Appears in "Genetic Algorithms in Optimisation, Simulation and Modelling", Eds: J. Stender, E. Hillebrand, J. Kingdon, IOS Press, 1994.

The Reproductive Plan Language RPL2: Motivation, Architecture and Applications

Nicholas J. Radcliffe and Patrick D. Surry

{njr,pds}@epcc.ed.ac.uk

Edinburgh Parallel Computing Centre

King's Buildings, University of Edinburgh

Scotland, EH9 3JZ

Abstract. The reproductive plan language RPL2 is a computer language designed to facilitate the writing, execution and modification of evolutionary algorithms. It provides a number of data parallel constructs appropriate to evolutionary computing, facilitating the building of efficient parallel interpreters and compilers. This facility is exploited by the current interpreted implementation. RPL2 supports all current structured population models and their hybrids at language level. Users can extend the system by linking against the supplied framework C-callable functions, which may then be invoked directly from an RPL2 program. There are no restrictions on the form of genomes, making the language particularly well suited to real-world optimisation problems and the production of hybrid algorithms. This paper describes the theoretical and practical considerations that shaped the design of RPL2, the language, interpreter and run-time system built, and a suite of industrial applications that have used the system.

1 Motivation

As evolutionary computing techniques acquire greater popularity and are shown to have ever wider application a number of trends have emerged. The emphasis of early work in genetic algorithms on low cardinality representations is diminishing as problem complexities increase and more natural data structures are found to be more convenient and effective. There is now extensive evidence, both empirical and theoretical, that the arguments for the superiority of binary representations were at least overstated. As the fields of genetic algorithms, evolution strategies, genetic programming and evolutionary programming come together, an ever increasing range of representation types are becoming commonplace.

The last decade, during which interest in evolutionary algorithms has increased, has seen the simultaneous development and wide-spread adoption of parallel and distributed computing. The inherent scope for parallelism evident in evolutionary computation has been widely noted and exploited, most commonly through the use of *structured population models* in which mating probabilities depend not only on fitness but also on *location.* In these structured population models each member of the population (variously referred to as a chromosome, genome, individual or solution) has a site—most commonly either a unique coordinate or a shared island number—and matings are more common between members that are close (share an island or have neighbouring coordinates) than between those that are more distant. Such structured population models, which are described in more detail in section 2, have proved not only highly amenable to parallel implementation, but also in

many cases computationally superior to more traditional *panmictic* (unstructured) models in the sense of requiring fewer evaluations to solve a given problem. Despite this, so close has been the association between parallelism and structured population models that the term *parallel genetic algorithm* has tended to be used for both. The more accurate term *structured population model* seems preferable, when it is this aspect that is referred to.

The authors both work for Edinburgh Parallel Computing Centre, which makes extensive use of evolutionary computing techniques (in particular, genetic algorithms) for both industrial and academic problem solving, and wished to develop a system to simplify the writing of and experimentation with evolutionary algorithms. The primary motivations were to support arbitrary representations and genetic operators along with all population models in the literature and their hybrids, to reduce the amount of work and coding required to develop each new application of evolutionary computing, and to provide a system that allowed the efficient exploitation of a wide range of parallel, distributed and serial systems in a manner largely hidden from the user. RPL2, the second implementation of the *Reproductive Plan Language*, was produced in partnership with British Gas plc to satisfy these aims. This paper motivates the design of the system, focusing in particular on the population models supported by RPL2 (section 2), its support for arbitrary representations (section 3), and the modes of parallelism it supports (section 4), details the current implementation (section 5), and illustrates the benefits of exploiting such a system by presenting a suite of applications for which it has been used (section 6). Several example *reproductive plans* are given in the appendix.

2 Population models

The original conception of genetic algorithms (Holland, 1975) contained no notion of the location of a genome in the population. All solutions were simply held in some unstructured group, and allowed to inter-breed without restriction. Despite the continuing successes of such unstructured, or *panmictic* models, much recent work has focused on the addition of a notional co-ordinate to each genome in the population. Interaction between genomes is then restricted to neighbours having similar co-ordinates. This was perhaps first initiated by the desire to efficiently exploit parallel and distributed computers, but the idea has been shown to be of more general utility, increasing the efficiency of the algorithm in terms of number of function evaluations even when simulated on a serial machine. Population structure is also useful in encouraging *niching* whereby different parts of the population converge to different optima, supporting multi-modal covering, and preventing premature convergence.

Structured populations fall into two main groups—fine-grained and coarse-grained. These differ primarily in the degree to which they impose structure on the population, and are explained in more detail in the following sections.

Unstructured populations are, of course, supported in RPL2 using simple variable-length arrays which may be indexed directly or treated as last-in-first-out stacks. This is illustrated in the code fragment below, as well as in the first example plan of the appendix. The example shows the declaration of an unstructured population (a genome stack). Two parents are selected from this population using tournament selection (a supplied operator), and they are crossed using N-point crossover to produce a child.

```
genome  mother, father, child;
gstack  population;

mother := SelectRawTournament(population, maxIsBest,
                tournSize, probOfBest, useDuplicates);
father := SelectRawTournament(population, maxIsBest,
                tournSize, probOfBest, useDuplicates);
```

```
child  := CrossNpt(mother, father, crossPts, probOfCross);
```

2.1 Coarse-Grained Models

In the *coarse-grained* model, (probably better known as the *island* model), several panmictic populations are allowed to develop in parallel, occasionally exchanging genomes in the process of *migration*. In some cases the island to which a genome migrates is chosen stochastically and asynchronously (e.g. Norman, 1988), in others deterministically in rotation (e.g. Whitley *et al.,* 1989a). In still other cases the islands themselves have a structure such as a ring and migration only occurs between neighbouring islands (e.g. Cohoon *et al.,* 1987); this last case is sometimes known as the *stepping stone* model. The largely independent course of evolution on each island encourages niching (or *speciation*) while ultimately allowing genetic information to migrate anywhere in the (structured) population. This helps to avoid premature convergence and encourages covering if the algorithm is run with suitably low migration rates.

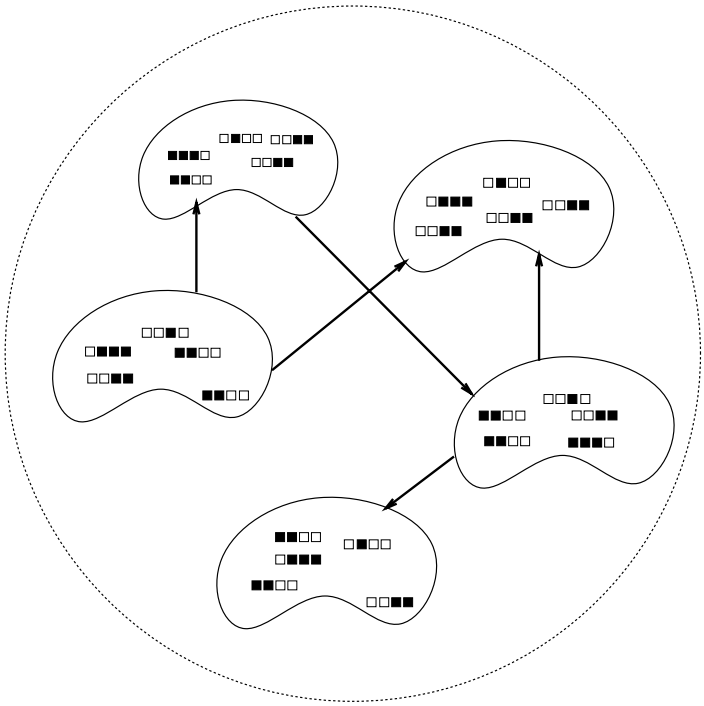


Figure 1: This picture on shows the coarse-grained *island* model, in which isolated subpopulations exist, possibly on different processors, each evolving largely independently. Genetic information is exchanged with low frequency through migration of solutions between subpopulations. This helps track multiple optima and reduces the incidence of premature convergence.

Coarse-grained models are typically only loosely synchronous, and work well even on distributed systems with very limited communications bandwidths. They are supported in RPL2 by allowing populations to be declared as arbitrary-dimensional structures with fixed or cyclic boundaries and by means of the `structfor` loop construct, which allows (any part of) a reproductive plan to be executed over such a structured population in an unspecified order, with the system exploiting parallelism if it is available. Migration occurs through the use of supplied operators (see the second example plan in the appendix). The following code fragment uses a structured population of ten islands connected in a ring ("@" denotes a cyclic boundary). The population is declared with a qualifier indicating that it is a parallel array corresponding to the structure template. The selection and crossover operators of the previous panmictic example are now enclosed in a `structfor` loop indicating that each step actually takes place simultaneously on all ten islands.

```
structure [10@island];
genome [*]      mother, father, child;
gstack [*]      population;

structfor [*]
        mother := SelectRawTournament(population, maxIsBest,
                        tournSize, probOfBest, useDuplicates);
        father := SelectRawTournament(population, maxIsBest,
                        tournSize, probOfBest, useDuplicates);
        child  := CrossNpt(mother, father, crossPts, probOfCross);
endstructfor
```

Other papers describing variants of the island model include Petty & Leuze (1989), Cohoon *et al.* (1990) and Tanese (1989).

2.2 Fine-grained models

The other principal structured population model is the fine-grained model (figure 2), also known variously as the *diffusion* (Muehlenbein *et al.,* 1991) or *cellular* model (Gordon & Whitley, 1993). In such models, it is usual for every individual to have a unique coordinate in some space—typically a grid of some dimensionality with either fixed or cyclic boundary conditions. In one dimension, lines or rings are most common, while in two dimensions regular lattices or tori and so forth dominate. More complex topologies in higher dimensions are also possible. Individuals mate only within a neighbourhood called a *deme* and these neighbourhoods overlap by an amount that depends on their size and shape. Replacement is also local. This model is well suited to implementation on so-called Single-Instruction Multiple-Data (SIMD) parallel computers (also called array processors or, loosely, "data-parallel" machines). In SIMD machines a (typically) large number of (typically) simple processors all execute a single instruction stream synchronously on different data items, usually configured in a grid (Hillis, 1991). Despite this, one of the earlier implementations was by Gorges-Schleuter (1989), who used a transputer array. It need hardly be said that the model is of general applicability on serial or general parallel hardware.

The characteristic behaviour of such fine-grained models is that in-breeding within demes tends to cause speciation as clusters of related solutions develop, leading to natural niching behaviour (Davidor, 1991). Over time, strong characteristics developed in one neighbourhood of the population gradually spread across the grid because of the overlapping nature of demes, hence the term *diffusion* model. As in real diffusive systems, there is of course a long-term tendency for the population to become homogeneous, but it does so markedly less quickly than in panmictic models. Like the island model, the diffusion model tends to help in avoiding premature convergence to local optima. Moreover, if the search is stopped at a suitable stage, the niching behaviour allows a larger degree of coverage of different optima to be obtained than is typically possible with unstructured populations. Other papers describing variants of the diffusion model include Manderick & Spiessens (1989), Muehlenbein (1989), Gorges-Schleuter (1990), Spiessens & Manderick (1991), Davidor (1991), Baluja (1993), Maruyama *et al.* (1993) and Davidor *et al.* (1993).

