns is again important here, as users must be able to describe the form of pattern which is required. An extension of these templates to allow further constraints to be placed on patterns, and to allow some degree of fuzzyness and departure from the template would be extremely useful.

There are still some aspects of the genetic algorithm method which we regard as problematic. Although the introduction of domain knowledge in the form of local search can be extremely productive, it is computationally expensive to re-evaluate each newly modified solution on the entire database. Clearly, the scalability of the system will not be badly affected by local search, however the wall clock time which an application takes to run on a single processor will inevitably increase substantially. Methods to reduce this overhead, such as the use of sampling when evaluating the modified patterns, suffer some drawbacks such as reduced accuracy or noisy fitness functions as discussed in section 5.3. A useful compromise would be to stop short of applying the local search algorithm repeatedly until a local optima was reached, and rather terminate the local search after making some limited but useful improvement to the solution.

A second problem with the genetic algorithm approach is that although repeated runs have the potential to produce different solutions, we have no way at present of forcing it to produce *only* new and previously undiscovered solutions. Ideally, we require some form of knowledge base which “remembers” those areas of the search space which have previously been well explored and which can be eliminated from consideration until other possibilities have been exhausted. It is as yet unclear how such a mechanism could be implemented efficiently.

6.5 Architectural Insights

Our experience throughout the GA-MINER project has been that data access, and more specifically the process of determining the number of records contained in a particular set (essentially a database selection operation) is the most expensive part of data mining and as such forms the biggest performance limitation on scalable data mining applications.

Provided data mining systems are content to use relatively small samples of data in the analysis process such that all data will fit within main memory, then data access will be relatively fast. In this case, I/O forms a more or less constant overhead, one read being required to bring the data into the memory from which all subsequent accesses are made. In this case, it is clear that the scalability of the application as a whole will be determined mainly by the scalability of its computational phase. In this respect, the form of parallelism most natural to genetic algorithms, that is, distribution of the population across nodes, works very well on both SMP and MPP platforms. In particular, the ability of MPP systems to scale well to hundreds of nodes for such compute-intensive applications makes them ideal in this scenario.

However, it is becoming increasingly apparent that commercial data mining systems will require to access volumes of data far in excess of available main memory. On an MPP system, one option is to parallelise the evaluation of every rule and to divide the data across processors. In this case, the inherent parallelism in genetic algorithms is not exploited. However, parallelisation is still clearly possible and is still likely to achieve relatively good performance, since I/O remains a constant overhead and inter-process communication is relatively light.

However, even the aggregate memory of all nodes on an MPP system are unlikely to be sufficient for some commercial databases. Thus, for large scale data mining systems, data access is likely to increasingly become a process of *disk* access.

This has some serious implications for data mining systems. Firstly, given the necessary and repeated access to the disk, it seems prudent to concentrate efforts on making disk access as efficient as possible. As the nature of the required access is highly specific, leaving the data in a general purpose database, accessed through SQL, is unlikely to give as good performance as storing the data directly on disk in whatever form is regarded as most efficient and directly accessing the data from this medium — unless of course the database supports such facilities internally, which we feel will increasingly be the case in future.

A question then arises as to the most appropriate architecture on which to run such a data mining application. Until very recently, MPP platforms were primarily developed for and used by the scientific community, for highly compute-intensive applications in which I/O played a relatively minor part. Perhaps as a result, I/O handling on MPP platforms is still relatively immature. Many such platforms, perform all I/O through the host machine, forming a bottleneck on data transfer between the host and parallel nodes and placing severe limitations on the scalability of I/O intensive applications. An alternative is to attach disks directly to several nodes and allow each processor direct access to this distributed file system. However, parallel file systems on MPP platforms are still uncommon, are relatively new technology, and have yet to prove themselves in a commercial environment.

On the other hand SMP platforms have traditionally been more commercially focussed. Their use in the commercial database market is widespread, and this has lead to a greater emphasis on fast disk access. File management is significantly simpler, requiring only that the data be moved or copied to a logical volume which has been spread across several physical disks. In addition, the ability of SMP platforms to utilise memory-mapped files offers substantial performance benefits. Efficient paging algorithms are available and data may be brought into memory in blocks, a form of pre-fetching which enables data to be made available in memory before it is accessed, and which allows the cache to be fully exploited.

Of course the amount of data which can be addressed by a single process is limited by the size of a system's address bus. MPP systems should therefore be able to address more data than an SMP system, since on the former each process can independently address its own data, while on the latter all threads share the same address space. However, the advent of 64-bit address buses substantially lowers the practical impact of this problem.

In the short to medium term we believe that SMP platforms are the most suitable architecture for large-scale parallel data mining, at least until the technology required for fast and efficient disk access and for integrated file management on MPP platforms is substantially more mature. In the longer term, effective parallel I/O systems will undoubtedly develop on MPP systems. It remains to be seen whether these advances will sufficiently outpace the development of SMP technology to usurp SMP's current leading rôle in the database market.

