Distributed Coevolutionary Genetic Algorithms for Multi-Criteria and Multi-Constraint Optimisation [*]

Phil Husbands
School of Cognitive and Computing Sciences
University of Sussex
Brighton, UK, BN1 9QH
email: philh@cogs.susx.ac.uk

Abstract

This paper explores the use of coevolutionary genetic algorithms to attack hard optimisation problems. It outlines classes of practical problems which are difficult to tackle with conventional techniques, and indeed with standard 'single species' genetic algorithms, but which may be amenable to 'multi-species' coevolutionary genetic algorithms. It is argued that such algorithms are most coherent and effective when implemented as distributed genetic algorithms with local selection operating. Examples of the successful use of such techniques are described, with particular emphasis given to new work on a highly generalised version of the job shop scheduling problem.

1 Introduction

The vast majority of genetic algorithm (GA) work involves a single 'species'. That is, a single genetic encoding aimed at finding solutions to a single problem. The GA machinery may be configured to work with a single population or, in the case of 'island' models [9], a number of interacting populations. But in either case there is just one evaluation function, and just one solution encoding.

This paper shows how coevolutionary genetic algorithms, employing more than one interacting 'species' evolving under different evaluation functions, can be

[*]From:T. Fogarty (Ed.), Evolutionary Computing, Lecture Notes in Computer Science, Vol. 865, Springer Verlag, pp. 150-165, 1994

used to tackle certain types of hard optimisation problems in a more efficient way than single species GAs.

Particular stress is put on multi-criteria and multi-constraint problems. These are often among the most demanding of optimisation problems and severely test all search techniques. From this class two types of problems are deemed potentially most amenable to coevolutionary techniques.

- Well defined problems with well defined evaluation functions, but which are hugely complex. Sometimes such problems can be redefined in terms of interacting sub-problems. In this paper a scheduling problem is given as an example of a such a task. It is shown how coevolutionary techniques can be usefully applied to it.
- Problems with evaluation functions which are not well defined. Examples are where solutions are desired to perform well over a potentially infinite set of test cases; or when there is no way of weighting different, irresolvable, criteria relative to each other. Hillis's work is given as an example of a coevolutionary approach to such problems [4].

Since they are seen as the basic techniques underpinning sensible coevolutionary GAs, the paper begins with descriptions of distributed and coevolutionary distributed GAs. Hillis's host-parasite coevolutionary system is explained, and its application to practical problems other than those he has explored is outlined. The remainder of the paper looks in detail at the coevolutionary aspects of my work on a highly generalised version of the job shop scheduling problem. Finally general conclusions are drawn.

2 Parallel Distributed GAs

From the very earliest days of its development the GA's potential for parallelisation, with all its attendant benefits of efficiency, has been noted. The availability of hardware has recently allowed significant progress in this direction. The standard sequential GA uses global population statistics to control selection, so the processing bottleneck is evaluation. The earliest parallel models simply parallelised this phase of the sequential algorithm, see, for instance, the paper by Grefenstette [2]. Recently more sophisticated parallel GAs have started to appear in which population can be thought of as being spread out geographically, usually over a 2D toroidal grid. All interactions, e.g. selection and mating, are local, being confined to small (possibly overlapping) neighbourhoods on the grid. Such GAs will be referred to as *distributed* in the remainder of this paper. By doing away with global calculations, it is possible to develop fine-grained highly parallel asynchronous algorithms. There is mounting evidence to suggest that such systems are more robust and faster (in terms of solutions evaluated) than other implementations, e.g. see the articles by Collins & Jefferson [1] and Husbands [5]. Highly parallel models can also result in powerful new ways of approaching optimisation problems at the conceptual level, as will be seen later in this paper.

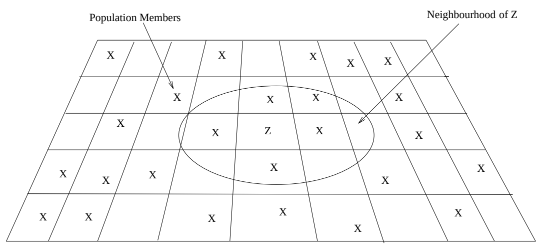

Figure 1: Fraction of typical 2D grid used for distributed GAs.