RPL2 supports fine-grained population models through use of the `structfor` loop construct, and through specification of a deme structure. Demes are specified using a special class of user-definable operator (of which several standard instances are provided), and indicate a pattern of neighbours for each location in the population structure. The code fragment below defines a ten by ten torus as the population structure, and indicates that a deme consists of all genomes within a three unit radius. The example is similar to the previous coarse-grained version except that the neighbours of each member of the population must first be collected

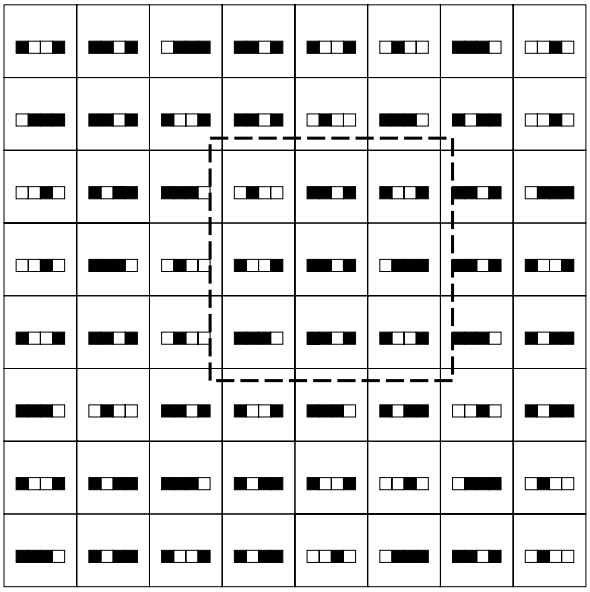


Figure 2: This picture illustrates a so-called *fine-grained* (*diffusion* or *cellular*) population structure. Each solution has a spatial location and interacts only within some neighbourhood, termed a deme. Clusters of solutions tend to form around different optima, which is both inherently useful and helps to avoid premature convergence. Information slowly diffuses across the grid as evolution progresses by mating within the overlapping demes.

using the `DemeCollect` operator before selection and crossover can take place.

```
structure [10@fine, 10@fine] deme Euclidean(3.0);
genome [*,*]    population, mother, father;
gstack [*,*]    deme;

structfor [*,*]
        DemeCollect(deme,population);
        mother := SelectRawTournament(deme, maxIsBest,
                tournSize, probOfBest, useDuplicates);
        father := SelectRawTournament(deme, maxIsBest,
                tournSize, probOfBest, useDuplicates);
        population := CrossNpt(mother, father, crossPts, probOfCross);
endstructfor
```

2.3 Hybrid Models

There is sufficient flexibility in the reproductive plan language to allow arbitrary hybrid models population models, for example, an array of islands each with fine-grained populations or a fine-grained model in which each site has an island (which could be viewed as a generalisation of the stepping stone model). Such models have not, as far as the authors are aware, been presented in the literature, but may yield interesting new avenues for exploration. An example plan which uses just such a hybrid model is given in the appendix.

3 Representation

One of the longest-running and sometimes most heated arguments in the field of genetic algorithms (and to a lesser extent the wider evolutionary computing community) concerns the representation of solutions. This is a multi-threaded debate taking in issues of problem-specific and representation-specific operators, hybridisation with other search techniques, the handling of constraints, the interpretation of the schema theorem, the meaning of genes and

the efficacy of newer variants of the basic algorithms such as genetic programming. The developers of RPL2 are strongly of the opinion that exotic representations should be the norm rather than the exception for a number of reasons outlined in this section.

3.1 Real Parameter Optimisation

A particular focus of disagreement about representations concerns the coding of real numbers in real parameter optimisation problems. The main split is between those who insist on coding parameters as binary strings and those who prefer simply to treat real parameters as floating point values. It is first necessary to clarify that the issue here is not one of the *physical* representation of a real parameter on the machine—whether it should be, for example, an IEEE floating point number, an integer array or a packed binary integer, which is an *implementational* issue—but rather how genes and alleles are defined and manipulated.

David Goldberg is usually identified—perhaps unfairly—as the leading advocate of binary representations. He has developed a theory of *virtual alphabets* for what he calls "real-coded" genetic algorithms (Goldberg, 1990). He considers the case in which the parameters themselves are treated as genes and processed using a traditional crossover operator such as n-point or uniform crossover (manipulating whole-parameter genes). In this circumstance, he argues that the genetic algorithm "chooses its own" low cardinality representation for each gene (largely from the values that happen to be present in relatively good solutions in the initial population) but then suffers "blocking", whereby the algorithm has difficulty accessing some parts of the search space through reduced ergodicity. These arguments, while valid in their own context, ignore the fact that people who use "real codings" in genetic search invariably use quite different sorts of recombination operators. These include averaging crossovers (Davis, 1991), random respectful recombination (R^3; Radcliffe, 1991a) and "blend crossover" (BLX-α; Eshelman & Schaffer, 1992). These are combined with appropriate *creep* (Davis, 1991) or *end-point* (Radcliffe, 1991a) forms of mutation. Similarly, the Evolution Strategies community, which has always used "real" codings, uses recombination operators that are equivalent to R^3 and BLX-0 (Baeck *et al.*, 1991).

The works cited above, together with Michalewicz (1992), provide compelling evidence that this approach outperforms binary codings, whether these are of the traditional or "Gray-coded" variety (Caruana & Schaffer, 1988). In particular, Davis (1991) provides a potent example of the improvement that can be achieved by moving from binary-coded to real-coded genetic algorithms. This example has been reproduced in the tutorial guide to RPL2 contained in Surry & Radcliffe (1994).

3.2 Real-World Optimisation

When tackling real-world optimisation problems, a number of further factors come into play, many of which again tend to make simple binary and related low-cardinality representations unattractive or impractical.

In industrial optimisation problems it is typically the case that the evaluation function has already been written and other search or optimisation techniques have been used to tackle the problem. This may impose constraints on the representation. While in some cases conversion between representations is feasible, in others this will be unacceptably time consuming. Moreover, the representation used by the existing evaluation function will normally have been carefully chosen to facilitate manipulation. If there is not a benefit to be gained from changing to a special "genetic" representation, it would seem perverse to do so. The same considerations apply even more strongly if *hybridisation* with a pre-existing heuristic or other search technique is to be attempted. This is important because hybrid approaches, in which a domain-specific technique is embedded, whole or in part, in a framework of evolutionary search, can almost always be constructed that outperform both pure genetic search and the

domain-specific technique. This is the approach routinely taken when tackling "real world" applications, such as those described in section 6.

Further problems arise in constrained optimisation, where some constraints (including simple bounds on parameter ranges) can manifest themselves in unnecessarily complex forms with restricted coding schemes. For example, a parameter that can take on exactly 37 different values is difficult to handle with a binary representation, and will tend either to lead to a redundant coding (whereby several strings may represent the same solution) or to having to search (or avoid) "illegal" portions of the representation space. Similar issues can arise when trying to represent permutations with, for example, binary matrices (e.g. Fox & McMahon, 1991), rather than in the more natural manner. It should be noted that even many of those traditionally associated with the "binary is best" school accept that for some classes of problems low cardinality representations are not viable. For example, it was Goldberg who, with Lingle, developed the first generalisation of schema analysis in the form of *o*-schemata for the travelling sales-rep problem (TSP; Goldberg & Lingle, 1985).

3.3 Multiple Representations

Some evolutionary algorithms have been developed that employ more than one representation at a time. A notable example of this is the work of Hillis (1991), who evolved sorting networks using a *parasite* model. Hillis's evaluation function evolved by changing the test set as the sorting networks improved. In a similar vein, Husbands & Mill (1991) have used co-evolution models in which different populations optimise different parts of a process plan which are then brought together for arbitration. This necessitates the use of multiple representations.

There are also cases in which control algorithms are employed to vary the (often large number of) parameters of an evolutionary algorithm as it progresses. For example, Davis (1989) adapts operator probabilities on the basis of their observed performance using a credit apportionment scheme. RPL2 caters for the simultaneous use of multiple representations in a single reproductive plan, which greatly simplifies the implementation of such schemes.

3.4 Schemata, Formae and Implicit Parallelism

In addition to the practical motivations for supporting complex representations, certain theoretical insights support this approach. These are obtained by considering the Schema Theorem (Holland, 1975) and the rôle of "implicit parallelism". Holland introduced the notion of a *schema* (pl. *schemata*) as a collection of genomes that share certain gene values (alleles). For example, the schema $3\square 4\square$ is the set of chromosomes with a 3 at the first locus and a 4 at the third locus.

The Schema Theorem may be stated in a fairly general form (though assuming fitness-proportionate selection for convenience) thus:

$$\left\langle N_\xi(t+1)\right\rangle \geq N_\xi(t)\frac{\hat{\mu}_\xi(t)}{\bar{\mu}(t)}\left[1-\sum_{\omega\in\Omega}p_\omega p_\omega^\xi\right] \tag{1}$$

where

- $N_\xi(t)$ is the number of members of the population at time t that are members of a given schema ξ;
- $\hat{\mu}_\xi(t)$ is the observed fitness of the schema ξ at time t, i.e. the average fitness of all the members of the population at time t that are instances (members) of the schema ξ;
- $\bar{\mu}(t)$ is the average fitness of the whole population at time t;
- Ω is the set of genetic operators in use;

- the term $p_\omega p_\omega^\xi$ quantifies the potential disruptive effect on schema membership of the application of operator $\omega \in \Omega$;
- $\langle \cdot \rangle$ denotes an expectation value.

This theorem is fairly easily proved. It has been extended by Bridges & Goldberg (1987), for the case of binary schemata, to replace the inequality with an equality by including terms for string gains as well as the disruption terms.

Holland used the concept of "implicit parallelism" (*née* intrinsic parallelism) to argue for the superiority of low cardinality representations, a theme picked up and amplified by Goldberg (1989, 1990), and more recently championed by Reeves (1993). Implicit parallelism refers to the fact that the Schema Theorem applies to *all* schemata represented in the population, leading to the suggestion that genetic algorithms process schemata rather (or as well as) individual solutions. The advocates of binary representations then argue that the *degree* of intrinsic parallelism can be maximised by maximising the number of schemata that each solution belongs to. This is clearly achieved by maximising the string length, which in turn requires minimising the cardinality of the genes used. This simple counting argument has been shown to be seriously misleading by a number of researchers, including Antonisse (1989), Radcliffe (1990, 1994a) and Vose (1991), and as Mason (1993) has noted, '[t]here is now no justification for the continuance of [the] bias towards binary encodings'.

It is both a strength and a weakness of the Schema Theorem that it applies equally, given a representation space $\mathcal{C}$ (of "chromosomes" or "genotypes") for a search space $\mathcal{S}$ (of "phenotypes"), no matter which mapping is chosen to relate genotypes to phenotypes. Assuming that $\mathcal{S}$ and $\mathcal{C}$ have the same size, there are $|\mathcal{S}|!$ such mappings (representations) available—clearly vastly more than the size of the search space itself—yet the schema theorem applies equally to each of them. The only link between the representation and the theorem is the term $\hat{\mu}_\xi(t)$. The theorem states that the expected number of instances of any schema at the next time-step is directly proportional to its *observed* fitness (in the *current* population) relative to everything else in the population (subject to the effects of disruption; Radcliffe, 1994a). Thus, the ability of the schema theorem, which governs the behaviour of a simple genetic algorithm, to lead the search to interesting areas of the space is limited by the quality of the information it collects about the space through observed schema fitness averages in the population.