Chapter 7

Exploitation and Future Work

7.1 Commercial Exploitation of GA-MINER results

Quadstone Ltd is an independent software and consulting company staffed by former employees of the Edinburgh Parallel Computing Centre at the University of Edinburgh. The fundamental architectural and analytical discoveries from the GA-MINER project will be crucial to the second wave of functionality within Quadstone's Decisionhouse software — initially a suite of software applications for visualisation, interrogation and analysis of data in data warehouses. The project has also generated base input on data mining for the industrial partners GMAP and Barclays Bank. In particular, Barclays realise the vital importance of making full use of the masses of data that they hold on their customers. Barclays will be early adopters of Decisionhouse software, and are at the forefront of advanced commercial analysis of large volumes of data. Results from the GA-MINER project have enabled Quadstone to provide Barclays with substantial technical input to two internal reports — one on High Performance Computing and the other on Data Visualisation and Data Mining technology (see appendix C).

The worldwide data warehouse market is already substantial, and is predicted to grow rapidly in the near future. Some claims put the market worth at $5billion annually by 1997. Quadstone is already targeting all UK organisations with over one million customers, as these are the ideal first users for Decisionhouse. Initial exploration of the US and European markets is also underway, and Quadstone will gain much assistance in these areas from two recently agreed partnership alliances with Tandem Computers and Oracle.

Within two years Quadstone should have made the successful transition from leading academic expertise in parallel and high performance computing applications, to business success in the early supply of commercial high performance computing software. This transition is made possible as the business world is now starting to make real operational use of parallel data machines. The GA-MINER project has been instrumental in establishing Quadstone's position. It will also provide key technological insights to keep Quadstone in a leading position within the data warehouse marketplace.

7.2 Future Work

There are many aspects of GA-MINER which we believe can benefit from future research. In this section we consider briefly some of the work which we believe would be helpful to the development of genetic algorithm data mining tools and to GA-MINER in particular.

7.2.1 Domain Knowledge

GA-MINER's use of domain knowledge concerning the database undergoing analysis is currently limited. Indeed, only a basic description of the type of each of the database fields is used. Although other domain knowledge is included in the form of the chosen representation, the genetic operators and the evaluation function, this knowledge is relatively weak and is not specific to particular databases. On the contrary, it is concerned with the more general application domain of rule and pattern discovery.

In fact, it is one of the principle advantages of genetic algorithms that they may still function well, even when virtually no domain knowledge is available. However, genetic algorithms are no panacea and when domain knowledge is available, improved results can almost certainly be achieved by its utlisation.

There are several ways in which this can be achieved, including non-random initialisation of the population (Ray, 1992, 1994), which is already used by GA-MINER for hypothesis testing and refinement, incorporation of extra ("non-genetic") local move operators and "hybridisation" with domain-specific heuristics (Davis, 1991). We believe that hybridisation and local move operators hold particularly good promise, as our experience has shown it to be useful in other application domains (Radcliffe & Surry, 1994b).

Such extensions are likely to involve the inclusion of a rule- or pattern-refiner whose purpose is to locally improve a given rule or pattern by local search, possibly incorporating concept hierarchies (Piatetsky-Shapiro, 1992) and heuristics based on statistical properties of the database. The rule-refiner would be applied to children newly generated by crossover and mutation, or at least to a subset of these. Quadstone Ltd. is in fact in the process of developing such an automated rule-refiner.

7.2.2 Pattern Templates

During the course of this research, pattern templates have been found to be extremely useful in adapting GA-MINER to particular tasks or applications. Our experience suggests that such templates could play a useful rôle more generally in data mining systems. Indeed templates have been used successfully to guide the search for interesting association rules in large collections of discovered rules (Klemettinen *et al.*, 1994).

Some research is still required to determine which features of patterns should be configurable through templates, and on how such constraints should be specified. It may be, for example, that a pattern template can be generated to some extent "behind the scenes", being derived automatically from a user's query expressed in a more user-friendly language.

Current experience suggests that extensions to allow a degree of fuzzyness in patterns would be helpful, as presently patterns must conform strictly to the given template. Of course, all such extensions will require thought as to how the constraints expressed in the template can be efficiently implemented by the genetic operators or otherwise. It may also be prudent to provide templates which describe the form of non-interesting rules (Klemettinen *et al.*, 1994), as a means of guiding the discovery process away from known or uninteresting rules.

7.2.3 Rule Interest Measures

More work is required on the process of evaluating and selecting interesting rules. Most of GA-MINER's current evaluation functions rely too heavily on statistical significance measures. Other rule interest measures, such as simplicity, could be directly incorporated into evaluation functions. Alternatively, the trade-offs between multiple goodness criteria could be handled using the notion of Pareto-optimality (Fonseca & Fleming, 1993, 1994; Surry *et al.*, 1995).

The present use of a runtime pruning mechanism to encourage rule simplicity requires further investigation to determine whether it may be improved. Furthermore, although the simple distance measures and heuristics used to gather a subset of the generated rules have been effective for the designated task, more sophisticated rule interest measures are likely to require that this scheme be modified.

