Husbands, P. and Mill, F. Simulated Co-Evolution as The Mechanism for Emergent Planning and Scheduling. Proc. 4th Int. Conf. on Genetic Algorithms, ICGA'91, Belew, R. & Booker, L. (eds), Morgan Kaufmann, 1991, 264-270

Simulated Co-Evolution as The Mechanism for Emergent Planning and Scheduling

Philip Husbands
School of Cognitive & Computing Sciences
University of Sussex
Falmer, Brighton, England, UK, BN1 9QH

Frank Mill
Dept. Mechanical Engineering
University of Edinburgh
The King's Buildings, Edinburgh EH9 3JL, Scotland

Abstract

The underlying structure of many combinatorial optimisation problems of practical interest is highly parallel. However, traditional approaches to these problems tend to use mathematical characterisations that obscure this inherent parallelism. By contrast, the use of biologically inspired models casts fresh light on a problem and may lead to a more general characterisation which clearly indicates how to exploit parallelism and gain better solutions. This paper describes a model based on simulated co-evolution that has been applied to a highly generalised version of the manufacturing scheduling problem, a problem previously regarded as too complex to tackle. Results from an implementation on a parallel computer are given.

1 Introduction

There is a very large body of work on solving planning and scheduling problems, mainly emanating from the fields of Artificial Intelligence and Operations Research. Traditional AI approaches have had limited success in real-world applications, indeed their shortcomings have been thoroughly explored and documented [2]. The general resource planning, or scheduling, problem is well known to be NP-Complete [5]. Consequently OR techniques have been developed to give exact solutions to restricted versions of the problem, but in general there is a reliance on heuristic-based methods. Because of the complexity and size of the search spaces involved, a number of simplifying assumptions are always used in practical applications. These assumptions are now implicit in what have become the standard problem formulations. The authors hold the view that in many instances this has led to the most general underlying optimisation problem being ignored or, more often, not even being acknowledged as existing at all.

This paper will concentrate on the domain of manufacturing planning and scheduling. In this domain the true relationship between planning and scheduling appears to have been lost sight of long ago. Scheduling is essentially seen as the task of finding an optimal way of interleaving a number of plans which are to be executed concurrently and which must share resources. The implicit assumption is that once planning has finished scheduling takes over. In fact there are many possible choices for the sub-operations in most planning problems. Very often the real optimisation problem is to *simultaneously* optimise the individual plans and the overall schedule. This paper describes how manufacturing planning has been radically recast to allow solutions to the simultaneous plan and schedule optimisation problem, a problem previously considered too hard to tackle at all. A model based on simulated co-evolution is described and it is shown how complex interactions are handled in an emergent way. Results from an implementation on a parallel machine are reported.

Although this paper is largely focused on one particular optimisation problem, it should be noted that the model presented can be generalised. This work is concerned with using parallel GA search in a form of distributed problem solving. Most previous parallel GA work has been concerned with speed up and devising parallel implementations which provide a more robust search [14,15]. In contrast, the work reported here is concerned with using parallel GA search to simultaneously solve interacting subproblems. From this emerges the solution to some wider more complex problem. The idea is to recast a highly complex problem in terms of the cooperative and simultaneous solution of a number of simpler interacting subproblems. Using this variation of divide and conquer, the inherent parallelism in a problem is brought out and thoroughly exploited. The model involves a number of separate populations each evolving using a GA. The genotype for each population is *different* and represents a solution to one of the subproblems. Because the fitness of any individual in any population takes into account the interactions with members of other populations, the separate species co-evolve in a shared world. In this model, possible conflicts between species (e.g. disputes over shared resources) are decided by a further *co-evolving* species, the Arbitrators. The Arbitrators evolve under a pressure to make decisions that benefit the whole ecosystem (cooperative distributed solution to the overall problem). Without explicitly encoding the overall problem, the Arbitrators are used to try and adhere to the global constraints demanded by the problem.

Further details should become clear on reading about the specific example described here. A drive behind this work was to find robust techniques for tackling extremely large combinatorial optimisation problems. Heuristic-free search (as provided by GAs) is of great interest because non-brittle heuristics are very hard to come by in many practical problems. It should be noted that we are describing problems with unimaginably huge search spaces (far larger than the number of particles in the universe) so 'solution' does not necessarily been global optimum. Of course, there are no know guaranteed methods for finding optimums in these sorts of problems.