Typical fine grained distributed GAs use a grid like that shown in Figure 1. Population members are scattered across the grid with no more than one member per cell. Potential mates for a given individual will be found in a neighbourhood centred on the individual. Local selection rules ensure that the most fit members of the population in the neighbourhood are the most likely to be chosen as mates. After breeding, the offspring produced are placed in the neighbourhood of their parents, so genetic material remains spatially local. A probabilistic replacement of the weaker members of the neighbourhood may be employed. By using overlapping neighbourhoods the resulting algorithms generate extremely fierce local selection but allow any improved solutions found to flow around the grid. The effect is to continually stimulate the search process and prevent convergence of the population. A particular distributed algorithm will be described in more detail in Section 5. Further discussion of the parallel implementation of distributed GAs, and their comparison with other forms of parallel GAs, can be found in [5].

3 Coevolutionary Distributed GAs

The algorithms described thus far have, at least implicitly, referred to a single population (or 'species') searching for solutions to a single problem. Coevolutionary algorithms involve more than one 'species' breeding separately, from their own gene pools, but interacting at the phenotypic level.

In Nature the environment of most organisms is mainly made up of other organisms. Hence a fundamental understanding of natural evolution must take into account evolution at the ecosystems level. There are competing theories from evolutionary biology [11], but many are based on Van Valen's Red Queen Hypothesis [12] which states that any evolutionary change in any species is experienced by coexisting species as a change in their environment. Hence there is a continual evolutionary drive for adaptation and counter adaptation. As the Red Queen explained to Alice in Wonderland:

> "... it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!" [From L. Carroll, *Through The Looking Glass*]

It is possible to exploit this phenomenon in GA-based artificial evolution to develop more and more complex competitive behaviours in animats (artificial animals [13]), without having to specify complicated evaluation functions [8]. As we will see, coevolutionary GAs can also be used to tackle difficult optimisation problems of the kinds outlined in the introduction to this paper.

How coevolutionary GAs should be implemented must, of course, at least in part depend on the application. But there are three obvious contenders.

- Separate sequential (or parallel) GAs for each species where the evaluation functions used in each somehow takes into account interactions with the other populations.
- A parallel 'island' implementation in which each population evolves in isolation with occasional interactions with members from other species.
- Distributed GAs for each species where the different populations are spread out over the *same* grid. Interactions between populations are (spatially) local.

In the first model it is difficult to maintain the kind of coherent coevolution required. Which members of any given population are coevolving with given members of another population? All with all? All members of any given population with randomly chosen members of all other populations? How can the population dynamics from generation to generation be controlled to maintain a consistent coevolutionary system? Negative experiences with such an implementation will be described later (see Section 5.1).

The second model will be adequate only if the inter-population interactions desired really are weak, otherwise it will give a misleadingly benign picture of the effects of the different species on each other's environments.

However, the third model gives a very clear and coherent implementation of coevolution. Each species interact locally with its own population but also with the members of the other species in its neighbourhood. Since all offspring appear in their parents' neighbourhoods, a consistent coevolutionary pressure emerges. Examples of such algorithms will be discussed in the following sections.

4 Parasites and Sorting Networks

Danny Hillis, who lead the team that developed the Connection Machine [3] , was the first to significantly extend the parallel GA paradigm by showing how to develop a more powerful optimisation system by making use of coevolution [4]. Using a distributed coevolutionary GA he had considerable success in developing sorting networks. In this extended model there are two independent gene pools, each evolving according to local selection and mating. One population, the hosts, represents sorting networks, the other, the parasites, represents test cases. Interaction of the populations is via their fitness functions. The sorting networks are scored according to the test cases provided by the parasites in their immediate vicinity. The parasites are scored according to the number of tests the network fails on.

Hillis's two species technique can be applied directly to many engineering optimisation problems where the set of test cases is potentially infinite. The most appropriate range and difficulty of evaluation tests can be coevolved with problem solutions as in the work described here. Another use of the technique, being explored at Sussex, is in the coevolution of problem solutions and sets of constraints to apply in the evaluation of these solutions. This is applicable where it is difficult, or impossible, to weight constraints relative to each other, and where there are too many constraints for it to be feasible to have them all active in all parts of the search space.

