INT. J. COMPUTER INTEGRATED MANUFACTURING, 1993, VOL. 6, NOS. 1 & 2, 74–86

An ecosystems model for integrated production planning

PHILIP HUSBANDS

Abstract. This paper re-evaluates the job-shop scheduling problem by showing how the standard definition is far more restrictive than necessary and by presenting a new technique capable of tackling a highly generalized version of the problem. This technique is based on a massively parallel distributed genetic algorithm and is capable of simultaneously optimizing the process plans of a number of different components, at the same time a near-optimal schedule emerges. Underlying the evolutionary machinery is a specialized feature-based generative process planner.

1. Introduction

Research on job-shop scheduling (JSS), as the most general of the classical scheduling problems, has generated a great deal of literature (Muth and Thomson 1963, Balas 1969, Garey *et al.* 1976, Graves 1981, Ow and Smith 1988, Carlier and Pinson 1989). All of this work has used a particular definition of the scheduling problem or very close variants of it. This paper will argue that the standard definition is far more restrictive than is necessary. In particular, it is claimed that the relationship between process planning and scheduling has been largely ignored. A new technique, capable of tackling a highly generalized JSS, is presented. The algorithm used is highly parallel and makes use of methods analogous to those occurring in a natural evolving ecosystem. Underlying the evolutionary machinery is a specialized feature-based generative process planner. It is shown how the technique provides a highly integrated production planning system, treating process planning and scheduling as inextricably interwoven parts of the same problem.

The very large body of work on solving planning and scheduling problems has emanated mainly 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 (Chapman 1985). The general resource planning, or scheduling, problem is well known to be NP-complete (Garey and Johnson 1979). Consequently OR techniques have been developed to give exact solutions to restricted versions of the problem, but in general, as with AI-based approaches to the problem, there is a reliance on heuristic-based methods. Because of the complexity and size of the search spaces involved, a number of simplifying assumptions have always been used in practical applications. These assumptions are now implicit in what have become the standard problem formulations. In many instances this has led to the most general underlying optimization problem being ignored or, more often, not even acknowledged as existing at all.

The most sweeping of these simplifications involves the relationship between process planning and scheduling. Scheduling is traditionally seen as the task of finding an optimal way of interleaving a number of fixed 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 often many possible choices for the sub-operations in the plans. Very often the real optimization problem is to optimize simultaneously all 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 optimization 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. The potential economic benefits are obvious.

The following section makes clear the domain definitions used in the work described. The core techniques used in this research are a specialized form of feature-based process planning and a distributed genetic algorithm. Section 3 gives a brief introduction to genetic

Author: P. Husbands, School of Cognitive and Computing Sciences, University of Sussex, Falmer, Brighton BN1 9QH, UK.

0951-192X/93 $3.00

algorithms and then Section 4 provides an overview of the whole ecosystems model. Section 5 details the feature-based process planning elements of the model, while Sections 6 and 7 describe the parallel genetic algorithm parts. Section 8 presents results from a massively parallel implementation and Section 9 concludes the paper with a discussion of the implications of the work.

2. Domain of application definitions

2.1. *Process planning*

The technique presented here is generally applicable but will be described in terms of the manufacture of medium-complexity prismatic parts requiring the application of a number of metal-removal processes. Within this framework a standard definition of process planning is used (Chang and Wysk 1985), namely establishing the operations required to manufacture a part, the appropriate machine tool and machining parameters to use for each operation and the order in which the operations should be performed. It will be seen that each operation in the plan corresponds to processing a manufacturing feature or group of features on the work piece. Hence, in this work, a process plan is essentially regarded as an ordered set of [feature, machine, process, tool, setup] tuples. Section 5 gives details of the various feature types used and how feature interactions are dealt with.

2.2. *The classical definition of JSS*

The standard JSS problem definition is taken to be the following. 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 JSS problem is to find the sequence of jobs on each machine in order to minimize 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 (Christophedes 1979).