2 Domain of Application: Manufacturing Planning and Scheduling

Consider a manufacturing environment in which *n* jobs or items are to be processed by *m* machines. Each job will have a set of constraints on the order in which machines can be used and a given processing time on each machine. The jobs may well be of different lengths and involve different subsets of the *m* machines. The job-shop scheduling problem is to find the sequence of jobs on each machine in order to minimise a given objective function. The latter will be a function of such things as total elapsed time, weighted mean completion time and weighted mean lateness under the given due dates for each job [3]. In the standard model process planning directly proceeds the scheduling. A process plan is a detailed set of instructions on how to manufacture each part (process each job). This is when decisions are made about the appropriate machines for each operation and any constraints on the order in which operations can be performed [1]. Very often completed process plans are presented as the raw data for the scheduler. However, in many manufacturing environments there are a vast number of legal plans for each component. These vary in the orderings between operations, the machines used, the tools used on any given machine and the orientation of the work-piece on any given machine. They will also vary enormously in their costs. Instead of just generating a reasonable plan to send off to the scheduler, it is desirable to generate a near optimal one. Clearly this cannot be done in isolation from the scheduling: a number of separately optimal plans for different components might well interact to cause serious bottle-necks. Because of the complexity of the overall optimisation problem, that is simultaneously optimising the individual plans and the schedule, and for the reasons outlined in the introduction, up until now very little work has been done on it. However, recasting the problem to fit an 'ecosystem' model of co-evolving organisms has provided a solution. Partly because of the power of the central optimisation technique (genetic algorithms) and partly because the recasting has allowed many of the complex interactions inherent in the problem to be represented in a simple and natural way.

3 Overview of Approach

This paper concentrates on one core aspect of a complete framework for dealing with a certain class of planning problems. To give the reader a clearer understanding of the context of the work described here, the overall approach is briefly presented. This is captured, at a very high level, in figure 1.

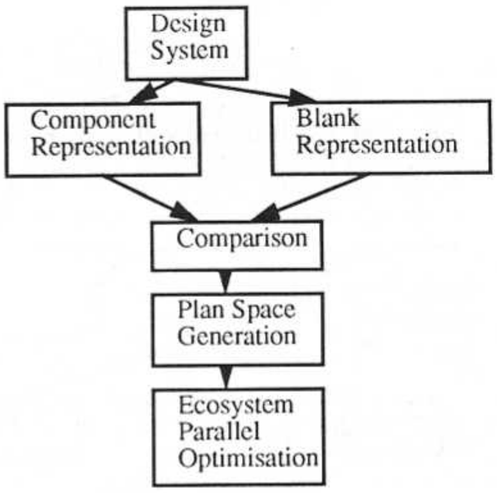

Figure 1: Information Processing Flow in System

A design system, whose description is outside the scope of this paper, produces component and blank representations. These representations are compared in order to find out which component features are to be machined and which, if any, already exist in the blank. The complete space of plans for each component is implicitly generated. These spaces are searched in parallel, taking into account interactions between and within plans, using an ecosystem model. From this emerges a solution to the simultaneously optimal plans and schedule problem. The earlier, knowledge based, parts of the system determine the boundaries and structure of the search space that the emergent optimisation techniques work in. For further details of other aspects of the system see [11,12].

In order to understand the interpretation of the genomes described later, a few more words need to be said about the plan space generation. This is done by a knowledge-based system, which breaks down the manufacture of a component into a number of nearly independent operations. The entire space of possible plans can then be generated by finding all the possible ways to carry out each operation, along with ordering constraints. The execution of an operation is defined in terms of a <machine/process/tool/setup> tuple. The first three fields indicate how to use the machine and the fourth refers to the orientation of the work-piece (partially completed component). The output from this process is a large number of interconnected networks like the one in figure 2. A manufacturing process for the sub-goal described by the fragment of network shown is a route from the starting conditions node to the goal conditions node. Implicit in the representation are functional dependencies and ordering constraints between sub-operations.