Finally, GA-MINER does not yet consider subjective measures of rule interest, which are likely to become of increasing importance as data mining systems become more sophisticated. Research into such measures and consideration of how they may be integrated with GA-MINER will be necessary if it is to be used by non-technical users.

7.2.4 User-Interaction and Presentation of Rules

The whole question user-interaction and rule presentation forms a subject in itself and it has not been possible during the course of this research to fully explore this area. It is clear that data mining systems can benefit greatly from general research into both *Human Computer Interaction* (HCI) and data visualisation. However, requiring particular consideration from the GA-MINER perspective are the questions of how users specify pattern templates, or more generally, how users specify areas of interests to the system. Of course, the display of rules and patterns in a clear, concise and illuminating way is also of vital importance, and this could benefit from collaborative research with visualisation experts.

Appendix A

Example RPL2 Plan

```
plan(Miner)

use Rule, StdInst, Debug;

structure [10:fine, 10:fine] deme Euclidean(1.0);

gstack [*,*] gs;
genome [*,*] g, gKid, gMum, gDad;

genome gBest;
genome gNext;
gstack gsAll;

bool bMaxIsBest;
bool bUseRawFitness;
bool bFuzzyVal;
bool bFuzzyMin;
bool bFuzzyMax;

int i;
int j;
int k;
int nGen;
int iCand;
int nCand;

int nPercentiles;

int evalFn;
int count;
int nS;

int nMinSizeTT;
int nMinSizeFF;

real rBias;
real rPUCross;
real rPdClsCross;
real rAccBias;

real rPAddDelCls;
real rPAddDelTerm;
real rPMutTerm;
real rPMutAttr;
real rPMutValue;
real rPMutConst;
real rConstHalfRange;
int  nConstSteps;
int  nMaxConstCreep;

real rPMutHypAttr;
real rPMutFuzzMin;
real rPMutFuzzMax;
real rFuzzHalfRange;
int  nFuzzSteps;
int  nMaxFuzzCreep;
real rPMutValueFuzz;
real rBeta;

real rSim;

int  stmtSetSize;

nMinSizeTT := 1;
nMinSizeFF := 1;

evalFn := 11; %J-measure

rBias := 0.5;                   % bias for xover

nPercentiles := 50;             % Number of percentiles to calculate


rPUCross := 0.5;                % P of using uniform crossover
rPdClsCross := 0.1;
rAccBias := 0.5;

rPAddDelCls := 0.10;            % P of adding or deleting a clause
rPAddDelTerm := 0.01;           % P of adding or deleting a term
rPMutHypAttr := 0.1;            % P of mutating a hypothesis attribute
rPMutTerm := 0.10;                % P of mutating a term in the rule
        rPMutAttr := 0.0;       % P of changing each of two field attrs
        rPMutConst := 1.0;      % P of creeping each const in rule
        nMaxConstCreep := 150;  % max # of creep steps for consts
        rPMutFuzzMin := 0.05;
        rPMutFuzzMax := 0.05;
        rPMutValue := 0.5;      % P of mutating a value in a value cst
        rPMutValueFuzz := 0.8;  % P of mutating the fuzzyness of a value
rConstHalfRange := 0.11;        % half range of normalised constants
nConstSteps := 20;              % number of discrete steps in the half range

rBeta := 20;

bFuzzyVal := FALSE;
bFuzzyMin := FALSE;
bFuzzyMax := FALSE;

rFuzzHalfRange := 0.1;
nFuzzSteps := 10;
nMaxFuzzCreep := 50;

bMaxIsBest := TRUE;
bUseRawFitness := TRUE;
nGen := 300;
nCand := 1;

stmtSetSize := 5;
rSim := 0.45;

SetParams(rAccBias, nMinSizeTT, nMinSizeFF, nPercentiles, bFuzzyVal,
          bFuzzyMin, bFuzzyMax, rPMutFuzzMin, rPMutFuzzMax, rFuzzHalfRange,
          nFuzzSteps, nMaxFuzzCreep, rPMutValueFuzz);

ReadData("/home/ga-miner/data/gmap/census91BDFH", "censusSet.template");

RuleSetInit(stmtSetSize);

for iCand := 1 to nCand

%=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=#

structfor [*,*]
        i := 234351 * iCand;
        Randomize(i);
        g := RandomGenome();
        EvalRule(g, evalFn, rBeta);
endstructfor

gBest := ReduceRawBest(g,bMaxIsBest);
PrintGenome(gBest);

Empty(gsAll);
Collect(g, gsAll);
StatsPrint(0,0,gsAll,"stdout");

count := 0;
for i := 1 to nGen
        structfor [*,*]
                Empty(gs);
                DemeCollect(gs, g);

                gMum := SelectRawTournament(gs, bMaxIsBest, 2, 0.6, TRUE);

                gKid := Cross(gMum, g, rBias, rPUCross, rPdClsCross);

                Mutate(gKid, rPAddDelCls, rPAddDelTerm, rPMutHypAttr,
                       rPMutTerm, rPMutAttr, rPMutValue, rPMutConst,
                       rConstHalfRange, nConstSteps, nMaxConstCreep);

                EvalRule(gKid, evalFn, rBeta);

                Empty(gs);
                Push(g,gs);
                Push(gKid,gs);
                g := SelectRawTournament(gs,bMaxIsBest, 2, 0.8, FALSE);

        endstructfor

        Empty(gsAll);
        Collect(g, gsAll);

        gBest := ReduceRawBest(g,bMaxIsBest);
        for j := 1 to 100
                gNext := SelectNth(gsAll, j);
                RuleSetUpdate(gNext, rSim, bMaxIsBest, bUseRawFitness);
        endfor

        PrintString("************* Current Statement set");
        RuleSetPrint();

        PrintString("************* Best in population");
        PrintGenome(gBest);

        PrintString("************************************");

        StatsPrint(i,1,gsAll,"stdout");

endfor

%=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=#

endfor

endplan

run
```