It can be seen that if schemata tend to collect together solutions with related performance, then the fitness-variance of schemata will be relatively low, and the information that the schema theorem utilises will have predictive power for previously untested instances of schemata that the algorithm may generate. Conversely, if schemata do not tend to collect together solutions with related performance, while the predictions the theorem makes about schema membership of the next population will continue to be accurate, the performance of the solutions that it generates cannot be assumed to bear any relation to the fitnesses of the parents. This clearly shows that it is essential that domain-specific knowledge be used in constructing a genetic algorithm, through the choice of representation and operators, whether this be implicit or—as is advocated in the present paper—explicit. If no domain-specific knowledge is used in selecting an appropriate representation, the algorithm will have no opportunity to exceed the performance of a random search.

In addition to these observations about the Schema Theorem's representation independence and the sensitivity of its predictions to the fitness variance of schemata, Vose (1991) and Radcliffe (1990) have independently proved that the "schema" theorem actually applies to *any* subset ξ of the search space, not only schemata, provided that the disruption coefficients $p_\omega p_\omega^\xi$ are computed appropriately for whichever set ξ is actually considered. Vose's response to this was to term a generalised schema a *predicate* and to investigate transformations

of operators and representations that change problems that are hard for genetic algorithms into problems that are easy for them (Vose & Liepins, 1991). This was achieved through exploiting a limited duality between operators and representations, which is discussed briefly in Radcliffe (1994a). Radcliffe instead termed the generalised schemata *formae* (sing. *forma*) and set out to develop a formalism to allow operators and representations to be developed with regard to stated assumptions about performance correlations in the search space. The aim was to maximise the predictive power of the Schema Theorem (and thus its ability to guide the search effectively) by allowing the developer of a genetic algorithm for some particular problem to codify knowledge about the search space by specifying families of formae that might reasonably be assumed to group together solutions with related performance.

3.5 Forma Analysis

Given a collection of formae (generalised schemata, or arbitrary subsets of the search space) thought relevant to performance, forma analysis suggests two key properties for a recombination operator, both motivated by the way conventional genetic crossover operators manipulate genes. *Respect* requires that if both parents are members of some forma then so should be all their children produced by recombination alone. For example, if eye colour has been chosen as an important characteristic, and both parents have blue eyes, then respect restricts recombination to produce only children with blue eyes. A stronger form of this condition, called *gene transmission*, requires that children inherit each of their genes from one or other parent, so that if one parent had green eyes and the other had blue eyes a child produced by recombination would be bound to have either green or blue eyes. It is not, however, always possible to identify suitable genes, so this condition is not always imposed. For a detailed exposition of "genetic search" without "genes" the reader is referred to the discussion of *allelic representations* in Radcliffe & Surry (1994).

The other desirable property for recombination operators is *assortment*, which requires that recombination should be capable of bringing together any mixture of compatible genetic material present in the parents. Thus, for example, if one parent has blue eyes, and the other has curly hair, then if these are compatible characteristics it should be possible for an assorting recombination operator to combine these characteristics.

Although these two principles seem rather innocuous, there are many problems for which the natural formae cannot simultaneously be respected and assorted. Such sets of formae are said to be *non-separable*. A varied suite of domain-independent recombination, mutation and hill-climbing operators has been developed using the principles of respect and assortment together with related ideas. These include random respectful recombination and random transmitting recombination (R^3 and RTR respectively; Radcliffe, 1991b), random assorting recombination (RAR; Radcliffe, 1991b), generalised n-point crossover (GNX; Radcliffe & Surry, 1994), binomial minimal mutation (BMM; Radcliffe, 1994b) and minimal-mutation-based hill-climbing (Radcliffe & Surry, 1994). Of these, R^3 is the simplest. It operates by taking all the genes common to the two parents and inserting them in the child while making random (legal) choices for remaining genes. In some situations this is surprisingly effective, while in others a more sophisticated approach is required. The set of all solutions sharing all the genes of two parents x and y is called their *similarity set*, denoted $\Sigma(\{x, y\})$, so R^3 can be seen to pick an arbitrary member of the parents' similarity set.

Locality Formae for Real Parameter Optimisation

In considering continuous real parameter optimisation problems it seems reasonable to suppose that solutions that are close to one another might have similar performance. *Locality formae* (Radcliffe, 1991a) group chromosomes on the basis of their proximity to each other, and can be used to express this supposition. Suppose that a single parameter function is defined over a real interval $[a, b)$. Then formae are defined that divide the interval up into

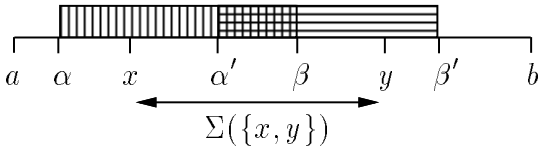


Figure 3: Given $x \in [\alpha, \beta)$ and $y \in [\alpha', \beta')$, with $y > x$, the formae are compatible only if $\beta > \alpha'$. The arrow shows the similarity set $\Sigma(\{x, y\})$.

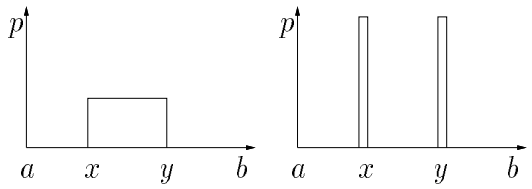


Figure 4: The left-hand graph shows (schematically) the probability of selecting each point along the axis under R^3. The right-hand graph shows the corresponding diagram for standard crossover with real genes.

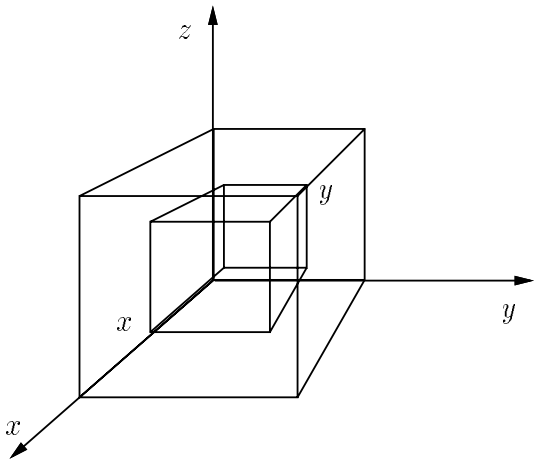


Figure 5: The n-dimensional R^3 operator for real genes picks any point in the hypercuboid with corners at the chromosomes being recombined, x and y.

strips of arbitrary width. Thus, a forma might be a half-open interval $[\alpha, \beta)$ with α and β both lying in the range $[a, b]$. These formae are separable. Respect requires that all children are instances of any formae which contain both parents x and y. Clearly the similarity set of x and y (the smallest interval which contains them both) is $[x, y]$, where it has been assumed, without loss of generality, that $y \geq x$. Thus respect requires that all their children lie in $[x, y]$. Similarly, if x is in some interval $\xi = [\alpha, \beta)$ and y lies in some other interval $\xi' = [\alpha', \beta')$, then for these formae to be compatible the intersection of the intervals that define them must be non-empty ($\beta > \alpha'$; figure 3) and so picking a random element from the similarity set $[x, y]$ allows any element that lies in the intersection to be picked, showing that R^3 fulfils the requirements of assortment (figure 4). The n-dimensional R^3 operator picks a random point in the n-dimensional hypercuboid with corners at the two chromosomes x and y (figure 5).

Both this operator and its natural analogue for k-ary string representations, which for each locus picks a random value in the range defined by the alleles from the two parents, suffer from a bias away from the ends of the interval. It is therefore necessary to introduce a mutation operator that offsets this bias in order to ensure that the whole search space remains accessible. An appropriate mutation operator acts with very low probability to introduce the extremal values at an arbitrary locus along the chromosome. In the one dimensional case this amounts to occasionally replacing the value of one of the chromosomes with an a or a b. The combination of R^3 and such *end-point* mutation provides a surprisingly powerful set of genetic operators for some problems, outperforming more common (binary) approaches (Radcliffe, 1991a). The *blend crossover* operator BLX-0.5, which is a generalisation of R^3 developed by Eshelman & Schaffer (1992), performs even better.

Locality formae are not, of course, the only alternatives to schemata which can be applied to real-valued problems, and there is no suggestion here that locality formae should be seen as a generic or definitive alternative to schemata. It would be interesting, for example, to attempt to construct formae and representations on the basis of fourier analysis, or some other complete orthonormal set of functions over the space being searched.

Edge Formae for the Travelling Sales-rep Problem

The travelling sales-rep problem (TSP) is perhaps the most studied combinatorial optimisation problem, and has certainly attracted much effort with evolutionary algorithms. Given a set of cities, the problem is to find the shortest route for a notional sales-rep to follow, visiting each city exactly once. This problem has a number of important industrial applications, including finding drilling paths for printed circuit boards to maximise production speed. It seems clear, as Whitley *et al.* (1989b) have argued, that the edges rather than the vertices of the graph are central to the TSP. While there might be some argument as to whether or not the edges should be taken to be directed, the symmetry of the euclidean metric used in the evaluation function suggests that undirected edges suffice.

If the towns (vertices) in an n-city TSP are numbered 1 to n, and the edges are described as non-ordered pairs of vertices (a, b), then apparently suitable *edge formae* are simply sets of edges, subject to the condition that every vertex appears in the description of exactly two edges. These formae are not separable. To see this, consider two tours x and y, with x containing the fragment 2–1–3 and y containing 4–1–3. Plainly these have the common edge $(1, 3)$ (which is, of course, the same as $(3, 1)$). Edge formae are described by the list of edges they require to be present in angle brackets, so that x is an instance of the forma $\langle(1, 2)\rangle$ and y is an instance of the forma $\langle(1, 4)\rangle$. These formae are clearly compatible, because any tour containing the fragment 2–1–4 is in their intersection[1]

$$\langle(1, 2)\rangle \cap \langle(1, 4)\rangle = \langle(1, 2), (1, 4)\rangle.$$

Any recombination operator that respects these formae is bound to include the common edge $(1, 3)$ in all offspring from these parents. This precludes generating a child in $\langle(1, 2), (1, 4)\rangle$. Since assortment requires that this child be capable of being generated this shows that these formae are not separable.

R^3 can be defined for edge formae even though they are not separable: it works simply by copying common edges into the child and then putting in random edges in such a way as to complete a legal tour. The lack of separability simply ensures that R^3 does not assort the formae. Extensive experiments with R^3-related operators for the TSP are related in Radcliffe & Surry (1994).

[1] Curiously, the intersection operation for these edge formae looks like the set *union* operation. This is because $\langle(1, 3)\rangle$ is really an abbreviation for the set of chromosomes containing the 1–3 edge.