2.3. *An integrated view of process planning and JSS*

Very often complete fixed process plans are presented as the raw data for the scheduler. However, in many manufacturing environments there is 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 given the machine and tool choices. 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 bottlenecks. Because of the complexity of the overall optimization problem, that is simultaneously optimizing 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. A number of researchers have developed scheduling techniques that allow a small number of options in their process plans (Sycara *et al.* 1991, Tonshoff *et al.* 1989), but still they are dealing with only a tiny fraction of the whole problem. Liang and Dutta (1990) have also pointed out the need to combine planning and scheduling, but their proposed solution was demonstrated on a very small simplified problem. It is not at all clear if it will scale up to be able to deal with the kinds of test problems described later. The technique presented in this paper, developed by viewing the problem in a completely new way, appears to be the only piece of work fully addressing this highly generalized version of the JSS problem.

3. An introduction to genetic algorithms

Genetic algorithms (GAs) are a key technique used in this work. Since knowledge of the method has not yet spread to all scientific and technical quarters, a brief introduction is given here. For further details see Goldberg (1989), Davis (1990), and Husbands (1992).

We are the existing proof of the astonishing power of natural evolution, a process of selection acting on small variations within a species. It is tempting to imagine that highly effective techniques for optimization, and for the design of adaptive systems, can be abstracted from the logic of natural evolution. Over the past 40 or so years a number of researchers have tried to do just that. The most powerful and successful methods emerged in the late 1960s and early 1970s and are based on Holland's genetic algorithm (Holland 1975).

Genetic algorithms are adaptive search strategies based on a highly abstract model of biological evolution. They can be used as an optimization tool or as the basis of more general adaptive systems. The fundamental idea is as follows. A population of structures, representing candidate solutions to the problem at hand, is produced. Each member of the population is evaluated according to

some fitness function. Fitness is equated with goodness of solution. Members of the population are selectively interbred in pairs to produce new candidate solutions. The fitter a member of the population is the more likely it is to produce offspring. Genetic operators are used to facilitate the breeding; i.e. operators that result in offspring inheriting properties from both parents (sexual reproduction). The offspring are evaluated and placed in the population, quite possibly replacing weaker members of the last generation. The process repeats to form the next generation. This form of selective breeding quickly results in those properties that promote greater fitness being transmitted throughout the population: better and better solutions appear. Normally some form of random mutation is also used to allow further variation. A simple form of this algorithm is illustrated in Figure 1. This population-based survival-of-the-fittest scheme has been shown to act as a powerful problem-solving method over a wide range of complex domains (Grefenstette 1985, 1987, Schaffer 1989, Belew and Booker 1991, Schwefel and Manner 1991, Davis 1990).

The population of structures to undergo adaptation generally consists of strings (chromosomes) of a fixed length. Each element (gene) of the string represents some aspect of the solution and will have a set of possible values (alleles) mapped to various attributes. The fitness of such a string is measured by some objective function that costs the particular combination of attributes present. Hence the chromosomes may be, for instance, strings of real numbers, strings of integers, bit strings (string of 1s and 0s to be decoded into a set of parameter values), a permutation of some set of elements, a list of rules or some combination of these representations.

The set of genetic operators developed by Holland, and the one generally used (possibly with domain-specific modifications), consists of three operators: crossover, inversion and mutation. Simple crossover involves choosing at random a crossover point (some position along the string) for two mating chromosomes, then two new strings are created by swapping over the sections lying after the crossover point. Multi-point crossovers are also frequently used. Inversion is simply a matter of reversing a randomly chosen section of a single string. Mutation changes the value of a gene to some other possible value. The genetic operators are applied at the breeding stage according to a routine like the following. When two strings are selected for breeding, first apply crossover (with some high probability) and randomly choose one of the two new strings thus formed. Next apply inversion (with a medium probability) to this string. Each gene on the resulting string undergoes mutation (with a very low probability) and the outcome is taken as the offspring. The basic operators and the breeding process are illustrated in Figure 2. Note the