Appendix B

Example Pattern Template Description

```
use Rule pattern;

dependent "Sales Fiesta per head population",
          "Sales Sierra per head population",
          "Sales Granada per head population",
          "Sales Escort per head population",
          "Ford market share (Fiesta)",
          "Ford market share (Escort)",
          "Ford market share (Sierra)",
          "Ford market share (Granada)";

begin(spec)

        MaxClauses = 1;
        MaxTerms = 6;
        MaxValues = 5;

        select variables from all independent except "Postal district",
        "XCoord", "YCoord";

        (true):fixed;

end(spec);

begin(cond)

        MaxClauses = 1;
        MaxTerms = 6;
        MaxValues = 5;

        select variables from all independent except "Postal district",
        "XCoord", "YCoord";

        (true);

end(cond);

begin(pred)

        MaxClauses = 1;
        MaxTerms = 1;
        MaxValues = 1;

        (
                (
                        (fix "Ford market share (Fiesta)"
                                = fix 0.2 .. fix 100.0) :fixed
                ):fixed
        ):fixed;

end(pred);
```

Appendix C

Data Mining at Barclays

Barclays is continuing to exploit results from GA-MINER and other research, and is presently conducting a study into data mining. A letter from Phil Brown of Barclays, outlining some preliminary results from this study is included below.

Dear Nick,

Thank you for providing input to our study into:

- the current status of data mining technology and its likely evolutions over the next 2-3 years,
- the current and planned data mining activity within Barclays, and
- the obstacles to the Bank's exploitation of the technology.

As you know, this study has taken input from a number of external sources such as yourselves and from most of the business areas with either a current or planned data mining activity. Our report is not yet completed but I thought that some of the provisional findings are particularly relevant to the AIKMS GA-MINER project:

Discovered relationships must be understandable One of the aims of data mining is the creation of new understanding of implicit relationships within data — to do this requires that the problem owner / analyst can understand the rule / relationship and that contribution made by the various factors. This builds confidence that the rule is plausible and not the result of 'spurious correlation'

A number of commercially available data mining tools already address this point.

Both top-down and bottom-up initiatives are required There is a need for data mining to allow

- relationships / proposed hypotheses to be automatically derived from data bottom-up,
- relationships / hypothesis proposed by the user to be tested, and
- a mixed initiative, where the users' tighten, relax or replace specific elements of a discovered relationship and let the computer search once again.

It is reputed that a few US tools address this point — I cannot say that we have seen it mentioned in the information we have gathered.

The types of 'discoverable' relationships are limited Current data mining tools are somewhat restricted in the variety of patterns/relationships they can discover especially in the undirected type of data mining. This seems to arise from them all using essentially the same set of proven algorithms — all of which have sound theoretical grounding. In contrast, the concept of using genetic algorithms and ‘relationship templates’ such as the ‘median shift’ seem to be a far more heuristic approach based less on mathematics and more on the sorts of patterns that analysts ‘look’ for by manual inspection.

As far as we have seem, no vendors currently seem to be offering this capability.