Formae for Set Problems

A large number of optimisation problems are naturally formulated as subset extraction problems, i.e. given some "universal" set, find the best subset of it according to some criterion. Examples include stock-market portfolio optimisation (Shapcott, 1992), choosing k sites from n possible sites for retail dealers (George, 1994) and optimising the connectivity of a three layer neural network (Radcliffe, 1993). If the size of the subset is not fixed, the natural way to tackle this problem is by using a binary string the length of the universal set, using a 1 to indicate that the given element is in the subset. If, however, the size is fixed this is more problematical, because this constrains the number of ones in the string. A more natural approach is to store the elements in the subset and apply appropriate genetic operators directly. In this case the elements themselves can form alleles, and approaches as simple as choosing the desired number of elements from those available between the parents can be effective. This method happens to equate to use of RAR_0 (Radcliffe, 1992b). Forma analysis for set problems is covered extensively in Radcliffe (1992a).

General Representations

It has been argued in the preceding sections that there are theoretical, practical and empirical motivations for moving away from the very simple binary string representations that have dominated genetic algorithms for so long. Combined with the successes shown by genetic programming, evolution strategies and evolutionary programming these form a compelling case for supporting the use of arbitrary data structures as genetic representations. The way in which RPL2 achieves this is discussed in section 5.

4 Parallelism

Evolutionary algorithms that use populations are inherently parallel in the sense that—depending on the exact reproductive plan used—each chromosome update is to some extent independent of the others. There are a number of options for implementation on parallel computers, several of which have been proposed in the literature and implemented. As has been emphasised, population structure has tended to be tied closely to the architecture of a particular target machine to date, but there is no reason, in general, why this need be so.

Parallelism is supported in RPL2 at a variety of levels. Data decomposition of structured populations can be achieved transparently, with different regions of the population evolving on different processors, possibly partially synchronised by inter-process communication. Distribution of fine-grained models tends to require more interprocess communication and synchronisation so their efficiency is more sensitive to the computation-to-communications ratio for the target platform.

Task farming of compute intensive tasks, such as genome evaluation (e.g. Verhoeven *et al.*, 1992; Starkweather *et al.*, 1990), is also provided via the `forall` loop construct, which indicates a set of operations to be performed on all members of a population stack in no fixed order. This is particularly relevant to real-world optimisation tasks for which it is almost invariably the case that the bulk of the time is spent on fitness evaluation. (For example see section 6.) User operators may themselves include parallel code or run on parallel hardware independently of the framework, giving yet more scope for parallelism.

RPL2 will run the same reproductive plan on serial, distributed or parallel hardware without modification using the minimum degree of synchronisation consistent with the reproductive plan specified.

5 System Architecture

RPL2 defines a C-like data-parallel language for describing reproductive plans. It is designed to simplify drastically the task of implementing and experimenting with evolutionary

algorithms. Both parallel and serial implementations of the run-time system exist and will execute the same plans without modification.

The language provides a small number of built-in constructs, along with facilities for calling user-defined operators. Several libraries of such operators are provided with the basic framework, as are definitions of several commonly used genetic representations. Two example plans are presented at the end of this paper.

5.1 Language Features

RPL2 is a simple procedural language, similar to C, with six basic data types—`bool`, `int`, `real`, `string`, `genome` and `gstack`. The genome and gstack types are explained further below.

Simple control flow using `if`, `while`, and `for` are provided, as are normal algebraic mathematical and logical expressions. User defined operators (C-callable functions) are made visible as procedures in the language with the `use` declaration, which is analogous to C's `#include`.

The data-parallel aspect of the language is supported using the concept of a population structure, which is declared as a multi-dimensional hypercuboid. Arrays corresponding to any combination of *axes* of the population structure may then be declared and manipulated in a SIMD-like way (i.e. an operation on such an array affects every element within it). Several special operators to project and reduce the dimensionality of such arrays are also provided. Built-in constructs to support parallelism include two types of parallel loops—the data-parallel `structfor`, and a `forall` construct to indicate that data-independent farming out of work is possible.

5.2 The RPL2 Framework

The RPL2 framework provides an implementation of the reproductive plan language based on a interpreter and a run-time system, supported by various other modules. The diagram in figure 6 shows how these different elements interact.

The interpreter acts in two main modes: interactive commands are processed immediately, while non-interactive commands are compiled to an internal form as a reproductive plan is being defined. Facilities also exist for batch processing, I/O redirection, and some on-line help. The interpreted nature of the system is especially useful for fast turn-around experimentation. The trade-off in speed over a compiled version is insignificant for real applications in which almost all of the execution time is spent in the evaluation function. The system uses the Marsaglia pseudo-random number generator (Marsaglia *et al.*, 1990), which as well as producing numbers with good statistical distributions allows identical results to be produced on different processor architectures provided that they use the same floating point representation.

Two versions of the run-time system exist, a serial (single-processor) implementation, and a parallel (multiple distributed processors) implementation. In the serial case, both the parser and the run-time system run on a single processor, and no communication is required. In the parallel case, the parser runs on a single processor, but the work of actually executing a reproductive plan is shared across other processors. Two methods for this work-sharing are provided, one in which the data space of a structured population is decomposed across a regular grid of processors and one in which extra processors are used simply as evaluation servers for compute-intensive sections of code, typically evaluation of genome fitness. A hybrid model in which the data space is distributed across some processors and others are used as a pool of evaluation servers is planned as a future extension.

Parallelism by data-decomposition is made possible by the SIMD-like nature of the language. In such a case, a structured population is typically declared and operations take

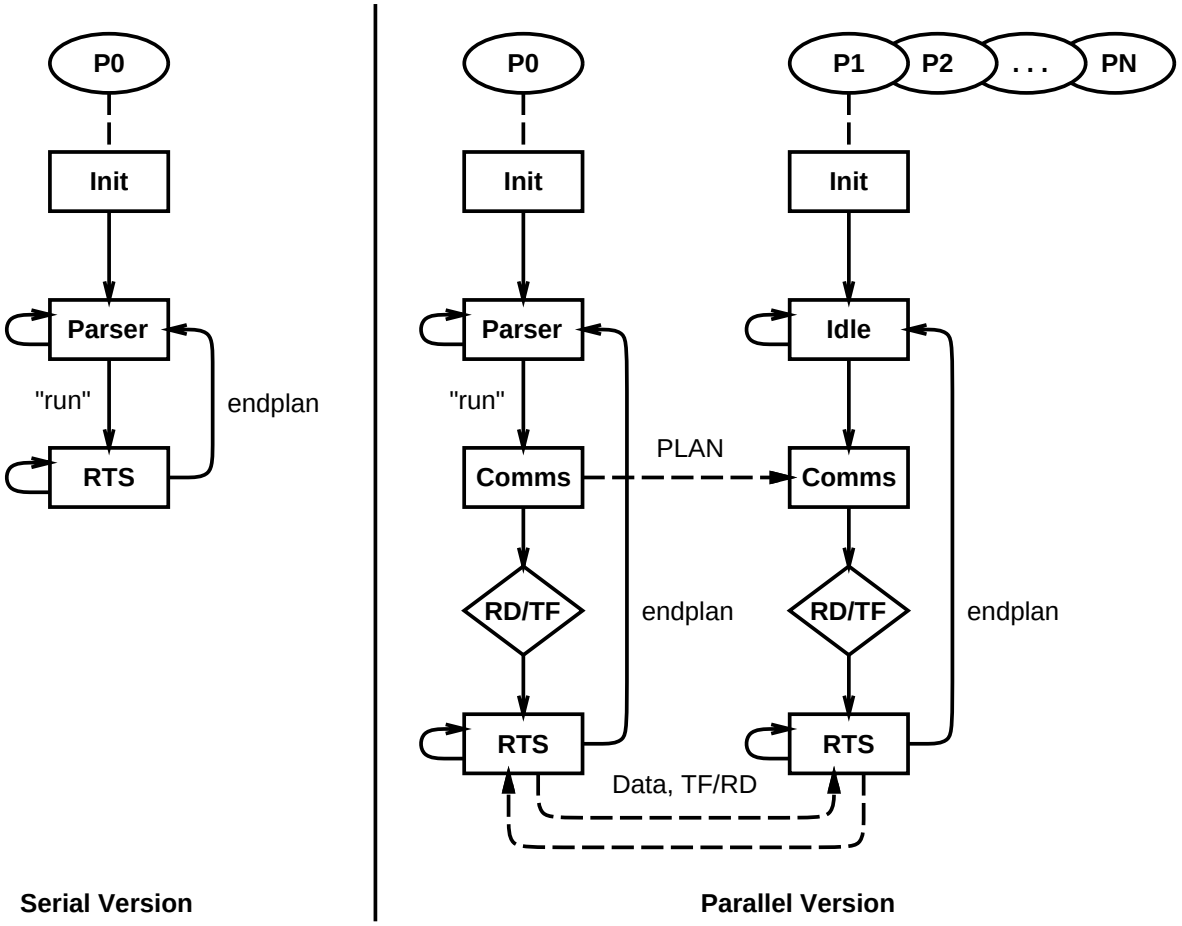


Figure 6: *Simplified Execution Flow.* The simplest mode of execution is the serial framework with a single process, P0, in which actions are processed by the parser, and "compiled" code is executed by the run time system. In parallel operation, the parser runs on a single process, and information about the reproductive plan is shared by communication. A decision about how to execute the plan is made, resulting either in the data space being split across the processes or in compute-intensive parts of the code being task farmed.

place uniformly over various projections of the multi-dimensional space. Each processor can then simply execute the common instructions over its local data, sharing information with neighbouring processes when necessary.

Task farming is supported by the the `forall` construct which allows the user to specify a set of operations which apply to all genomes in population. This is illustrated in the first example plan presented in the appendix.

It was stressed in sections 1 and 3 earlier that a major design aim for RPL2 was that it should impose no constraints on the data structures used to represent solutions. This is achieved by providing a completely generic `genome` data structure which contains only information that any type of genome would have. From the user's point of view, this consists only of the raw fitness value. A generic pointer is then included that references a user-definable data structure, allowing a genome to be completely general. Collections of `genomes` are called `gstacks`, and admit the notion of scaled fitness, relative to other genomes in the group. These data structures are illustrated in figure 7.