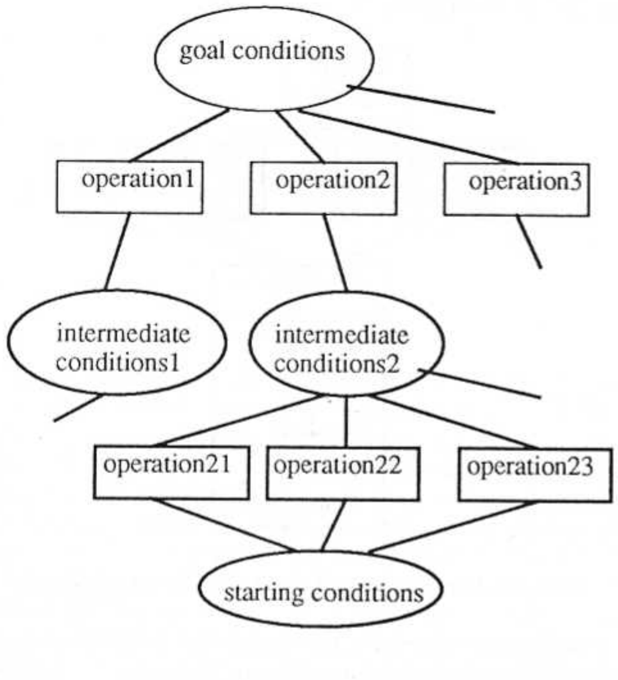

Figure 2: Planning Network

4 The Co-Evolving Species Model

4.1 Overview

The idea behind the co-evolving species model is shown in figure 3. The genotype of each specie represents a feasible 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.

The data provided by the plan space generator, and depicted in figure 2, is used to randomly generate populations of structures representing possible plans, one population for each component to be manufactured. An important part of this model is the 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 standard Holland-type genetic algorithms [10,6], using crossover and mutation and employed as the breeding stage of the core algorithm of figure 3. It is important to note that the

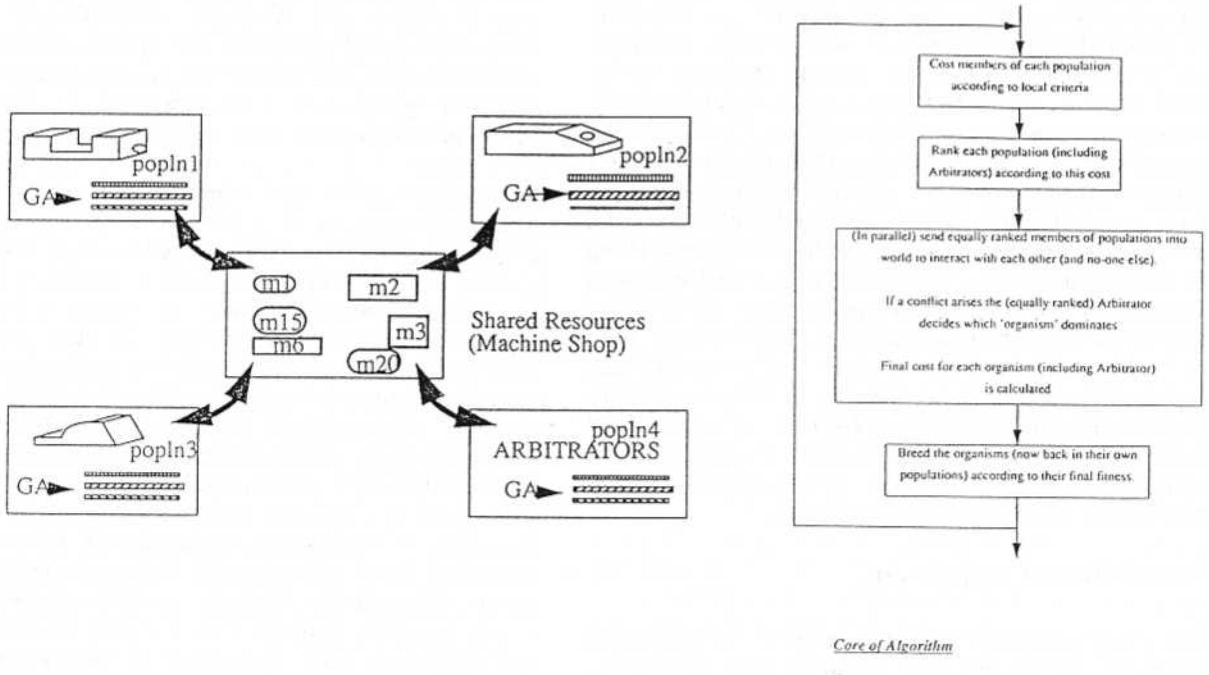
popln1
GA
popln2
GA
m1
m2
m15
m3
m6
m20
Shared Resources (Machine Shop)
popln3
GA
popln4 ARBITRATORS
GA
Cost members of each population according to local criteria
Rank each population (including Arbitrators) according to this cost
(In parallel) send equally ranked members of populations into world to interact with each other (and no-one else).
If a conflict arises the (equally ranked) Arbitrator decides which 'organism' dominates
Final cost for each organism (including Arbitrator) is calculated
Breed the organisms (now back in their own populations) according to their final fitness.
Core of Algorithm

Figure 3. Co-Evolving Species Model for Simultaneous Plan/Schedule Optimisation

environment of each population includes the influence of all the other populations.