Phil Brown
GIST
Barclays

Appendix D

Data provided by GMAP

Field Name	Description
POSTDIST	Postal district
XCOORD	Grid reference (X) of postal district
YCOORD	Grid reference (Y) of postal district
TOTPOPM	Proportion of males
TOTPOPF	Proportion of females
POP0_4	Proportion of the population aged 0-4
POP5_15	Proportion of the population aged 5-15
POP16_24	Proportion of the population aged 16-24
POP25_44	Proportion of the population aged 25-44
POP45_RT	Proportion of the population aged 45-64
POPRT_PLUS	Proportion of the population aged 65 and over
SEG_A	Proportion of households with head in Social Class A (Professional occupations)
SEG_B	Proportion of households with head in Social Class B (Managerial occupations)
SEG_AB	Proportion of households with head in Social Class A or B
SEG_C1	Proportion of households with head in Social Class C1 (Clerical occupations)
SEG_C2	Proportion of households with head in Social Class C2 (Skilled manual occupations)
SEG_D	Proportion of households with head in Social Class D (Semi-skilled occupations)
SEG_E	Proportion of households with head in Social Class E (Unskilled occupations)
SEG_DE	Proportion of households with head in Social Class D or E
ECON_ACT	Economically active heads per household
OWNER_OCC	Proportion of households - owner-occupied
COUNCIL	Proportion of households - council-rented
PRIV_RENT	Proportion of households - private rented
DETACH	Proportion of households - detached
SEMID	Proportion of households - semi-detached
TERRAC	Proportion of households - terraced
FLAT	Proportion of households - flats
OTHER_RESI	Proportion of households - other permanent
NPERM	Proportion of households - non-permanent
CAR0	Proportion of households lacking a car
CAR1	Proportion of households with one car
CAR2	Proportion of households with two cars
CAR3	Proportion of households with three or more cars
CHILD0	Proportion of households with no children aged 0-15
CHILD1	Proportion of households with one child
CHILD2	Proportion of households with two children
CHILD3	Proportion of households with three or more children

WHITE	Proportion of the population - ethnic class 'white'
NEW_COMMON	Proportion of the population - 'New Commonwealth'
OTHER_PERS	Proportion of the population - others
STUDENTS	Proportion of Students
UNEMPLOYED	Proportion of Unemployed
QHDEG	Proportion of the population with higher education qualifications
QDEG	Proportion of the population with degrees
FDIST	Distance to nearest Ford dealer (from center of postal district)
ODIST	Distance to nearest non-Ford dealer (from center of postal district)
NFORD	Number of Ford dealers in postal district
NOTHER	Number of non-Ford dealers in postal district
BSALES	Ford Sales in segment B (Fiesta)
DSALES	Ford Sales in segment D (Escort)
FSALES	Ford Sales in segment F (Sierra)
HSALES	Ford Sales in segment H (Granada)
BSHARE	Ford market share in segment B (Fiesta)
DSHARE	Ford market share in segment D (Escort)
FSHARE	Ford market share in segment F (Sierra)
HSHARE	Ford market share in segment H (Granada)

Appendix E

Example Patterns Found by GA-MINER

E.1 Patterns found in the GMAP Data

if Proportion of population aged 5-15 $\geq$ 0.12
(Approximate percentiles 22% - 100%)
(true: 1980 false: 577 unique false: 120)
and
Proportion of households with 0 cars $\geq$ 0.33
(Approximate percentiles 64% - 100%)
(true: 889 false: 1668 unique false: 313)
and
Prop. head household in socio-economic group A or B $\leq$ 0.21
(Approximate percentiles 0% - 46%)
(true: 1148 false: 1409 unique false: 78)
then
Ford market share segment D (Escort) $\geq$ 0.23
(Approximate percentiles 58% - 100%)
(true: 1043 false: 1514 unique false: 1514)
Left hand side matches 23% of the database
Right hand side matches 41% of the database

	Expected	Actual
Accuracy:	41%	71%
Coverage:	23%	39%

if Distance to nearest Ford dealer $\geq$ 12968
(Approximate percentiles 92% - 100%)
(true: 202 false: 2355 unique false: 2355)
then
Ford market share segment F (Sierra) $\geq$ 0.0009
(Approximate percentiles 12% - 100%)
(true: 2204 false: 353 unique false: 353)
Left hand side matches 8% of the database
Right hand side matches 88% of the database

	Expected	Actual
Accuracy:	86%	38%
Coverage:	8%	3%

if Number of Ford Dealers 0
(Approximate percentiles 62% - 100%)
(true: 936 false: 1621 unique false: 1621)
then
Ford market share segment B (Fiesta) $\geq$ 0.20
(Approximate percentiles 40% - 100%)
(true: 1386 false: 1171 unique false: 1171)
Left hand side matches 38% of the database
Right hand side matches 60% of the database

	Expected	Actual
Accuracy:	54%	68%
Coverage:	37%	46%

if Proportion of households with 0 cars ≥ 0.33
(Approximate percentiles 64% - 100%)
(true: 867 false: 1690 unique false: 383)
and
Proportion of households with 2 children ≥ 0.11
(Approximate percentiles 22% - 100%)
(true: 1939 false: 618 unique false: 304)
and
Proportion of unemployed in population ≥ 0.036
(Approximate percentiles 48% - 100%)
(true: 1316 false: 1241 unique false: 34)
then
Ford market share segment D (Escort) ≥ 0.23
(Approximate percentiles 58% - 100%)
(true: 1012 false: 1545 unique false: 1147)
and
Ford market share segment F (Sierra) ≥ 0.03
(Approximate percentiles 18% - 100%)
(true: 2073 false: 484 unique false: 86)
Left hand side matches 20% of the database
Right hand side matches 36% of the database