5.3 Extending the Framework

It has become clear that real-world applications demand good quality problem-specific genome representations, as discussed in section 3. The RPL2 system leaves the user completely free in

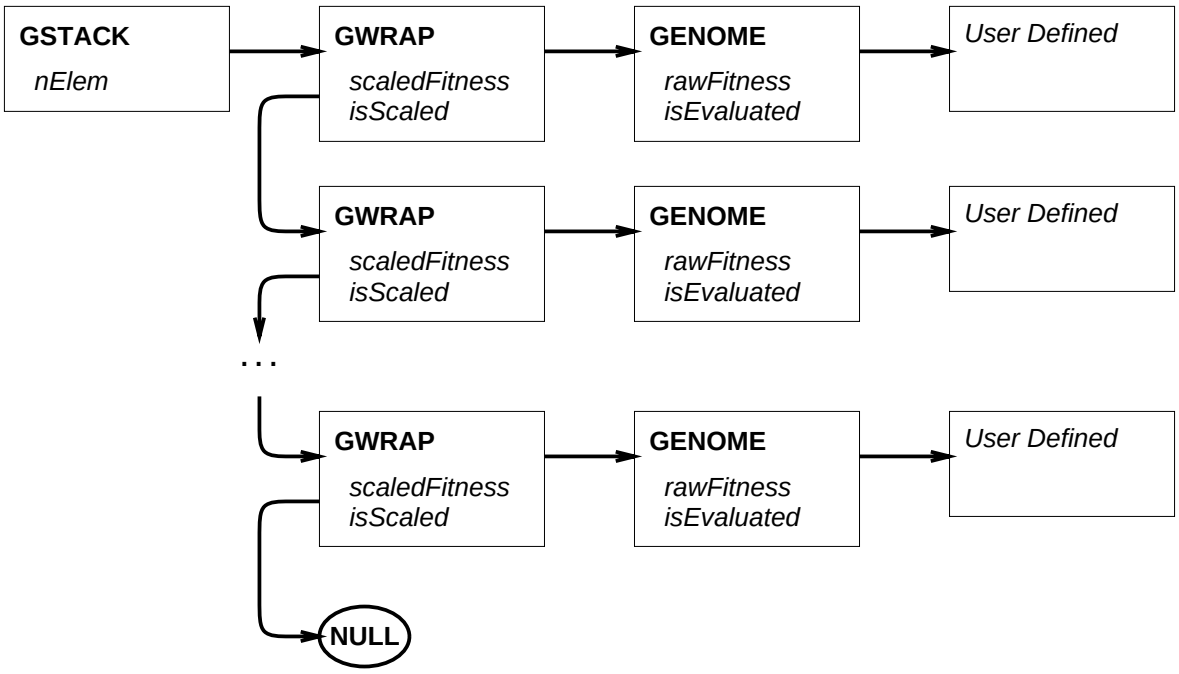


Figure 7: *Construction of a* `gstack`. A `gstack` is made up of a linked list of `gwrap` structures, each of which points at a `genome` (allowing genomes be referenced in more than one stack). A genome has a scaled fitness only in the context of a stack, but has an absolute raw fitness. Each genome also contains a pointer to the (user-defined) representation-dependent data.

his or her choice of representation: the framework works with generic genomes that include a user-defined and user-manipulated component. This makes it equally suitable for all modes of evolutionary computation from genetic algorithms and evolution strategies (Baeck *et al.,* 1991) to genetic programming (Koza, 1992), evolutionary programming (Fogel, 1993) and hybrid schemes.

New operators and new genetic-representations are defined by writing standard ANSI C-callable functions with return values and arguments corresponding to RPL2 data types. A supplied preprocessor (`rpp`) generates appropriate wrapper code to allow the operators to be called dynamically at plan run-time (see the top of figure 8). Operator libraries may optionally include initialisation and exit routines, start-up parameters, and check-pointing facilities, supporting an extremely broad class of evolutionary computation. New representation libraries must also provide routines which allow the framework to pack, unpack and free the user-defined component of a genome in order to permit representation-independent cross-processor communication.

A distinction is made between *representation-independent* operators, whose action depends only on the standard fields of a genome (such as fitness measures), and *representation-dependent* operators, which may manipulate the problem-specific part. Examples of representation-independent operators include selection mechanisms, replacement strategies, migration and deme collection. All "genetic" operators (most commonly recombination, mutation and inversion) are representation dependent, as are evaluation functions, local optimisers and generators of random solutions.

This distinction strongly promotes code re-use as domain-independent operators can form generic libraries. Even representation-dependent operators may have fairly wide applicability since many different problems may share operators at the genetic level: it is only evaluation functions that invariably have to be developed freshly for new problem domains.

Several libraries of operators and representations are provided with the framework, both

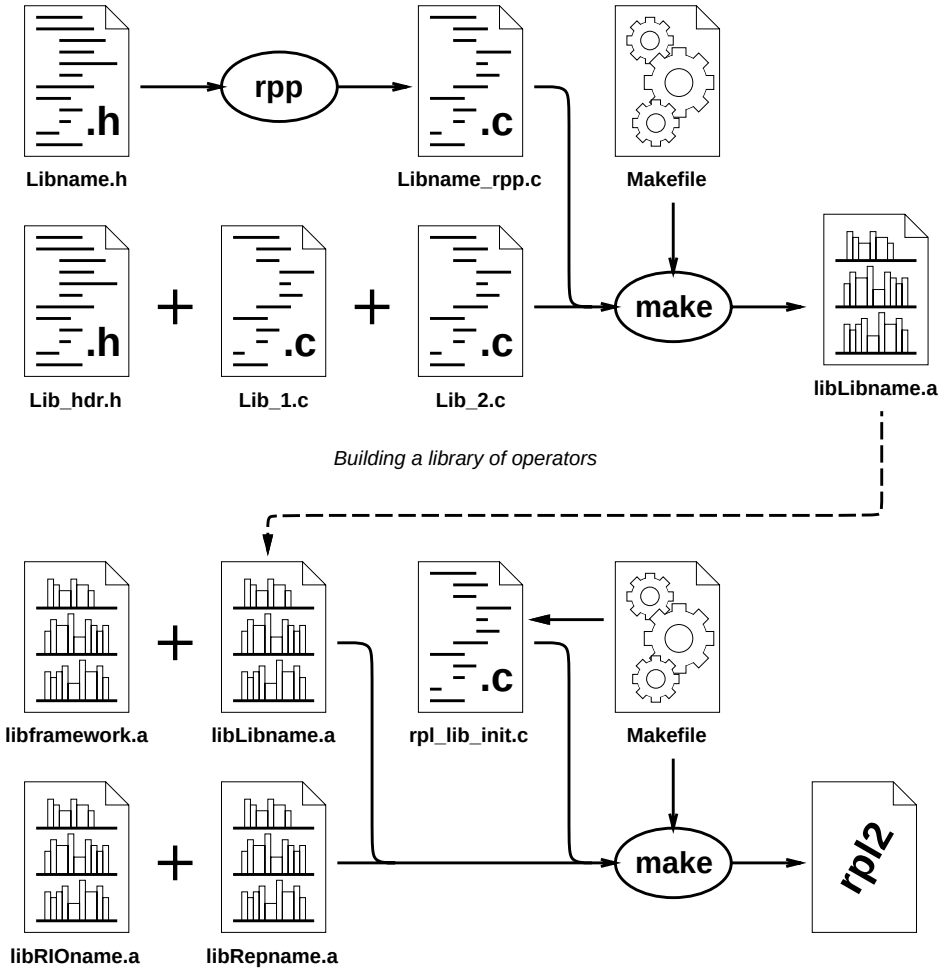


Figure 8: *User Library Management.* In the top half of the figure, `rpp` is used to generate the wrapper functions which interface between the RPL2 framework and the user's C-callable code. This code is compiled to generate a library of object code. Shown below is the process by which a customised version of RPL2 is built. An interface function which calls the initialisation code for each library is generated from the list of libraries in the `Makefile`. This is linked along with the requested libraries and the framework code to produce the executable.

as examples for customisation, and to facilitate quick prototyping and development of applications. The supplied representations currently include real strings, binary strings, variable-cardinality integer strings, sets, and permutations. A number of simple evaluation functions are also included to allow initial familiarisation with the system before tackling a particular application.

Customised versions of the framework are built by linking together whatever combination of operator libraries and representations is desired, allowing locally developed operators to be tested in the context of existing libraries, maintained in some central location. This is shown in the bottom of figure 8.

Contributions of new libraries of operators and representations are solicited and will be considered for inclusion with future general releases of RPL2, allowing even more scope for code sharing and re-use.

6 Applications

6.1 Gas Network Pipe-Sizing

The problem of designing a gas supply network can be broken down into two essential components—defining the routes that the pipes should follow and deciding the diameter of each of the pipes. The choice of route is generally constrained to follow the road network fairly closely and can be achieved efficiently by hand, but the process of selecting the pipe sizes is more complex and requires optimisation.

Other things being equal, thinner pipes are preferable to thicker pipes because they are cheaper, but the pipe network must also satisfy two implicit constraints. The first requires that the network be capable of supplying all customer gas demands at or above a "minimum design pressure". The second, an engineering constraint, requires that every pipe in the network should have at least one other pipe of equal or larger diameter "upstream" of it.

The problem is thus to determine the cheapest pipe network that can be constructed that satisfies the two constraints. EPCC worked with British Gas on this problem, at the same time as designing and implementing RPL2.

In the particular problem considered, the network contained 25 pipes, each of which could be selected from six possible sizes, giving rise to a search space of size $6^{25} \simeq 3 \times 10^{19}$ (about thirty billion billion). The pipes connect 25 nodes, 23 of which are (varying) demand nodes and two of which are pressure-defined source nodes (flow-defined source nodes may also be specified). The network is a real one, that was actually installed (with pipe-sizes determined using the existing heuristic method discussed below).

Genetic Representation

The representation used to represent the pipe sizes in the network is a variable cardinality integer string. A genome is a sequence of N integers $a_1 a_2 \cdots a_N$ with $a_i \in \{0, 1, \ldots, C_i - 1\}$, where C_i is the cardinality (number of alleles) of the ith gene. The particular problem instance tackled happened to have $C_i = 6$ for each pipe, but this is not generally the case.

This library is provided in RPL2 as `IntVC`, and was supplemented for this work by a sub-library of problem specific operators for evaluation and so forth.

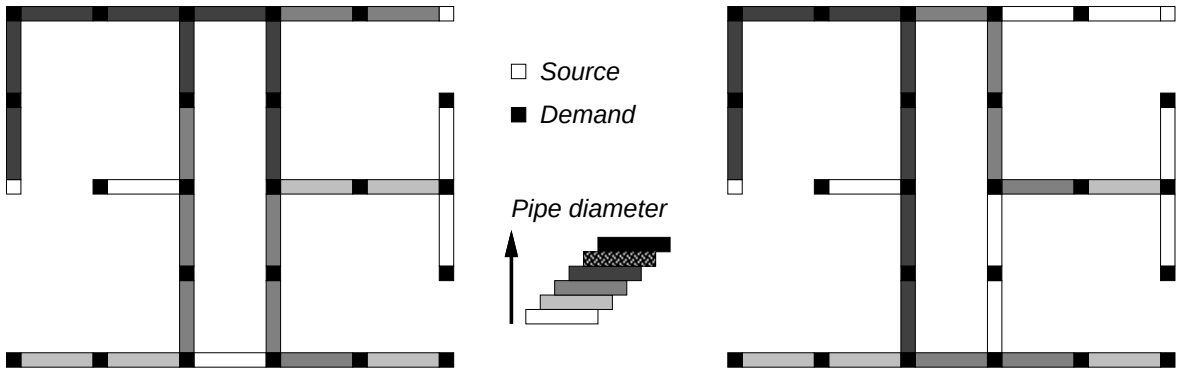


Figure 9: The genetic algorithm finds a solution approximately 4% cheaper (in cash terms) than that found by the heuristic technique and actually used for the installed network. The networks are shown only schematically: the pipes are actually of different lengths. The demand and supply requirements are also different at each node. Both networks are valid in that they satisfy both the upstream-pipe and minimum pressure constraints.