4.2 The Genotypes

The genotype of a process plan organism can be represented as follows:

$op_1m_1s_1op_2m_2s_2Gop_3m_3s_3op_4m_4s_4op_5m_5s_5G$ ....

Where op_i refers to the ith operation in a plan, m_i to the machine to use for that operation and s_i to the setup. Operations with interdependencies are grouped together, each group being terminated by a special symbol (G in above example). As long as the group terminators are the only legal crossover points, the crossover operation will always produce legal plans. If crossover were to occur within a group, data for dependent operations would be split up and illegal plans would probably occur on recombination. The mutation operator is also fairly involved because the gene values are context sensitive due to the dependencies. This encoding encapsulates the network structures of the data produced by the plan space generator. Each op_i, m_i and s_i have associated with them finite sets of possible integer coded values. Because these sets are all quite different, bit string representations would be awkward and unnatural, hence so called real valued codes are used.

The genotype is transformed into another form for interpretation by the fitness function. This is to take into account the ordering aspect of the problem. There is a network of partial ordering constraints associated with each genotype (specie), the operations must be ordered in accordance with these. Several methods have been used to represent the ordering part of the problem: extra genes to denote the relative orderings, a separate chromosome that holds the equivalent information and the use of a separate species of parasites who performed the translation. The first two required PMX type crossover operators [7]. Further work is required on the latter, but it appears to be the most promising.

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. There is one bit for each possible population pairing at each possible stage. Hence the Arbitrator genome is a bit string of length S.N.(N-1)/2, where S = maximum number of stages in a possible plan (defined later) and N = number of process plan organism populations. Each bit is uniquely identified with a particular population pairing and is interpreted according to the following function:

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

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.

4.3 Cost Functions

As indicated in the algorithm shown in figure 3, the cost functions for all species involve two stages. The first stage involves local criteria and the second stage takes into account interactions between populations. The first stage cost function for the process plan organisms, COST1 shown below, is applied to the genotype shown above *after it has first been translated into a linearised format that can be interpreted sequentially.*

$$\text{COST1 (plan)} = \sum_{i=1}^{N} (M(m_i, i) + S(s_i, i, m_i))$$

Where s_i = setup used while processing ith operation, m_i = machine used for processing ith operation, $S(s_i, i, m_i)$ = setup cost for ith operation, $M(m_i, i)$ = machining cost for ith operation, N = number of operations to be processed and $M(m_i, i)$ has been previously calculated and is looked up in a table. Note that a setup cost is incurred every time a component is moved to a new machine or its orientation on the same machine changes. This function performs a basic simulation of the execution of the plan. Its input data is an ordered set of (machine,setup) pairs, one for each operation. The operations must be ordered in such a way that none of the constraints laid down by the planner are violated. Ordered sets of operations to be processed using a particular machine/setup combination are (effectively) built up on a 2D grid. $S(s_i, i, m_i)$ governs the way in which the sets are built up on the grid. The operations in any set can be performed in isolation from those in any other set. Such a set is referred to as a *stage* of a job throughout this paper. These sets themselves are ordered and the outcome is a process plan like the one shown below, where the integers in the sets refer to particular operations.

1) machine: 6 setup: 5 [0,3,5,7]
2) machine: 2 setup: 21 [1,8,12,19]
3) machine: 11 setup: 4 [2,4,6,9,13,15] ...etc

In fact COST1 provides a mapping from the process plan genotype to its phenotype: one of the plans illustrated above. Note that the setup cost is often considerably more (orders of magnitude) than the basic machining costs. The essential workings of COST1 is to sequentially process the transformed genome in order to group operations together in clusters which can then be scheduled as single units (stages). At the same time the final executable ordering of the operations is found, as well as the basic machining costs.

The definition of $S(s_i, i, m_i)$ is given below:

environment of each population includes the influence of all the other populations.