5 Coevolution, Arbitrators and Emergent Scheduling

This section outlines the use of a multi-species coevolutionary GA to handle a highly generalised version of Job Shop Scheduling (JSS). It is based on the notion of the manufacturing facility as a complex dynamical system akin to an ecosystem. Space does not allow a full description of the cost functions and encodings, such engineering and mathematical details can be found in an earlier paper [6]. This paper concentrates on previously unpublished details of the distributed genetic algorithms used.

The traditional academic view of JSS is shown in Figure 2. A number of *fixed* manufacturing plans, one for each component to be manufactured, are interleaved by a scheduler so as to minimise some criteria such as the total length

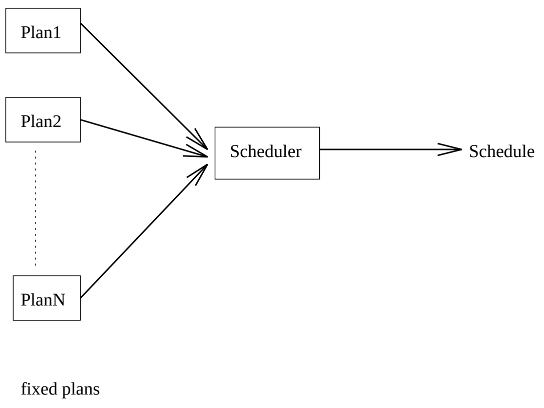

Figure 2: Traditional academic approach to job shop scheduling.

of the schedule. However, a problem that would often be more useful to solve is that illustrated in Figure 3. Here the intention is to optimise the individual manufacturing plans *in parallel*, taking into account the numerous interactions between them resulting from the shared use of resources. This is a much harder and far more general problem than the traditional JSS problem. In many manufacturing environments there is a vast number of legal plans for each component. These vary in the number of manufacturing operations, the ordering of the operations, the machines used for each operation, the tool used on any given machine for a particular operation, and the orientation of the work-piece (setup) given the machine and tool choices. All these choices will be subject to constraints on the ordering of operations, and technological dependencies between operations. Optimising a single process plan is an NP-hard problem. Optimising several in parallel requires a powerful search technique. This section presents a promising GA-based method for tackling the problem. This paper will not delve very deeply into the problem-specific technbical details, instead it will concentrate on GA issues. Much very useful work has been done in the application of GAs to scheduling problems. This will not be discussed here, since this paper is not intended to be specifically about scheduling. However, see [6, 10] for discussions of related work.

The idea behind the ecosystems model is as follows. The genotype of each specie represents a feasible manufacturing (process) plan for a particular component to be manufactured in the machine shop. Separate populations evolve under the pressure of selection to find near-optimal process plans for each of the components. However, their fitness functions take into account the use of shared resources in their common world (a model of the machine shop). This means that without the need for an explicit scheduling stage, a low cost schedule will emerge at the same time as the plans are being optimised.

Data provided by a plan space generator (complex and beyond the scope of this

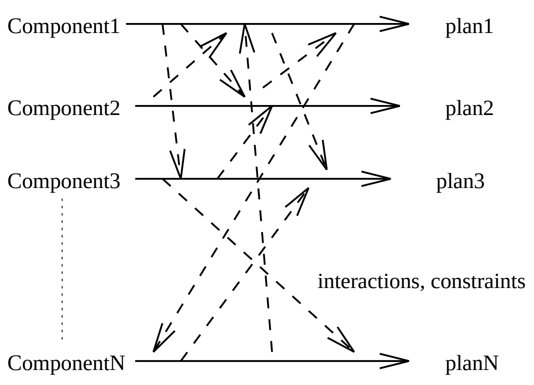

Figure 3: Parallel plan optimisation leading to emergent scheduling.

paper, see [6]) is used to randomly construct initial populations of structures representing possible plans, one population for each component to be manufactured. An important part of this model is a population of Arbitrators, again initially randomly generated. The Arbitrators' job is to resolve conflicts between members of the other populations; their fitness depends on how well they achieve this. Each population, including the Arbitrators, evolve under the influence of selection, crossover and mutation.