Evaluation Function

The evaluation function used determines the cost of a genome by summing the cost of the pipes making up the network.

The satisfaction or non-satisfaction of the two constraints must however, also be considered. Both the upstream pipe constraint and the minimum pressure constraint are implicit. Their satisfaction can only be determined by solving the non-linear gas flow equations in the network which defines the upstream direction and the pressure at each node in the network. A penalty function is really the only viable approach for handling implicit constraints such as these, since it would be extremely difficult or impossible to construct genetic operators that respected them. Ideas from Richardson *et al.* (1989), Michalewicz & Janikow (1991) and Michalewicz (1993) were used to increase the penalty gradually as a function of generation number.

The form of the cost function used was

$$cost = \sum_{j=1}^{N_{pip}} l_j \cdot c(D_j) + \alpha n_{gen}^{k_1} \cdot (p_{des} - p_{min})^{k_2} + \beta n_{gen}^{k_3} \cdot \sum_i (D_i - D_k)^{k_4} \qquad (2)$$

where the first term is the cost of the pipes as a function of diameter, the second term is the minimum pressure constraint, and the final term is the upstream pipe constraint. In this final term, summation is over pipes i where there is no upstream pipe which is of greater or equal diameter and D_k is the diameter of the largest upstream pipe from pipe i.

Values were selected for the constants α and β that normalised nominal values of the penalties to the same scale as the basic cost of the network. The exponents k_2 and k_4 were selected in order to make the penalties grow at roughly the same rate as networks became "worse" at satisfying the constraints. The annealing parameters k_1 and k_3 were subject to experimentation.

The Reproductive Plan

A fairly conventional reproductive plan was developed to tackle the problem, using elitism to preserve the best member of the population. Because the fitness of a genome depends on penalty terms that vary with the current generation number, the entire population must be re-evaluated at each generation, prior to fitness-based reproduction. Various forms of structured populations were investigated during the problem, using the facilities provided by RPL2. A panmictic (unstructured) population was used in most initial experiments, in order to tune values of parameters, particularly for the fitness function. A fine-grained structure did not yield significant benefits, perhaps due to the relative simplicity of the problem. An island structure was also used, with the main benefit being the ability to run on multiple processors and try longer runs with larger populations. However, this also did not significantly improve performance.

Results

The heuristic technique in previous operational use by British Gas was applied to the problem, in order to compare its performance to the genetic approach. The installed network was actually designed using the results obtained from the heuristic.

The heuristic determines a good configuration of pipe sizes by first assuming a constant pressure drop over the whole network and guessing some initial pipe sizes that will yield a valid but not necessarily optimal network. The heuristic proceeds by locally optimising this solution, repeatedly trying to reduce single pipe diameters while maintaining a valid network. Eventually this process terminates when no pipe size can be reduced while maintaining network validity. The algorithm takes on the order of ten seconds on a 486 PC (25MHz) to reach its "optimal" configuration for the problem under study. A schematic (which does

not represent differences in pipe lengths and source/demand requirements) of this solution appears in the left-hand part of figure 9.

The genetic technique produced consistently good results, although it did not always converge to the same optimal solution. In most cases it found networks which were better, often significantly so, than that determined by the heuristic approach. In almost all cases the algorithm found a valid network by the end of the run (i.e. one in which the penalty terms were zero). Run times for typical populations of 100 networks through 100 generations (testing at most 10,000 different networks) were of the order of several minutes on a Sun SPARC 2 workstation. Note however that the increased cost of this computer time over that of the heuristic is trivial relative to the savings in pipeline construction. The best result was a network approximately 4% cheaper than the heuristic solution. A schematic of this network is shown in the right-hand part of figure 9.

This project has clearly demonstrated to British Gas that genetic algorithms and RPL2 can produce cost savings on real business problems. British Gas now intends to adopt this powerful solution technique for a number of other problems.

6.2 Retail Dealership Location

Geographical Modelling and Planning (GMAP) Ltd. specialises in the planning of efficient delivery networks for goods and services. In particular, GMAP helps its clients to optimise their networks of dealerships or retail outlets. In order to achieve this goal, a mathematical model has been developed that predicts the pattern and volume of business expected from a given distribution network by integrating demographic, geographic and marketing information. The model used—a so-called *spatial interaction model*—provides a highly complex function to be optimised. GMAP's approach had been to use a heuristic to search for dealership networks with high levels of predicted sales within single regions of Britain, of which there are sixteen.

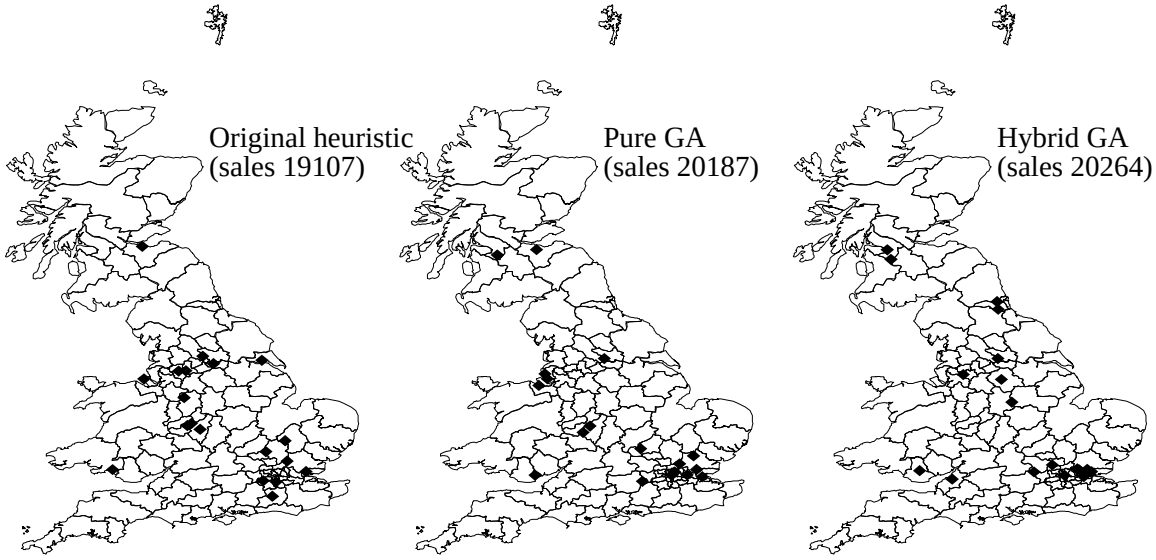
Working collaboratively with GMAP, Felicity George (1994) first produced a parallel implementation of the spatial interaction model and heuristic on a Connection Machine CM-200. This ran some 2,500 times faster than the sequential implementation on a Sun SPARCstation. This reduced the (projected) run-time for the heuristic on the whole of Britain from a matter of a few months to around an hour, and facilitated the tackling of this problem with genetic search techniques.

The prototype implementation of RPL2—RPL Russo (1991)—was used to construct a hybrid genetic algorithm for this problem. RPL ran on the front end of the Connection Machine while the evaluation function (the spatial interaction model) and the heuristic ran on the Connection Machine itself. The problem of choosing the optimal dealer network is naturally formulated as a subset extraction problem (choosing in which of Britain's 8,500 postal districts dealerships should exist) so the forma analysis of set problems reviewed in section 3.5 was exploited. R^3 provided good results, as did a spatial recombination operator designed especially for the problem. The spatial operator exchanges geographical clusters between parents while using directed mutations around boundaries to keep the number of dealers fixed. The results were improved still further, as expected, when the pure genetic algorithm was augmented by incorporating a pseudo-genetic operator based on the original heuristic used by GMAP.

Network Solutions

The three diagrams below show small dealer networks found by the original heuristic technique, by a pure genetic algorithm and by a hybrid genetic algorithm. In this case, the hybrid genetic algorithm produces results some 6% better than those found by the heuristic. Larger networks can also be designed, allowing larger scope for improvements over the heuristic

technique, but are harder to visualise. The investigations carried out covered a wide range of network sizes and in all cases provided resulting networks with predicted sales levels between 5% and 20% higher than those found by the heuristic.



The hybrid genetic algorithm implementation has further advantages over the heuristic. It allows the existing network to be taken as a starting point and perturbed to produce a better network, thus allowing *incremental* improvement to the network as well as the design of complete networks from scratch. Moreover, genetic search allows the construction of a range of high-quality solutions thus permitting the decision maker to choose from a number of good networks. This allows additional decision criteria to be incorporated, including those that may not be straightforward to compute, such as more subjective considerations.

6.3 Data Mining

Large organisations routinely gather vast and ever-increasing amounts of data in the ordinary course of their business. While the information is typically collected with a specific purpose in mind, once collated it also has the potential to be exploited for other purposes. There is increasing interest in the production of systems for performing automatic inductive reasoning conditioned by the data residing in such databases, in the process becoming known as *data mining* (Holsheimer & Siebes, 1994).

As a general concept, data mining is intuitively appealing but rather poorly defined. Many different formulations can be imagined, and each will lead to different detailed goals, forms for the discovered knowledge and—presumably—rates of success.

Data mining has as its goal the discovery and elucidation of interesting and useful patterns within a database. The emphasis is on *novel* patterns, which in this context means patterns that humans find hard to extract either by eye or using standard look-up or statistical techniques. The most useful form in which the results of data mining can be presented is as explicit rules, perhaps expressed as predicates (*if* x *then* y). It is not necessary for rules to be strictly correct in order for them 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.

While data mining in its purest form is thought of as a (relatively) undirected search, in that the subject of the rules to be found is not normally specified, expressing knowledge in the form of rules makes it straightforward to restrict some part of the rule and thus to direct the data miner towards particular kinds of knowledge discovery.

While the discovery of rules is in an intuitive sense a "search" problem, there is much freedom in its casting as a well-defined search or optimisation task. It is clear that the goal is not simply to find the single "best" rule describing the database, not only because "best" is extremely difficult to quantitatively define, but because the goal is to find a selection of rules representing different kinds and instances of patterns within the database. Thus the problem has a *covering* aspect to it. Presumably rules will form clusters and it seems natural to try to collect one (good) representative from each cluster of similar rules surpassing some minimum quality.

In the context of evolutionary computing this raises a host of interesting possible approaches. While traditional genetic algorithms and evolution strategies have most often been applied to strict optimisation problems, there are numerous techniques for encouraging niching and speciation. Current work on data mining at EPCC uses structured population models to encourage niching. The covering ability of the resultant genetic algorithm is then further enhanced by exploiting a two-level *hierarchical* scheme, described below.

The Hierarchical Genetic Algorithm

It has been emphasised already that the goal of data mining is to discover not a single rule but a useful collection of different rules. To achieve this, genetic algorithms operating at two different levels of a hierarchy are used. The "low-level" genetic algorithms searches for individual rules competitively, using a fine-grained structured population to encourage a degree of speciation. A higher level genetic algorithm then takes rules generated by the lower level algorithms and uses them as basic "genetic material" from which to form *sets* of rules using techniques for genetic set-based optimisation discussed in section 3.5.