	Expected	Actual
Accuracy:	36%	69%
Coverage:	20%	38%

if Number of Ford Dealers > 0
(Approximate percentiles 62% - 100%)
(true: 936 false: 1621 unique false: 1621)
then
Ford market share segment B (Fiesta) ≥ 0.12
(Approximate percentiles 16% - 100%)
(true: 2113 false: 444 unique false: 31)
and
Ford market share segment F (Sierra) ≥ 0.02
(Approximate percentiles 14% - 100%)
(true: 2156 false: 401 unique false: 131)
and
Ford market share segment D (Escort) ≥ 0.11
(Approximate percentiles 14% - 100%)
(true: 2146 false: 411 unique false: 98)
and
Sales Fiesta per head population ≥ 0.0007
(Approximate percentiles 26% - 100%)
(true: 1876 false: 681 unique false: 256)
Left hand side matches 37% of the database
Right hand side matches 59% of the database

	Expected	Actual
Accuracy:	59%	82%
Coverage:	37%	50%

The distribution of “Sales Granada per head population” when
Prop. head household in socio-economic group B ≥ 0.20
(Approximate percentiles 56% - 100%)
(true: 1091 false: 1466 unique false: 86)
and
Proportion of population aged 5-15 ≤ 0.15
(Approximate percentiles 0% - 88%)
(true: 2248 false: 309 unique false: 56)
and
Prop. head household in socio-economic group A ≥ 0.05
(Approximate percentiles 72% - 100%)
(true: 710 false: 1847 unique false: 435)

has median 0.00004 and is significantly shifted from the
distribution in the database as a whole which has median value 0.

The distribution of “Sales Fiesta per head population” when
Proportion of households — Semi-Detached ≥ 0.35
(Approximate percentiles 70% - 100%)
(true: 744 false: 1813 unique false: 620)
and
Prop. head household in socio-economic group B ≥ 0.22
(Approximate percentiles 66% - 100%)
(true: 814 false: 1743 unique false: 550)

has median 0.0013 and is significantly shifted from the
distribution in the database as a whole which has median value 0.0010.

The distribution of “Ford market share segment B (Fiesta)” when
Prop. head household in socio-economic group C2 ≥ 0.19
(Approximate percentiles 78% - 100%)
(true: 525 false: 2032 unique false: 1508)
and
Prop. head household in socio-economic group C1 ≥ 0.05
(Approximate percentiles 20% - 100%)
(true: 2038 false: 519 unique false: 90)
and
Proportion of households with 1 car ≤ 0.50
(Approximate percentiles 0% - 90%)
(true: 2295 false: 262 unique false: 36)

has median 0.25 and is significantly shifted from the
distribution in the database as a whole which has median value 0.21.

E.2 Patterns found in Barclays Credit Data

```
  if      field28 in{ 02, 01 }
          (true: 1022 false: 711 unique false: 3)
  and
          field50 in{ 01 }
          (true: 189 false: 1544 unique false: 174)
  and
          field4 in{ 03, 06 }
          (true: 247 false: 1486 unique false: 79)
  or
          field5 in{ 02 }
          (true: 103 false: 1630 unique false: 213)
  and
          field4 in{ 03 , 06 }
          (true: 247 false: 1486 unique false: 69)
  then
          class = 1
          (true: 294 false: 1439 unique false: 1439)
Left hand side matches 4% of the database
Right hand side matches 17% of the database

              Expected   Actual
  Accuracy:     17%       62%
  Coverage:      4%       13%
```

```
  if      field4 in{ 02 , 03 , 06 , 04 }
          (true: 773 false: 960 unique false: 312)
  and
          field60 in{ 02 }
          (true: 594 false: 1139 unique false: 491)
  then
          class = 1
          (true: 294 false: 1439 unique false: 1439)
Left hand side matches 16% of the database
Right hand side matches 17% of the database

              Expected   Actual
  Accuracy:     17%       42%
  Coverage:     16%       40%
  if      field49 in{ 01 }
          (true: 1474 false: 259 unique false: 179)
  and
          field60 in{ 01 }
          (true: 1139 false: 594 unique false: 514)
  then
          class = 0
          (true: 1439 false: 294 unique false: 294)
Left hand side matches 55% of the database
Right hand side matches 83% of the database

               Expected   Actual
  Accuracy:      83%        90%
  Coverage:      55%        60%
```

```
  if      field19 in{ 05 , 06 }
          (true: 936 false: 797 unique false: 797)
  then
          class = 0
          (true: 1439 false: 294 unique false: 294)
Left hand side matches 54% of the database
Right hand side matches 83% of the database

               Expected   Actual
  Accuracy:      83%        90%
  Coverage:      54%        59%
```

Bibliography

A. V. Aho, R. Sethi, and J. D. Ullman, 1986. *Compilers: Principles, Techniques and Tools*. Addison-Wesley Computer Science series. Addison-Wesley, Reading, Massachusetts.