Each process plan 'species' uses the plan encoding described in detail in [6], that paper also describes the special genetic operators used with the encoding, and the machining cost functions used in the algorithms given later.

The Arbitrators are required to resolve conflicts arising when members of the other populations demand the same resources during overlapping time intervals. The Arbitrators' genotype is a bit string which encodes a table indicating which population should have precedence at any particular stage of the execution of a plan, should a conflict over a shared resource occur. A conflict at stage L between populations K and J is resolved by looking up the appropriate entry in the Lth table. Since population members cannot conflict with themselves, and we only need a single entry for each possible population *pairing*, the table at each stage only needs to be of size $N(N-1)/2$, where N is the number of separate component populations. As the Arbitrators represent such a set of tables flattened out into a string, their genome is a bit string of length $SN(N-1)/2$, where S is the maximum possible number of stages in a plan. Each bit is uniquely identified with a particular population pairing and is interpreted according to the function given in Equation 1.

$$f(n_1, n_2, k) = g\left[\frac{kN(N-1)}{2} + n_1(N-1) - \frac{n_1(n_1+1)}{2} + n_2 - 1\right] \qquad (1)$$

Where n_1 and n_2 are unique labels for particular populations, $n_1 < n_2$, k refers to the stage of the plan and $g[i]$ refers to the value of the ith gene on the Arbitrator genome. If $f(n_1, n_2, k) = 1$ then n_1 dominates, else n_2 dominates.

By using pair wise filtering the Arbitrator can be used to resolve conflicts between any number of different species. It is the Arbitrators that allow the scheduling aspect of the problem to be handled. In general, a population of coevolving Arbitrators could be used to resolve conflicts due to a number of different types of operational constraint, although their representation may need to increase in complexity.

It should be noted that in early versions of the work to be described, the Arbitrators were not used. Instead fixed population precedence rules were applied. Not surprisingly, this and similar schemes were found to be too inflexible and did not give good results. Hence the Arbitrator idea was developed and has proved successful.

5.1 Early MIMD Implementation

The first implementation of the basic ecosystems model was described in [7]. It used a set of interacting sequential genetic algorithms and was implemented on a MIMD parallel machine as well as on a conventional sequential machine. On each cycle of the algorithm each population was ranked according to cost functions taking into account plan efficiency. The concurrent execution of *equally ranked* plans from each population was then simulated. The simulation provided a final cost taking into account interactions between plans for different components. Conflicts were resolved by an equally ranked Arbitrator. This final cost was used by the selection mechanism in the GAs.

Although promising results were achieved with this model, it suffered from population convergence and little progress was made after a few hundred generations, despite many attempts to cure this problem. It was also felt that the implementation was over complicated and lacked coherence at some levels, resulting in some of the problems discussed earlier in Section 3. The ranking process appears to facilitate coevolution to some extent. But, since the populations are continually reordered there is little continuity, from generation to generation, in the members being costed together until population members are very similar. This may have indirectly been one of the causes of strong convergence. For these reasons this implementation was abandoned and another much more coherent version developed. This is based on the kind of geographically distributed GA mentioned earlier.

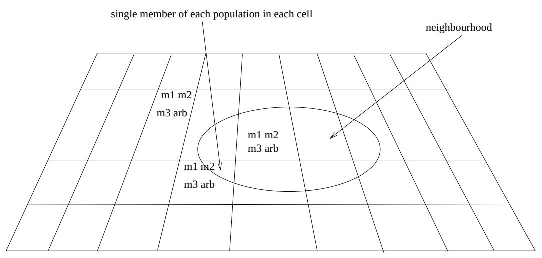

Figure 4: Distributed interacting populations.

5.2 Distributed Implementation

In the second implementation, the cost, hence selection, functions for plan organisms again involve two stages, but for Arbitrators now just one. The first stage involves population specific criteria (basic machining costs), as before, and the second stage again takes into account interactions between populations. Arbitrators are only costed at the second stage. Again the second phase of the cost function involves simulating the simultaneous execution of plans derived from stage one. The process plans and Arbitrators are costed as before, it is the way in which the cost functions are used within the GA machinery which is now quite different.