More precisely, rules are taken from each of the low-level populations to form a *universal set* of rules from which it will be the task of the high-level genetic algorithm to find the "best" set of some given size—for example, the best set of twenty rules. Notice that this is *not* to say that the high-level genetic algorithm seeks to find (say) the twenty best rules: in the high-level genetic algorithm the fitness function is applied to entire sets of rules and it is these sets that compete. The aim is to find the best *collection* of rules with regard to a balance of rules with different characteristics. In this manner, competition at two different levels of the hierarchy results in the discovery of co-operatively useful sets of rules. This is shown schematically in figure 10, and is of general relevance to covering problems: the low-level populations search for good areas of the search space and the high-level population combines these into useful coverages.

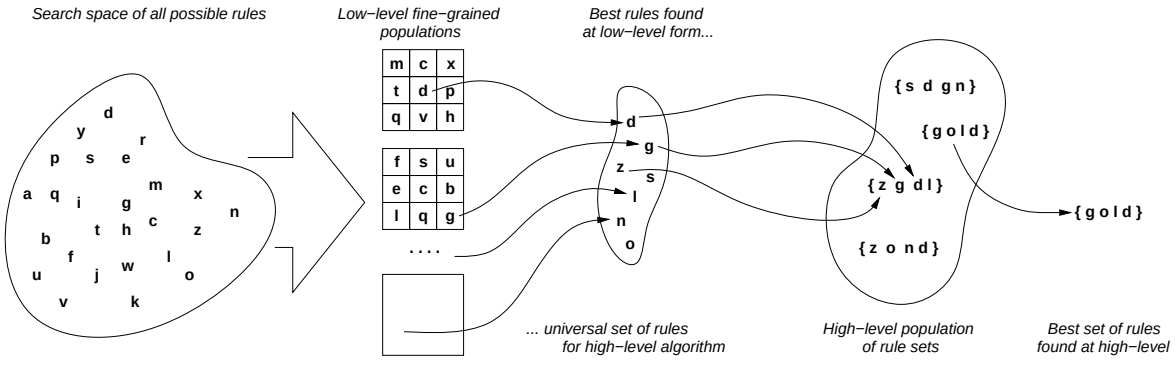


Figure 10: The hierarchical genetic algorithm. The low-level genetic algorithms use structured populations to search competitively for individual rules (or more generally, good candidate elements for a set). The high-level genetic algorithm searches for the best set of rules it can form using those individual rules generated by the low-level genetic algorithms (or, in general, the best set it can form from the elements found by the low-level genetic algorithms).

A suitable evaluation function for the high-level algorithm is arrived at by considering the characteristics sought from collections of rules discovered by the data miner. Clearly each rule should be of high utility, but perhaps more importantly the rules presented ought to be significantly different from one another, covering as many different rule types as possible. In the present implementation coverage has been chosen as the key criterion, but in the longer term the search will be cast as a multi-criterion problem at the higher level.

The Low-Level Genetic Algorithm

The low-level genetic search is used to find a number of interesting characterisations (rules) of the database, to be used in the high-level algorithm to cover the space of interesting rules.

The data used in this work is week-on-week sales data from a major high-street grocery retailer. The database holds information for 156 products over 55 weeks. Data is divided into 12 fields such as price, quantity sold, supplier and promotional information. The database is augmented by weather information corresponding to the sales periods, with figures for weekly average precipitation, minimum and maximum temperature and hours of sun. Each field in the database can be viewed as a function of product and time, as it has a unique value for each such pair. Weather data of course only varies as a function of time, being constant with respect to product.

Fields are distinguished as either dependent or independent. Independent fields are values that are not thought to be functions of other fields in the database, or are otherwise outside the scope of the database. Dependent fields are those which might in principle be functions of other data in the database, and about which the data-miner attempts to theorise. In experiments conducted to date, quantity sold and total value of product sold are the only dependent categories.

Rules

In this work, a rule is an *if then* predicate made up of three parts—the *specificity* (S), a number of *conditional clauses* ($C_1, C_2, \ldots, C_n$), and a *predictive clause* (P). Thus a rule is of form:

$$\text{if } S \text{ and } C_1 \text{ and } C_2 \text{ and } \ldots \text{ and } C_n \text{ then } P \tag{3}$$

The specificity indicates which part of the domain of the database the rule applies to (in terms of products and times). It currently specifies a contiguous range of time values and either a single product or "all products" (e.g. $S = ([40, 50], \text{bananas})$).

The conditional part of the rule is formed by the conjunction (logical and) of a number of simple clauses. Each clause is a linear inequality between one or two fields in the database. For enumerated fields, the clause simply fixes some field (e.g. promotional position := "BIN6"). For two continuous fields, the clause relates their normalised values in an inequality with two arbitrary real constants. The second field in the clause may contain a time lag, and may be with respect to a different product. This allows the data-miner, for instance, to relate the price of apples today to the number of oranges sold two weeks ago (e.g. $\text{price}(\text{apples}, \text{today}) < 1.2\text{price}(\text{oranges}, \text{last week})$).

The prediction part of the rule is a single clause. This clause is identical in form to those in the conditional part of the rule except that one of the fields in the clause must be a dependent category from the database. This excludes rules that theorise about relationships among purely independent variables, which may or may not be true, but are certainly not relevant.

The fitness of a rule is determined by how well it applies to the database, but this evaluation function will not be discussed in detail here.

The Reproductive Plan

Within the hierarchical genetic algorithm, the low-level search produces a number of good quality rules that form the basis for a coverage of the space of all interesting rules. Thus it is

not desirable for the search to converge to the same rule in each low-level population: rather the aim is to cover many local optima.

The reproductive plan uses a fine-grained population structure in order to promote niching in the search for rules. This assists the finding of many good rules. A number of low-level searches are performed, and the results are collected for use as the universal set for the high-level genetic algorithm (see figure 10). In the future, feedback from the high-level search will direct the low-level genetic algorithms to search new regions, or to avoid highly covered areas.

At each step, a child is formed by crossing the current genome with a member of its *deme* (neighbourhood) chosen by binary-tournament selection (with replacement). The child is mutated and evaluated, and possibly replaces the current parent using binary-tournament replacement. After a specified number of generations, the best rule is saved for use in the high-level algorithm, and the process is repeated.

A simple reproductive plan was also written for the high-level search. In current experiments a panmictic (unstructured) population is used to search a universal set of 200 rules for good subsets of size five.

During each generation, the population is incrementally replaced. A child is formed by crossing two parents chosen using binary-tournament selection. The resulting child is mutated and evaluated, and added back into the population using binary-tournament replacement.

After a specified number of generations, the best collection is presented to the user. Statistical measures such as population variance could also be used as stopping criteria.

Results

Results from the technique are still preliminary, as the work is in progress. However, both the low-level and high-level algorithms appear to be functioning correctly and producing at least some interesting results. Some examples rules produced by the low-level search are:

> For golden delicious apples during the final 44 weeks of data, if the average sunshine (in hours) plus 5.4 times the retail price (in pounds) is greater than 9.5 then the total value of of apples sold is greater than £632.
>
> For oranges during the first 50 weeks of data, if the average maximum temperature is greater than $5°C$ and the average rainfall is greater than 0.4 mm then the total value (in pounds) plus 0.7 times the quantity sold five weeks ago will be greater than 423.
>
> For bananas, if the average rainfall is less than 1.5mm then more than 1373 lbs of bananas will be sold.

The higher-level algorithm does tend to produce collections of dissimilar rules with high fitness, as expected. However, further investigation and analysis of the results are required before more detailed conclusions can be drawn.

6.4 Other Applications

A number of other applications have been tackled using RPL2 and its predecessor, RPL. These include the Travelling Sales-rep Problem using a permutation representation with Random Assorting Recombination (Radcliffe, 1994a), stock market tracking with a hybrid scheme based on a set-based representation (Shapcott, 1992) and neural network topology optimisation (Dewdney, 1992). Trivial test problems using binary representations have also been implemented for demonstration purposes.

Availability

The RPL2 framework, along with a variety of common representation libraries, is distributed without charge. The language interpreter and run-time system is provided in object form while source code is provided for the representation libraries, to allow users to study and modify them. The distribution includes high quality manuals, providing not only reference material for RPL2 but general genetic algorithm reference and a tutorial modelled on that of Davis (1991).

Portability across architectures is an important feature of RPL2, allowing exactly the same reproductive plans to be run on a wide variety of platforms. This has been achieved by basing the software on portable tools, both for compilation and inter-process communication. A serial version of the system requires only the widely available compiler generation tools `lex` and `yacc`, and an ANSI-C compiler. EPCC's CHIMP communications interface, and a number of parallel utility library modules have been used in the parallel version.

The CHIMP communications software is currently available on the following systems: Sun SPARCstation, Silicon Graphics Workstation, IBM RS/6000 Workstation, DEC Alpha Workstation, Sequent Symmetry, Meiko Computing Surface 1, Meiko Concerto, and the Meiko Computing Surface 2. It is planned that RPL2 will migrate to the emerging MPI standard for message-passing, to ensure an even wider range of host hardware, including such platforms as: Cray T3D, Fujitsu AP1000, Intel iPSC, Intel Paragon, Thinking Machines CM5, and the IBM SP1.

Inquiries concerning RPL2 are welcomed by email to `rpl2-support@epcc.ed.ac.uk`.

Acknowledgements

The serial prototype of RPL2 was implemented by Claudio Russo (1991), who developed many of the ideas with Nicholas Radcliffe. The parallel prototype was developed by Graham Jones (1992). Mark Green and Ian Boyd from British Gas worked together with Patrick Surry on the design and implementation of RPL2. The work on retail dealership location with Ford was undertaken primarily by Felicity George (EPCC) in collaboration with GMAP Ltd.