4.2 The Genotypes

The genotype of a process plan organism can be represented as follows:

$op_1m_1s_1op_2m_2s_2Gop_3m_3s_3op_4m_4s_4op_5m_5s_5G$ ....

Where op_i refers to the ith operation in a plan, m_i to the machine to use for that operation and s_i to the setup. Operations with interdependencies are grouped together, each group being terminated by a special symbol (G in above example). As long as the group terminators are the only legal crossover points, the crossover operation will always produce legal plans. If crossover were to occur within a group, data for dependent operations would be split up and illegal plans would probably occur on recombination. The mutation operator is also fairly involved because the gene values are context sensitive due to the dependencies. This encoding encapsulates the network structures of the data produced by the plan space generator. Each op_i, m_i and s_i have associated with them finite sets of possible integer coded values. Because these sets are all quite different, bit string representations would be awkward and unnatural, hence so called real valued codes are used.

The genotype is transformed into another form for interpretation by the fitness function. This is to take into account the ordering aspect of the problem. There is a network of partial ordering constraints associated with each genotype (specie), the operations must be ordered in accordance with these. Several methods have been used to represent the ordering part of the problem: extra genes to denote the relative orderings, a separate chromosome that holds the equivalent information and the use of a separate species of parasites who performed the translation. The first two required PMX type crossover operators [7]. Further work is required on the latter, but it appears to be the most promising.

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. There is one bit for each possible population pairing at each possible stage. Hence the Arbitrator genome is a bit string of length S.N.(N-1)/2, where S = maximum number of stages in a possible plan (defined later) and N = number of process plan organism populations. Each bit is uniquely identified with a particular population pairing and is interpreted according to the following function:

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

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.

4.3 Cost Functions

As indicated in the algorithm shown in figure 3, the cost functions for all species involve two stages. The first stage involves local criteria and the second stage takes into account interactions between populations. The first stage cost function for the process plan organisms, COST1 shown below, is applied to the genotype shown above *after it has first been translated into a linearised format that can be interpreted sequentially.*

$$\text{COST1 (plan)} = \sum_{i=1}^{N} (M(m_i, i) + S(s_i, i, m_i))$$

Where s_i = setup used while processing ith operation, m_i = machine used for processing ith operation, $S(s_i, i, m_i)$ = setup cost for ith operation, $M(m_i, i)$ = machining cost for ith operation, N = number of operations to be processed and $M(m_i, i)$ has been previously calculated and is looked up in a table. Note that a setup cost is incurred every time a component is moved to a new machine or its orientation on the same machine changes. This function performs a basic simulation of the execution of the plan. Its input data is an ordered set of (machine,setup) pairs, one for each operation. The operations must be ordered in such a way that none of the constraints laid down by the planner are violated. Ordered sets of operations to be processed using a particular machine/setup combination are (effectively) built up on a 2D grid. $S(s_i, i, m_i)$ governs the way in which the sets are built up on the grid. The operations in any set can be performed in isolation from those in any other set. Such a set is referred to as a *stage* of a job throughout this paper. These sets themselves are ordered and the outcome is a process plan like the one shown below, where the integers in the sets refer to particular operations.

1) machine: 6 setup: 5 [0,3,5,7]
2) machine: 2 setup: 21 [1,8,12,19]
3) machine: 11 setup: 4 [2,4,6,9,13,15] ...etc

In fact COST1 provides a mapping from the process plan genotype to its phenotype: one of the plans illustrated above. Note that the setup cost is often considerably more (orders of magnitude) than the basic machining costs. The essential workings of COST1 is to sequentially process the transformed genome in order to group operations together in clusters which can then be scheduled as single units (stages). At the same time the final executable ordering of the operations is found, as well as the basic machining costs.

The definition of $S(s_i, i, m_i)$ is given below:

$$S(s_i, i, m_i) = \begin{cases} f(s_i, m_i) \text{ ,if s,m combo not previously encountered} \\ f(s_i, m_i) \text{ ,if i causes break-constraint in all grid sets} \\ f(s_i, m_i) \text{ ,if i causes set-incompatibility on all grid sets} \\ 0, \quad \text{otherwise} \end{cases}$$

$f(s_i,m_i)$ is a simple table look-up function, the pre-calculated cost of performing operation i with the particular machine and setup.