This second, more satisfactory, implementation spreads each population 'geographically' over *the same* 2D toroidal grid, this is illustrated in Figure 4. Each cell on the grid contains exactly one member of each population. Selection is local, individuals can mate only with those members of their own species in their local neighbourhood. Following Hillis [4], the neighbourhood is defined in terms of a Gaussian distribution over distance from the individual; the standard deviation is chosen so as to result in a small number of individuals per neighbourhood. Neighbourhoods overlap allowing information flow through the whole population without the need for global control. Selection works by using a simple ranking scheme within a neighbourhood: the most fit individual is twice as likely to be selected as the median individual. Offspring produced replace individuals from their parents' neighbourhood. Replacement is probabilistic using the inverse scheme to selection. In this way genetic material remains spatially local and a robust and coherent coevolution (particularly between Arbitrators and process plan organisms) is allowed to unfold. Interactions are also local: the second phase of the costing involves individuals from each population *at the same location on the grid.* This implementation consistently gave better faster results than the first. The notion of coevolution is now much more coherent; by doing away with the complicated ranking mechanism, and only using local selection, based on a concrete model of geographical neighbourhood, the problems and inconsistencies of the first implementation are swept away.

The overall algorithm is quite straightforward. It can be implemented sequentially or in a parallel asynchronous way, depending on hardware available.

Overall()

```
1. Randomly generate each population, put one member of each population in each cell of a toroidal grid.
2. Cost each member of each population (phase1 + phase2 costs). Phase2 cost are calculated by simulating the concurrent execution of all plans represented in a given cell on grid, any resource conflicts are resolved by Arbitrator in that cell.
3. i = 0.
4. Pick random starting cell on the toroidal grid.
5. Breed each of the representatives of the different populations found in that cell.
6. If all cells on the grid have been visited Go to 7. Else move to next cell,Go to 5.
7. If $i <$ MaxIterations, i = i + 1, Go to 4. Else Go to 8.
8. Exit.
```

The breeding algorithm, which is applied in turn to the members of the different populations, is a little more complicated.

Breed(current_cell,current_population)

```
1. i = 0.
2. Clear NeighbourArray
3. Pick a cell in neighbourhood of current_cell by generating x and y distances (from current_cell) according to a binomial approximation to a Gaussian distribution. The sign of the distance (up or down, left or right) is chosen randomly (50/50).
4. If the cell chosen is not in NeighbourArray, put it in NeighbourArray, i = i+1, Go to 5. Else Go to 3.
5. If $i <$ LocalSelectionSize, Go to 3. Else Go to 6.
6. Rank (sort) the members of current_population located in the cells recorded in NeighbourArray according to their cost. Choose one of these using a linear selection function.
```

7. **Produce offspring using the individual chosen in 6 and current_population member in current_cell as the parents.**
8. **Choose a cell from ranked NeighbourArray according to an inverse linear selection function. Replace member of current_population in this cell with offspring produced in 7.**
9. **Find phase one (local) costs for this new individual (not necessary for Arbitrators).**
10. **Calculate new phase two costs for all individuals in the cell the new individual has been placed in, by simulating their concurrent execution. Update costs accordingly.**
11. **Exit.**

The binomial approximation to a Gaussian distribution used in step 3, falls off sharply for distances greater than 2 cells, and is truncated to zero for distances greater than four cells.

In the results reported here a 15×15 grid was used, giving populations of size 225.

5.3 Results of Distributed Implementation

Results of a typical run for a complex 5 job problem shown in Figure 5. Each of the components needed between 20 and 60 manufacturing operations. Each operation could be performed, on average, on six different machines using 8 possible setups. There were many constraints on and between plans, but there were still a very large number of possible orderings of the operations within each plan. The graph at the top shows the best 'total factory cost' found plotted against the number of plan evaluations on the X axis. The graph represents the average of ten runs. The 'total factory cost' is calculated per grid cell by summing the costs of all the process plans (exactly one per component), including the waiting-time costs, represented at that cell. The lower left Gantt chart shows the loading of the machines (M0-M12) in the job shop early on in a typical run, and that at the right shows the loading towards the end of the same run. Time, in arbitrary units, is shown on the horizontal axis. A very tight schedule *and* low cost plans for each of the components are obtained. Figure 6 shows the state of the geographical grid in terms of the 'total factory cost' at each cell. The aim of the overall simultaneous plan optimisation problem is to find the best cell, i.e. the cell with the lowest 'total factory cost'. This will not necessarily contain copies of the *individually* lowest cost members of each population, but it will contain that set of plans, one from each population, that interact in the most favourable way. It can be seen from the figure that the initial random