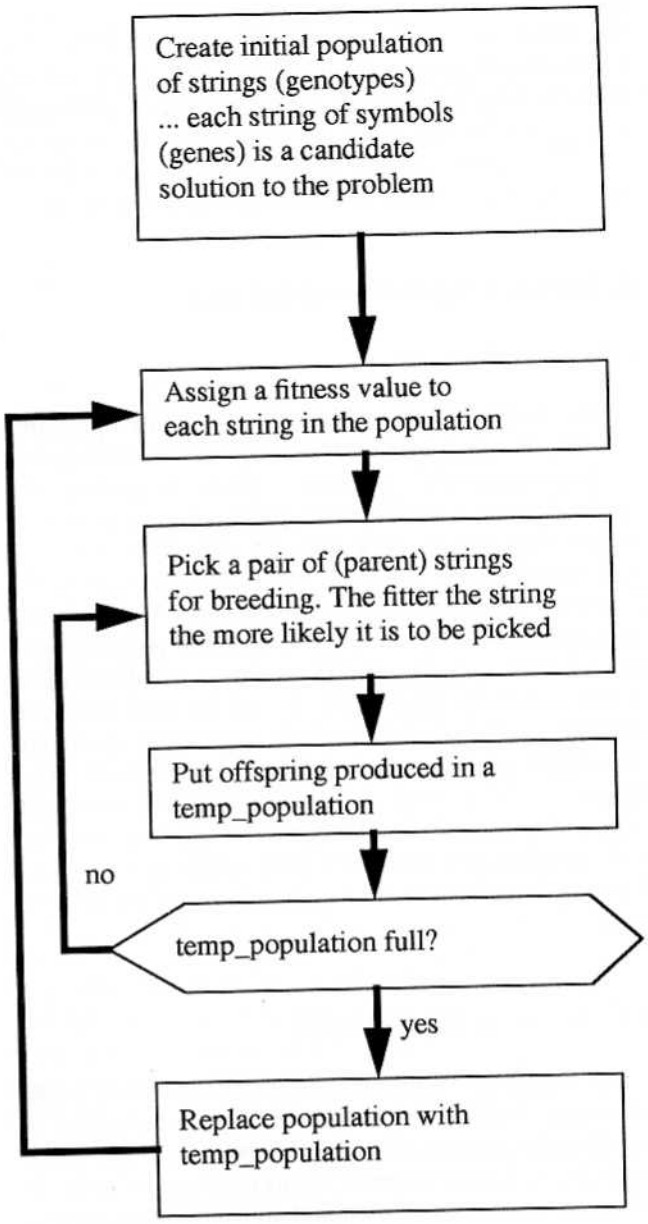


Figure 1. A simple genetic algorithm.

stochastic nature of this process. All operators are applied probabilistically and crossover and inversion points are chosen randomly.

The overall effect is to emphasize combinations of basic building blocks (groups of genes) that produce maximum fitness.

In some problem domains it may be beneficial to allow dynamic length strings. This can be achieved by randomly selecting different crossover points on each parent rather than forcing them to be the same, although recent arguments (Harvey 1992) strongly suggest that changes in length should be restricted to be small and gradual.