Full details of the simulation functions would take up too much space, but the following gives a brief explanation of break-constraint and set-incompatibility, as mentioned in the function definition above. Suppose operation x is being processed by COST1(plan) . It has associated with it <machine/setup> combination (m_x,s_x). Also suppose this combination has already been encountered by the objective function and so a corresponding set, S_0, exists on the simulation grid. A break-constraint occurs, in an attempt to add x to S_0, under the following condition:

$\exists z \exists y \, (y \in S_0 \wedge z \in S_n \wedge S_n \neq S_0 \wedge y \rightarrow z \wedge z \rightarrow x)$
Where, -> represents a partial ordering constraint, y and z are operations and S_n is any set on the grid.

That is, a break-constraint occurs for operation x with the <m_x , s_x > combination when there exists some operation y which has already used this same combination, i.e $s_y = s_x$ and $m_y = m_x$, and has the following additional property: due to the ordering constraints on the problem there exists a third operation, z , which must be processed after y but before x and does not use the same < setup , machine > combination, i.e. $(m_z , s_z) \neq (m_x, s_x)$. When a break-constraint occurs it is not possible to process all those operations linked to a particular <machine, setup > combination without changing machine and/or setup part way through to process some other operation. Obviously the setup cost is incurred again when processing moves back to the original machine. As far as the mechanism of the simulation is concerned, if a break-constraint occurs in an attempt to add operation x to grid set S_0, a new set, with x as the first member, is started at the same grid location as S_0.

The set-incompatibility condition is slightly more subtle and is defined as follows: feature x causes a set-incompatibility if, when we are trying to add it to some set, S_0, of operations on the grid,

$\exists y \, (y \in S_n \wedge S_n \neq S_0 \wedge y \rightarrow x)$
$\& \; \exists z \exists w \, (z \in S_n \wedge w \in S_0 \wedge w \rightarrow z)$
Where, -> represents a partial ordering constraint, y, w and z are operations and S_n is any set on the grid.

The sets S_0 and S_n are incompatible as it is not possible to order them in relation to each other. We must start a new set at the same position as S_0 on the grid and with x as the first member. Every time a new set is created on the grid its number (order in which sets are created) and grid position are added to the end of a 'sets_so_far' list. The set ordering algorithm is based on the well known bubble-sort method; it uses this list as the array to sort. The crucial test in any sorting algorithm is one for deciding whether two adjacent members of the array are in the correct order. The action of the setup function $S(s_i , i , m_i)$, particularly the set-incompatibility condition, ensures that it is possible to order any two grid sets. The ordering condition is simple. If any member of set S_i has any member of set S_j in its extended after-constraints list, then S_i is ordered before S_j. Formally, S_i -> S_j if:

$(\exists x \exists y \, (x \in S_i \wedge y \in S_j \wedge y \in A_x^{ext}))$ (i.e. x->y)

Where, S_i and S_j are grid sets, x and y are operations, -> represents a partial ordering constraint and A^{ext}_x is the complete set of all operations lying after x in the overall partial ordering.

It should be clear from the above that COST1 involves a fairly complex *interpretation* of the genotype, there is quite a high level of epistasis and the mechanisms for genes to influence each other's contributions to the overall genome fitness are complex. Despite this, genetic search performs very well.

The local cost criteria for the Arbitrators is derived from the final fitness of their parents. The function is given below, the section of the offspring genome up to *cp* (crossover point) was copied from parent1 and the section after *cp* was copied from parent2.

If cp > ALEN, cost = cost of parent1 (active part of genome was inherited solely from parent1)
Else, cost = (cp/ALEN)(cost of parent1) + (1 - cp/ALEN)(cost of parent2)

W here ALEN is the average useful Arbitrator length. This is a dynamic quantity as fitter plans tend to become shorter meaning that arbitrating decisions are not needed for later stages as the system evolves. This is only used because a fully dynamic implementation of the Arbitrators has not yet been completed.