Figure 5: Results of distributed coevolution model with large problem. See text for explanantion.

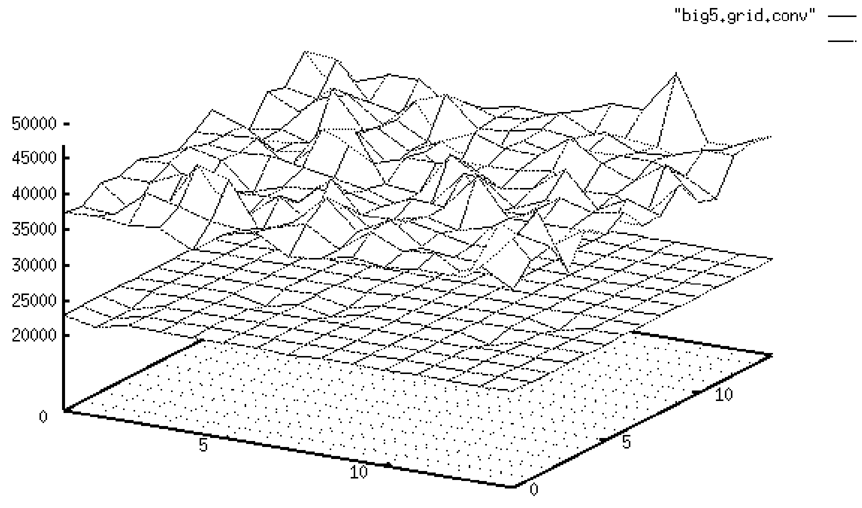

Figure 6: Geographical grid states for large problem.

populations give a spread of poor 'total factory costs', this is rapidly reduced (a few thousand function evaluations) to a set of very good costs spread throughout the whole grid. These are shown more clearly in Figure 7, but note that grid states have not converged. In much longer runs this was still found to be the case. Results very close to those shown here were found on each of 50 runs of the system. Far more consistent behaviour than was found with the earlier implementation.

By replacing the Arbitrators in low cost cells with randomly generated ones, higher cost schedules were produced demonstrating that the Arbitrators are coevolving to make good decisions over all the stages of the plans.

Very promising preliminary results have been obtained for this complex optimisation problem. Although the search spaces involved are very large, this method has exploited parallelism sufficiently to produce good solutions. Most scheduling work deals with a problem rather different from that tackled here, so a direct comparison is very difficult. However, Palmer [10] has recently tackled the generalised problem using a simulated annealing based technique. He demonstrated that his technique had significant advantages over traditional scheduling for a class of manufacturing problems a little simpler than those used here. A direct comparison of the GA and simulated annealing methods is currently underway.

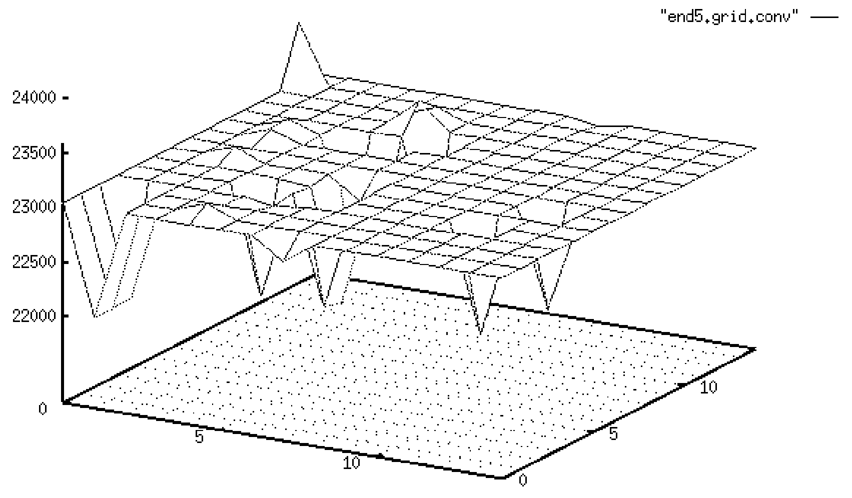

Figure 7: Expanded view of final geographical grid states for large problem.