Thomas Bäck and Hans-Paul Schwefel, 1993. An overview of evolutionary algorithms for parameter optimisation. *Evolutionary Computation*, 1(1):1–24.

Ian D. Boyd, Patrick D. Surry, and Nicholas J. Radcliffe, 1994. Constrained gas network pipe sizing with genetic algorithms. Technical Report EPCC–TR94–11, Edinburgh Parallel Computing Centre.

L. Breiman, J. Friedman, R. Olshen, and C. Stone, 1984. *Classification and Regression Trees*. Wadsworth International Group.

Corinna Cortes, L. D. Jackel, and Wan-Ping Chiang, 1995. Limits on learning machine accuracy imposed by data quality. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press.

Jun Cui, Terence C. Fogarty, and John Gammack, 1993. Searching databases using parallel genetic algorithms on a transputer computing surface. *Future Generation Computer Systems*, 9:33–40.

Lawrence Davis and David Orvosh, 1993. Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo).

Lawrence Davis, 1991. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold (New York).

Kenneth A. DeJong, William M Spears, and Diana F Gordon, 1993. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188.

John F. Elder and Daryl Pregibon, 1995. A statistical perspective on kdd. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press.

Ian W. Flockhart, 1995a. Functional specification for RPL2 rule library. Technical Report EPCC-AIKMS-GA-MINER-FS-RULE-LIB, Edinburgh Parallel Computing Centre.

Ian W. Flockhart, 1995b. GA-MINER: Functional specification for rule representations. Technical Report EPCC-AIKMS-GA-MINER-FS-RULE, Edinburgh Parallel Computing Centre.

L. J. Fogel, A. J. Owens, and M. J. Walsh, 1966. *Artificial Intelligence Through Simulated Evolution*. Wiley Publishing (New York).

Carlos M. Fonseca and Peter J. Fleming, 1993. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423. Morgan Kaufmann (San Mateo).

Carlos M. Fonseca and Peter J. Fleming, 1994. Multiobjective evolutionary algorithms: An overview. In Terence C. Fogarty, editor, *AISB Workshop on Evolutionary Computing, University of Leeds*.

William J. Frawley, 1991. Using functions to encode domain and contextual knowledge in statistical induction. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 261–275. MIT Press.

A. A. Freitas and S. H. Lavington, 1995. A data-parallel primitive for high performance knowledge discovery in large databases. Technical Report CSM-242, Department of Computer Science, University of Essex.

Friedrich Gerbhardt, 1994. Discovering interesting statements for a database. *Applied Stochastic Models and Data Analysis*, 10:1–14.

J. D. Gibbons, 1985. *Nonparametric Statistical Inference*. Statistics: textbooks and monographs. Marcel-Dekker, Inc, New York and Basel, 2nd edition.

Attilio Giordana, Filippo Neri, and Lorenza Saiat, 1994. Search-intensive concept induction. Technical report, Univerità di Torino, Dipartimento di Informatica, Corso Svizzera 185, 10149 Torino, Italy.

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

David Perry Green and Stephen F. Smith, 1993. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257.

David Perry Greene and Stephen F. Smith, 1994. Using coverage as a model building constraint in learning classifier systems. *Evolutionary Computation*, 2(1).

T. J. Harding, P. R. King, and N. J. Radcliffe, 1995. Hydrocarbon production scheduling with genetic algorithms. submitted to J. Petr. Tech.

John H. Holland, 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press (Ann Arbor).

John H. Holland, 1985. Properties of the bucket brigade algorithm. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates (Hillsdale).

Marcel Holsheimer and Martin L. Kersten, 1994. Architectural support for data mining. Technical Report CS-R9429, ISSN 0169-118X, CWI.

Peter Hoschka and Willi Klösgen, 1991. A support system for interpreting statistical data. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*. MIT Press.

David Jensen, 1991. Knowledge discovery through induction with randomization testing. In G. Piatetsky-Shapiro, editor, *Proceedings of the 1991 Knowledge Discovery in Databases Workshop*. AAAI.

Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A.Inkeri Verkamo, 1994. Finding interesting rules from large sets of discovered association rules. In Nabil R. Adam, Bharat K. Bhargava, and Yelena Yesha, editors, *Third International Conference on Information and Knowledge Management*. ACM Press.

W. Klösgen, 1992. Problems for knowledge discovery in databases and their treatment in the statistics interpreter EXPLORA. *International Journal for Intelligent Information Systems*, 7(7):649–673.

W. Klösgen, 1994a. Efficient discovery of interesting statements in databases. Technical report, German National Research Center for Computer Science (GMD), 53757 Sankt Augustin, Germany.

W. Klösgen, 1994b. Exploration of simulation experiments by discovery. In *Proceedings of KDD-94 Workshop*. AAAI.

John R. Koza, 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Books, MIT Press (Cambridge, Mass).

W. Mendenhall and R. J. Beaver, 1994. *Introduction to Probability and Statistics*. Duxbury Press, Belmont, California, 9th edition.