The second phase of the cost function involves simulating the simultaneous execution of plans derived from stage one. Additional cost are incurred for waiting and going over due dates. There are a number of interesting problems here. We are working towards a set of optimal plans, one for each component, which when executed simultaneously will provide an optimal schedule. This means that most of the possible interactions between members of one population and all the members of another population are largely irrelevant. The solution used here was to rank each population according to the local cost functions described above and to run the simulation of phase two for equally ranked organisms. What happens when two plans want the same resource at the same time? Fixed precedences would be far too inflexible and random choices would be of no help. As already indicated, the most general and powerful solution developed was to introduce a new species, the Arbitrators, whose genetic code holds a table indicating which population had precedence at any stage. The Arbitrators are costed according to the amount of waiting and the total elapsed time for a given simulation. The smaller these two values, the fitter the Arbitrator. Hence the Arbitrators, initially randomly generated, are allowed to co-evolve with the plan organisms. Again, the Arbitrators are ranked and a simulation involves equal ranking members from each population, including the Arbitrators. If there is a conflict the Arbitrator resolves it. This scheme allows the evolution of sensible priorities at the various stages of the simulation. After the second phase each individual's fitness is calculated according to its *total* cost. This means that selection pressure takes account of both optimisation problems: interactions during phase two that increase an individuals cost will reduce its chances of reproduction, just as will a poor result from phase one of the costing. In general, a population of co-evolving Arbitrators could be used to resolve conflicts due to a number of different types of operational constraint. The cost of an Arbitrator after the second phase simulation is a function of the total schedule length and the weighted penalties incurred by the various plan populations.

5 Results

Figure 4 shows results from an implementation on a transputer based parallel machine. Typical results for a two job problem are shown. The graph shows how the machining costs (COST1) of the best individual in each population reduce with time, and also how the Arbitrator costs reduce. It also shows how the total elapsed time reduces. The gantt charts show how the emergent schedule evolves. The vastly reduced number of stages in the lower chart reflects the fact that machining costs can be decreased by putting more operations into a single stage. Clearly both optimisation problems have been tackled simultaneously. Note that there is some tension between the various objectives, one cost may momentarily rise while others drop, but the overall trend is down. A model of a real job-shop is used and the components planned for are of medium to high complexity needing 25-60 operations to manufacture. Each job has a number of internal partial ordering constraints but is by no means strongly constrained. Typically each operation has 8 candidate machines and each of these machines has 6 possible setups To simplify matters, tool changes and machine transfer costs have not been modelled in great detail. However, it is a simple matter to include them and future versions of the model will be complete in that respect. Experiments with up to 4 jobs have been conducted. Very promising results have been obtained for this extremely complex optimisation problem, never before attempted.