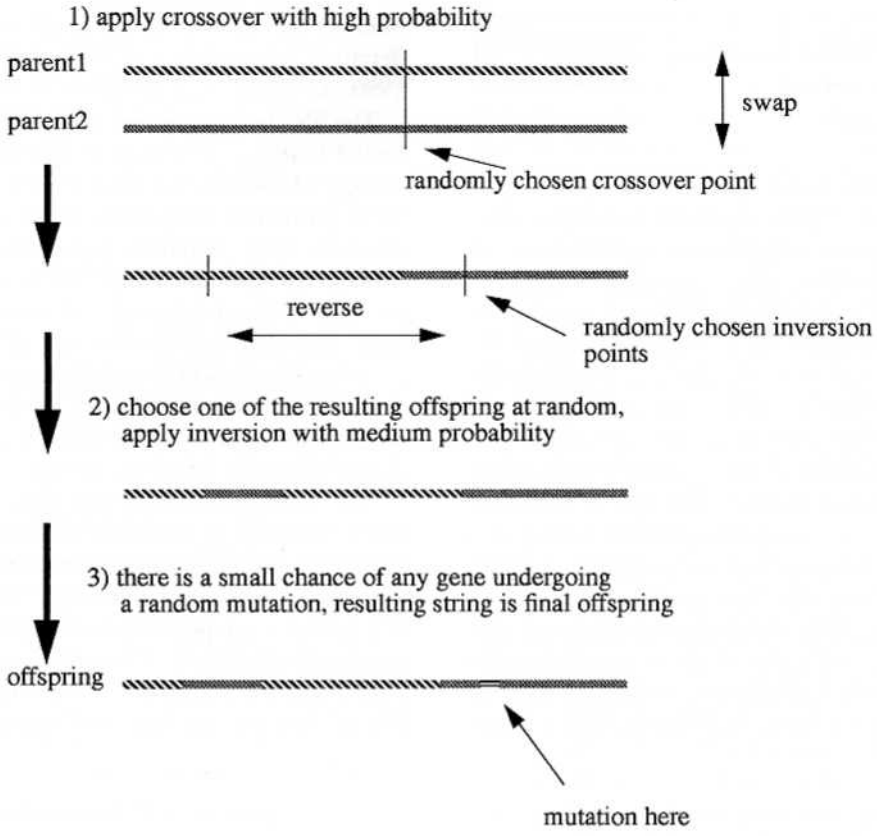


Figure 2. Application of the genetic operators.

Other operators, such as translocation (moving a section of the string to a new location), may also be useful.

There are many variations on and extensions to the basic algorithm. In particular, highly parallel implementations of GAs, with 'geographically' distributed populations and local selection only, appear to be the most powerful (Husbands 1992).

Because Holland and his students developed GAs to serve as adaptive problem-solving strategies able to operate over a large range of environments, their GAs have qualities that make them suitable for many large combinatorial problems and string-representable search tasks. By a combination of selection and reproduction via genetic operators, they are able to find very fit structures by searching only a tiny proportion of the whole problem space. As long as the string representations and the cost function are accurate, GAs can conduct a successful search without recourse to any special domain-specific heuristics. The subtlety of their action prevents them from getting stuck on local optima and ensures that they simultaneously search widely separated parts of the problem space. This is largely due to the random elements in the action of the genetic operators. No assumptions need to be made about the search space, often in contrast to the situation with branch and bound and various heuristic search techniques. Because GAs manipulate populations of legal solutions, they do not suffer from exponential memory usage like many versions of branch and bound and dynamic programming, which attempt to build up gradually a single optimal solution. These qualities make GAs an extremely robust problem-solving method. It is this robustness that makes them an attractive and useful search technique.

Although the basic algorithm is computationally trivial, it should be noted that a great deal of ingenuity is often needed to derive a suitable encoding for a problem and to provide it with an appropriate set of genetic operators and a sufficiently discriminating fitness function. This point will be illustrated later in this paper when the somewhat more complex GA used in this work is described.

4. Overview of ecosystems model

This paper concentrates on two core aspects of a complete framework for dealing with a certain class of design and manufacturing problems. The overall approach is now briefly presented. This is captured, at a very high level, in Figure 3. A design system, whose description is

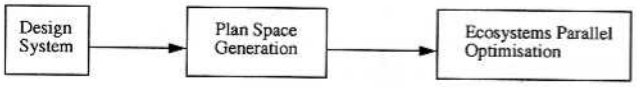


Figure 3. Overall approach.

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. (This refers to the fact that all the data needed to construct explicitly the search space point by point is made available. This amount of data is of course quite manageable, whereas the explicitly generated search space would certainly not be. Enumerative search on this kind of problem is quite out of the question.) 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 optimization techniques work in. The last two modules of this system are dealt with here (for further details of other aspects of the system see Husbands *et al.* 1990, and Mill *et al.* 1992).