5.4 Dynamics and Noise

Very important properties of manufacturing environments, namely dynamics and noise, are not handled at all well by most classical JSS techniques. The distributed coevolutionary method can handle both in a natural way. The dynamics of a job shop include such things as machines breaking down, job priorities changing, jobs starting at different times, job due dates changing. These changes are modelled directly in the cost functions and the whole 'ecosystem' reconfigures. The distributed implementation allows the rapid flow of re-adapted solutions throughout the grid, the unconverged state of the populations facilitates the re-adaptation. Noise arises from the fact that the manufacturing processes are not perfectly reliable and deterministic. This means that in reality all cost functions must have stochastic elements. Again this can be handled in a straightforward manner by using evaluation functions which require good performance over distributions of random variables representing such things as machining times. Hence a solution which is very sensitive to *exact* timings of manufacturing processes will be selected against.

6 Conclusions

This paper has advocated the use of coevolutionary genetic algorithms to tackle certain types of very hard optimisation problems. Examples of the successful use of such techniques to sorting network design and generalised job shop scheduling were described. It was argued that coevolutionary genetic algorithms are most coherent when implemented as distributed systems with local selection rules operating. These coevolutionary extensions to traditional GAs may well considerably strengthen our armoury of techniques to be applied against real-world optimisation problems.

Acknowledgements

This work was supported by EPSRC grant GR/J40812. Thanks to two anonymous referees for some useful comments.

References

[1] R. Collins and D. Jefferson. Selection in massively parallel genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth Intl. Conf. on Genetic Algorithms, ICGA-91*, pages 249–256. Morgan Kaufmann, 1991.

[2] J. Grefenstette. Parallel adaptive algorithms for function optimisation. Technical Report CS-81-19, Compt. Sci., Vanderbilt University, 1981.

[3] W.D. Hillis. *The Connection Machine.* MIT Press, 1985.

[4] W.D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[5] P. Husbands. Genetic algorithms in optimisation and adaptation. In L. Kronsjo and D. Shumsheruddin, editors, *Advances in Parallel Algorithms*, pages 227–277. Blackwell Scientific Publishing, Oxford, 1992.

[6] P. Husbands. An ecosystems model for integrated production planning. *Intl. Journal of Computer Integrated Manufacturing*, 6(1&2):74–86, 1993.

[7] Philip Husbands and Frank Mill. Simulated co-evolution as the mechanism for emergent planning and scheduling. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth Intl. Conf. on Genetic Algorithms, ICGA-91*, pages 264–270. Morgan Kaufmann, 1991.

[8] Geoffrey F. Miller and D. T. Cliff. Protean behavior in dynamic games: Arguments for the co-evolution of pursuit-evasion tactics. In D. Cliff, P. Husbands, J.A. Meyer, and S. Wilson, editors, *Animals to animats 3: Proceedings of the third international conference on simulation of adaptive behavior.* MIT Press, 1994.

[9] H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17, 1991.

[10] G. Palmer. *An Integrated Approach to Manufacturing Planning.* PhD thesis, University of Huddersfield, 1994.

[11] N.C. Stenseth. Darwinian evolution in ecosystems: the red queen view. In P. Greenwood, P. Harvey, and M. Slatkin, editors, *Evolution: Essays in honour of John Maynard Smith*, pages 55–72. Cambridge University Press, 1985.

[12] L. Van Valen. A new evolutionary law. *Evolutionary Theory*, 1:1–30, 1973.

[13] S.W. Wilson. Knowledge growth in an artificial animal. In J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their applications.* Lawrence Erlbaum Assoc., 1985.