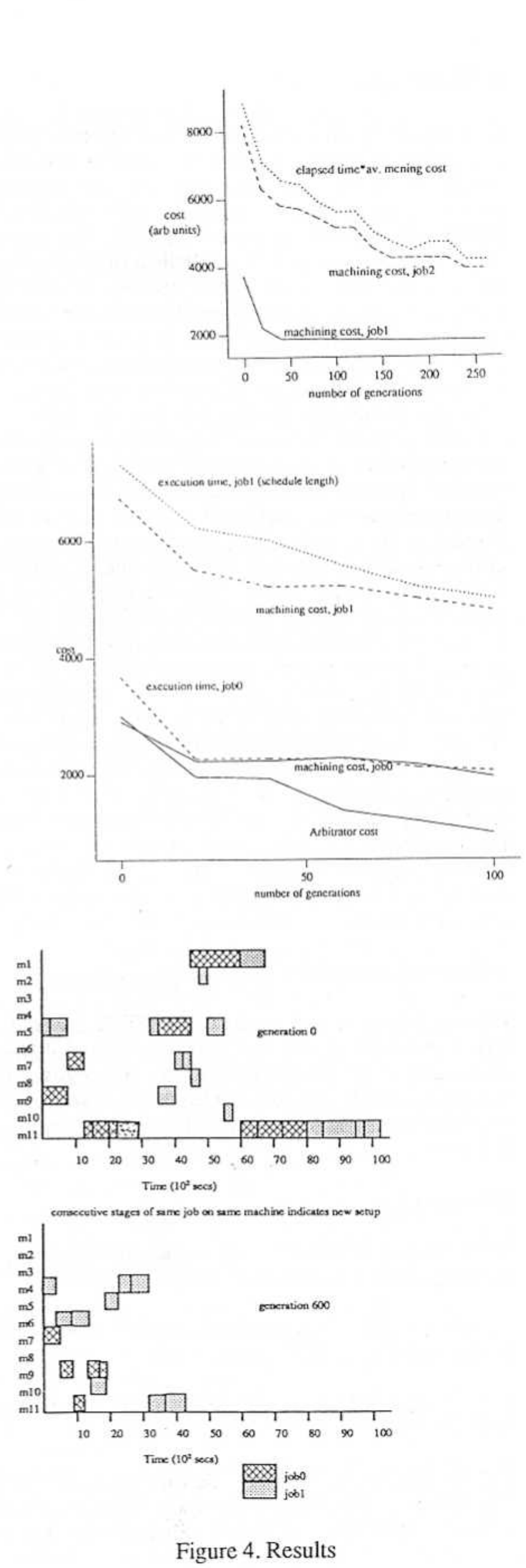

Figure 4. Results

6 Discussion

Davis has done some work on using GAs to solve job-shop scheduling problems [4], but his solution was for the simplified problem that does not take into account the proper relationship between planning and scheduling. Each genome represented an entire schedule, that approach cannot exploit the inherent parallelism of the problem in the same way that the work described here has. Hilliard et al [8] have used a classifier system to discover scheduling heuristics. That work may possibly tie in with ongoing research on enabling the Arbitrators to learn how to resolve a number of different type of conflicts, there is no reason why the Arbitrators should not become fully blown classifier systems. Because this system runs on a powerful parallel machine (500 transputers) very good solutions are found within a few minutes, because of this not much effort has yet been put into making the system react to sudden changes in the manufacturing environment. However, this is an area for future research. One possible scenario that is envisaged is that the system will run in the background and be continuously updated with feedback from the job-shop, in other word the simulated environment will dynamically mirror the actual manufacturing environment. Hillis and Koza [9,13] have previously used co-evolution but in quite different contexts. The work has used a very straightforward implementation of the actual GAs on the transputers, it may benefit from some more work in that direction. Certainly work is ongoing in extending the model and running it with 50-60 jobs rather than the handful used to date. There is a great deal of work in setting up the planning data for further jobs. However, an interactive system that should make that much easier is near completion.

Acknowledgements

This work was in part supported by SERC grants GR/D 63103 and GR/E 04837. It has benefited from discussion with many people including Stephen Warrington, Inman Harvey and Phil Agre. Acknowledgements are also due to The Edinburgh Manufacturing Planning Group and The Edinburgh Parallel Computing Centre.

References

[1] Chang, T. & Wysk,R. "An Introduction to Automated Process Planning Systems", Prentice-Hall, 1985.

[2] Chapman, D. "Planning for Conjunctive Goals", Tech. report AI-TR-802, MIT AI Lab, 1985.

[3] Christophedes, N. "Combinatorial Optimisation", Wiley, 1979.

[4] Davis, L. 'Job Shop Scheduling with Genetic Algorithms', in J. Grefenstette (ed), *Proc. Int. Conf. on Genetic Algorithms and their Applications*, Lawrence Erlbaum,1985.

[5] Garey, M. & Johnson, D. "Computers and Intractibility: A Guide to the Theory of NP-Completeness", W.H. Freeman, 1979.

[6] Goldberg, D. "Genetic Algorithms", Addison Wesley, 1989.

[7] Goldberg, D. & Lingle, R.
"Alleles, Loci and The TRavelling Salesman Problem", in J. Grefenstette (ed), *Proc. Int. Conf. on GAs and their Applications, Lawrence Erlbaum*, 1985.

[8] Hilliard, M et al. 'A Classifier based system for discovering scheduling heuristics', in J. Grefenstette (ed), *Proc. 2nd Int. Conf. on GAs*, Lawrence Erlbaum,1987.

[9] Hillis, W.D. 'Co-Evolving Parasites Improve Simulated Evolution As an Optimisation Procedure', Physica D 42,228-234, 1990.

[10] Holland, J. "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1976.

[11] Husbands,P., Mill,F.G. & Warrington,S.W., 'Representation, Reasoning and Decision Making in Process Planning with Complex Components',in "*Geometric Reasoning*", Woodwark, J (ed),203-215, Oxford University Press, 1989.

[12] Husbands,P., Mill,F.G. & Warrington,S.W., 'Generating Optimal Process Plans from First Principles', in *"Expert Systems for Management and Engineering"*, Balagurasamy, E. & Howe, J. (eds),130-153, Ellis Horwood, 1990.

[13] Koza, J. 'Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems', Tech. Report STAN-CS-90-1314, Dept. Compt. Sci., Stanford University, 1990.

[14] Pettey, C., Leuze, M., Grefenstette, J.
"A Parallel Genetic Algorithm", in J. Grefenstette (ed), *Proc. 2nd INt. Conf. on GAs*, Lawrence Erlbaum, 1987.

[15] Muhlenbein, H.
"Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimisation", in J. Schaffer (ed), *Proc. 3rd INt. Conf. on GAs*, Morgan Kaufmann, 1989.