Appendix: Example RPL2 plans

Panmictic example

```
% This is a simple panmictic (unstructured) example that illustrates how
% parallelism can be applied to such problems (the forall construct)
% The plan is based on a population/cache model with generational update.

plan(PanmicticExample)

use     StdInst, Binary(128);           % parameterised by string length
string  sFile;
bool    bMaxIsBest;

int     iCounter, nGenerations, i, nPopsize;

genome  gNew,gChild,gParentA,gParentB;
gstack  gsPop, gsCache, gsParents;      % population, cache, and parents

sFile := "stdout";                      % direct output to the terminal
nPopsize := 200;                        % population size
nGenerations := 100;                    % number of generations
bMaxIsBest := TRUE;                     % maximise the evaluation function

Randomize(0);                           % pseudo-random initialisation

for iCounter := 1 to nPopsize           % create initial population
        gNew := RandomGenome();
        Push(gNew,gsPop);
endfor

forall gChild in gsPop                  % parallel evaluation of population
        EvalOneCount(gNew);             % number of 1s in string
endforall

for i := 1 to nGenerations              % generational update scheme

        Empty(gsCache);                 % empty the stack for next generation
        Empty(gsParents);               % empty the array of parents

        ScaleRanked(gsPop, bMaxIsBest, 0.0, 1.0);       % scale on ranking
        SelectScaledSUS(gsPop, 2 * nPopsize, gsParents); % choose all parents

        for iCounter := 1 to nPopsize   % create new generation of genomes
                gParentA := Pop(gsParents);  % do 3pt crossover,
                gParentB := Pop(gsParents);  % with 20% clone rate
                gChild := CrossNpt(gParentA, gParentB, 3, 0.8);
                Push(gChild,gsCache);
        endfor

        forall gChild in gsCache        % mutate and evaluate in parallel
                Mutate(gChild, 0.01);   % bit-wise mutation rate of 1%
                EvalOneCount(gChild);
        endforall

        Swap(gsPop, gsCache);           % swap the new generation for the old

        StatsPrint(i,10,gsPop,sFile);   % collect population statistics
endfor
endplan
```

Hybrid population structure example

```
% In this more complex plan, the population is a set of islands,
% each of which contain a fine-grained population.  Each generation is
% formed by crossing the best and a random genome at each point.

plan(CombinationExample)

% Declare the population structure and defines the deme.  Cyclic (toroidal)
% and no-wrap boundaries are indicated by '@' and ':' respectively.

use StdInst, Set(200);                  % use a 200 element set representation
structure [5:island, 5:island, 10@fine, 10@fine] deme Taxicab(2);

int     iGeneration, nGenerations;
bool    bMaxIsBest;

% These correspond to the entire population structure (5x5x10x10 elements)
gstack  [*,*,*,*]       gsTmp;
genome  [*,*,*,*]       gPop, gMate, gSelf;

% These correspond only to the island axes of the population (5x5 elements)
gstack  [*,*,-,-]       gsImmigrants;
genome  [*,*,-,-]       gBestImmigrant, gEmigrant;

nGenerations := 100;
bMaxIsBest := FALSE;
Randomize(0);                           % pseudo-random initialisation

structfor [*,*,*,*]                     % loop over everything
        gPop := RandomGenome();         % generate a random initial population
        Squash(gPop);                   % simple evaluation function
endstructfor

for iGeneration := 1 to nGenerations    % generational update scheme

        structfor [*,*,-,-]             % loop over the island axes

                structfor [*,*]         % loop over remaining fine-grained axes

                        DemeCollect(gsTmp,gPop);        % collect neighbours
                        gSelf := SelectRawTournament
                                (gsTmp, bMaxIsBest, 2, 0.7, FALSE);
                        gMate := SelectRandom(gsTmp);
                        gPop := rarw(gSelf, gMate, 1);  % RAR-1 operator
                        Mutate(gPop, 0.02);             % set mutation
                        Squash(gPop);                   % evaluate population

                endstructfor

                % possibly migrate the best member of each island randomly
                % and replace a random member with the best immigrant

                gEmigrant := ReduceRawBest(gPop, bMaxIsBest);
                MigrateRandom(gsImmigrants, gEmigrant, 0.1);
                gBestImmigrant := SelectRawBest(gsImmigrants, bMaxIsBest);
                ProjectRandom(gPop, gBestImmigrant);

        endstructfor
endfor
endplan
```

References

Antonisse, 1989. Jim Antonisse. A new interpretation of schema notation that overturns the binary coding constraint. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.

Bäck *et al.*, 1991. Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1991.

Baluja, 1993. Shumeet Baluja. Structure and performance of fine-grain parallelism in genetic search. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1993.

Bridges and E., 1987. C. Bridges and Goldberg D. E. An analysis of reproduction and crossover in a binary-coded genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates (Hillsdale, New Jersey), 1987.

Caruana and Schaffer, 1988. Richard A. Caruana and J. David Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the 5th International Conference on Machine Learning*. Morgan Kaufmann (Los Altos), 1988.

Cohoon *et al.*, 1987. J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. Richards. Punctuated equilibria: a parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates (Hillsdale, New Jersey), 1987.

Cohoon *et al.*, 1990. J. P. Cohoon, W. N. Martin, and D. S. Richards. Genetic algorithms and punctuated equilibria. In H. P. Schwefel and R. Manner, editors, *Parallel Problem Solving From Nature*, pages 134–144. Springer-Verlag, October 1990.

Davidor *et al.*, 1993. Yuval Davidor, Takeshi Yamada, and Ryohei Nakano. The ECOlogical framework II: Improving ga performance at virtually zero cost. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1993.

Davidor, 1991. Yuval Davidor. A naturally occurring niche and species phenomenon: The model and first results. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 257–263. Morgan Kaufmann (San Mateo), 1991.

Davis, 1989. Lawrence Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.

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

Dewdney, 1992. Nigel Dewdney. Genetic algorithms for neural network optimisation. Master's thesis, University of Edinburgh, 1992.

Eshelman and Schaffer, 1992. Larry J. Eshelman and David J. Schaffer. Real-coded genetic algorithms and interval schemata. In Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann (San Mateo, CA), 1992.

Fogel, 1993. David B. Fogel. Evolving behaviours in the iterated prisoner's dilemma. *Evolutionary Computing*, 1(1), 1993.

Fox and McMahon, 1991. B. R. Fox and M. B. McMahon. Genetic operators for sequencing problems. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1991.

George, 1994. Felicity A. W. George. Using genetic algorithms to optimise the configuration of networks of car dealerships. Technical Report EPCC-TR94-05, Edinburgh Parallel Computing Centre, 1994.

Goldberg and Lingle Jr, 1985. David E. Goldberg and Robert Lingle Jr. Alleles, loci and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates (Hillsdale), 1985.

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

Goldberg, 1990. David E. Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. Technical Report IlliGAL Report No. 90001, Department of General Engineering, University of Illinois at Urbana-Champaign, 1990.

Gordon and Whitley, 1993. V. Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimisers. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1993.

Gorges-Schleuter, 1989. Martina Gorges-Schleuter. ASPARAGOS: an asynchronous parallel genetic optimization strategy. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann (San Mateo), 1989.

Gorges-Schleuter, 1990. Martina Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 150–159. Springer Verlag (Berlin), 1990.

Hillis, 1991. W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In Stephanie Forrest, editor, *Emergent Computation*. MIT Press (Cambridge, MA), 1991.

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

Holsheimer and Siebes, 1994. Marcel Holsheimer and Arno Siebes. Data mining: The search for knowledge in databases. Technical Report CS-R9406, CWI Amsterdam, 1994.

Husbands and Mill, 1991. Philip Husbands and Frank Mill. Simulated co-evolution as the mechanism for emergent planning and scheduling. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1991.

Jones, 1992. Graham P. Jones. Parallel genetic algorithms for large travelling salesrep problems. Master's thesis, University of Edinburgh, 1992.

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

Manderick and Spiessens, 1989. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 428–433, San Mateo, 1989. Morgan Kaufmann Publishers.

Marsaglia *et al.*, 1990. G. Marsaglia, A. Zaman, and W. W. Tsang. Toward a universal random number generator. *Statistics and Probability Letters*, 9(1):35–39, 1990.

Maruyama *et al.*, 1993. Tsutomo Maruyama, Tetsuya Hirose, and Akihiko Konagaya. A fine-grained parallel genetic algorithm for distributed parallel systems. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1993.

Mason, 1993. Andrew J. Mason. Crossover non-linearity ratios and the genetic algorithm: Escaping the blinkers of schema processing and intrinsic parallelism. Technical Report Report No. 535b, School of Engineering, University of Auckland, 1993.

Michalewicz and Janikow, 1991. Zbigniew Michalewicz and Cezary Z. Janikow. Handling constraints in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1991.

Michalewicz, 1992. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

Michalewicz, 1993. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag (Berlin), 1993.

Mühlenbein *et al.*, 1991. Heinz Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimiser. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 271–278. Morgan Kaufmann (San Mateo), 1991.

Mühlenbein, 1989. H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 416–421. Morgan Kaufmann (San Mateo), 1989.

Norman, 1988. Michael Norman. A genetic approach to topology optimisation for multiprocessor architectures. Technical report, University of Edinburgh, 1988.

Petty and Leuze, 1989. Chrisila C. Petty and Michael R. Leuze. A theoretical investigation of a parallel genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.

Radcliffe and Surry, 1994. Nicholas J. Radcliffe and Patrick D. Surry. Formal memetic algorithms. In *To appear in AISB94*, 1994.

Radcliffe, 1990. Nicholas J. Radcliffe. *Genetic Neural Networks on MIMD Computers*. PhD thesis, University of Edinburgh, 1990.

Radcliffe, 1991a. Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5(2):183–205, 1991.

Radcliffe, 1991b. Nicholas J. Radcliffe. Forma analysis and random respectful recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 222–229. Morgan Kaufmann (San Mateo), 1991.

Radcliffe, 1992a. Nicholas J. Radcliffe. Genetic set recombination. In Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann (San Mateo, CA), 1992.

Radcliffe, 1992b. Nicholas J. Radcliffe. 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), 1992.

Radcliffe, 1993. Nicholas J. Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications*, 1(1):67–90, 1993.

Radcliffe, 1994a. Nicholas J. Radcliffe. The algebra of genetic algorithms. *To appear in Annals of Maths and Artificial Intelligence*, 1994.

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

Reeves, 1993. Colin R. Reeves. Using genetic algorithms with small populations. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo, CA), 1993.

Richardson *et al.*, 1989. John T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–195. Morgan Kaufmann (San Mateo), 1989.

Russo, 1991. Claudio V. Russo. A general framework for implementing genetic algorithms. Technical Report EPCC–SS91–17, Edinburgh Parallel Computing Centre, University of Edinburgh, 1991.

Shapcott, 1992. Jonathan Shapcott. Genetic algorithms for investment portfolio selection. Technical Report EPCC–SS92–24, Edinburgh Parallel Computing Centre, University of Edinburgh, 1992.

Spiessens and Manderick, 1991. Piet Spiessens and Bernard Manderick. A massively parallel genetic algorithm: Implementation and first analysis. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 279–286. Morgan Kaufmann (San Mateo), 1991.

Starkweather *et al.*, 1990. T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In H. P. Schwefel and R. Manner, editors, *Parallel Problem Solving From Nature*, pages 176–185. Springer-Verlag, October 1990.

Surry and Radcliffe, 1994. Patrick D. Surry and Nicholas J. Radcliffe. *The Reproductive Plan Language RPl2*. Edinburgh Parallel Computing Centre, 1994.

Tanese, 1989. Reiko Tanese. Distributed genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.

Verhoeven *et al.*, 1992. M. G. A. Verhoeven, E. H. L. Aarts, E. van de Sluis, and R. J. M. Vaessens. Parallel local search and the travelling salesman problem. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature, 2*, pages 543–552. Elsevier Science Publishers/North Holland (Amsterdam), 1992.

Vose and Liepins, 1991. Michael D. Vose and Gunar E. Liepins. Schema disruption. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 237–243. Morgan Kaufmann (San Mateo), 1991.

Vose, 1991. Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 1991.

Whitley *et al.*, 1989a. Darrell Whitley, Timothy Starkweather, and Christopher Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. Technical Report CS-89-117, Colorado State University, 1989.

Whitley *et al.*, 1989b. Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1989.