Pablo Moscato and Michael G. Norman, 1992. A “memetic” approach for the travelling salesman problem — implementation of a computational ecology for combinatorial optimisation on message-passing systems. In *Proceedings of the International Conference on Parallel Computing and Transputer Applications*. IOS Press (Amsterdam).

Sreerama Murthy and Steven Salzberg, 1995. Decision tree induction: How effective is the greedy heuristic? In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press.

Filippo Neri and Attilio Giordana, 1995. A parallel genetic algorithm for concept learning. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufman.

G. Piatetsky-Shapiro, C. Matheus, P. Smyth, and R. Uthurusamy, 1993. Kdd-93: Progress and challenges in knowledge discovery in databases. Kdd-93 workshop summary, Knowledge Discovery in Databases-93.

Gregory Piatetsky-Shapiro, 1991. Discovery, analysis and presentation of strong rules. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*. MIT Press.

G. Piatetsky-Shapiro, 1992. Knowledge discovery workbench for exploring business databases. *International Journal for Intelligent Information Systems*, 7(7):675–686.

J. R. Quinlan, 1986. Induction of decision trees. *Machine Learning*, 1(1):81–106.

J. R. Quinlan, 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in Machine Learning. Morgan Kaufmann, San Mateo, California.

Françis Raab, 1994. TCP Benchmark D. Technical report, Transaction Processing Performance Council.

Nicholas J. Radcliffe and Patrick D. Surry, 1994a. Fitness variance of formae and performance prediction. Technical report, To appear in Foundations of Genetic Algorithms 3.

Nicholas J. Radcliffe and Patrick D. Surry, 1994b. Formal memetic algorithms. In Terence C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, pages 1–16. Springer-Verlag, Lecture Notes in Computer Science 865.

Nicholas J. Radcliffe and Patrick D. Surry, 1994c. The Reproductive Plan Language RPL2: Motivation, architecture and applications. In J. Stender, E. Hillebrand, and J. Kingdon, editors, *Genetic Algorithms in Optimisation, Simulation and Modelling*. IOS Press (Amsterdam).

Nicholas J. Radcliffe and Patrick D. Surry, 1995. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In *To appear in Lecture Notes in Computer Science, Volume 1000*. Springer-Verlag (New York).

Nicholas J. Radcliffe, 1992. Non-linear genetic representations. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 259–268. Elsevier Science Publishers/North Holland (Amsterdam).

Nicholas J. Radcliffe, 1994. The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 10:339–384.

T. S. Ray, 1992. Evolution and optimization of digital organisms. Distributed with Tierra v3.1.

Thomas S. Ray, 1994. An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(1/2):195–226.

Avi Silberschatz and Alexander Tuzhilin, 1995. On subjective measures of interestingness in knowledge discovery. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press.

Stephen F. Smith, 1980. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh.

Stephen F. Smith, 1983. Flexible learning of problem solving heuristics through adaptive search. In Alan Bundy, editor, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, volume 1.

Stephen F. Smith, 1984. Adaptive learning systems. In Richard Forsyth, editor, *Expert Systems, Principles and case studies*. Chapman and Hall Ltd.

Padhraic Smyth and Rodney M. Goodman, 1991. Rule induction using information theory. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159–176. MIT Press.

Patrick D. Surry and Nicholas J. Radcliffe, 1994a. *The Reproductive Plan Language RPL2*. Edinburgh Parallel Computing Centre.

Patrick D. Surry and Nicholas J. Radcliffe, 1994b. RPL2: A language and parallel framework for evolutionary computing. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature III*, pages 628–637. Springer-Verlag, Lecture Notes in Computer Science 866.

Patrick D. Surry, Nicholas J. Radcliffe, and Ian D. Boyd, 1995. A multi-objective approach to constrained optimisation of gas supply networks: The COMOGA method. In Terence C. Fogarty, editor, *AISB Workshop on Evolutionary Computing, University of Sheffield*.

Patrick D. Surry, 1993a. RPL2 programmer's guide. Technical Report EPCC-BG-PAP-RPL2-PG, Edinburgh Parallel Computing Centre.

Patrick D. Surry, 1993b. RPL2 user guide. Technical Report EPCC-BG-PAP-RPL2-UG, Edinburgh Parallel Computing Centre.

J. D. Ullman, 1988. *Principles of Database and Knowledge Base Systems Volume 1*. Principles of Computer Science Series. Computer Science Press, Rockville, Maryland.

W.H.Inmon and S.Osterfelt, 1991. *Understanding Data Pattern Processing: the key to Competitive Advantage*. QED Technical Publishing Group, Wellesley, MA.

David H. Wolpert and William G. Macready, 1995. No free lunch theorems for search. Technical Report SFI–TR–95–02–010, Santa Fe Institute.

Jan M. Zytkow and John Baker, 1991. Interactive mining of regularities in databases. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 31–53. MIT Press.

Jan M. Zytkow and Robert Zembowicz, 1993. Database exploration in search of regularities. *Journal of Intelligent Information Systems*, 2:39–81.