The idea behind the ecosystems, or coevolving species, model is shown in Figure 4. The genotype (genetic encoding) of each species 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 optimized.

The data provided by the plan space generator are used randomly to construct initial 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

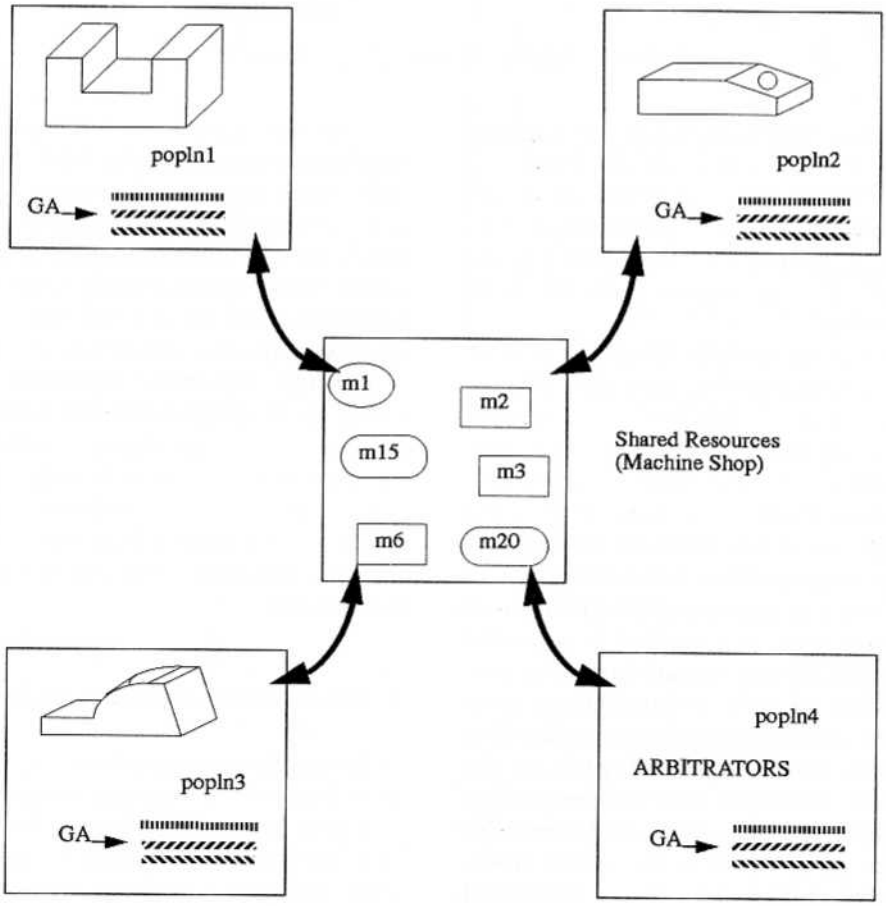


Figure 4. The ecosystems model.

population, including the arbitrators, evolves under the influence of selection, crossover and mutation. It is important to note that the environment of each population includes the influence of all the other populations. The following three sections will fill in the details of this sketch, starting with the plan space generator.

5. The plan space generator

The plan space generation algorithm attempts to break the manufacture of a component down into a number of nearly independent stages. The entire space of possible plans can then be generated by finding all the possible operations to carry out each stage along with the ordering constraints which must exist between the stages. Essentially a stage refers to a finishing operation on a single feature or super-feature (a group of features treated as one due to some network of constraints binding them together) or a roughing operation on an intermediate feature (defined later). So each stage of the plan has a unique feature, super-feature or intermediate feature associated with it. The operations found to manufacture these are described in terms of [machine/process/tool/setup/cost] combinations. The setup refers to the orientation of the workpiece and the cost refers to the machining cost associated with that operation. Along with this information the planner generates a separate network representing the partial orderings it has deduced hold between the stages of the plan.

The simplest way to describe the algorithm in more detail is to start with the highest level structures it build and manipulates. These are planning networks like th one shown in Figure 5. In common with most generativ process planners, the manufacturing processes ar treated as material addition operations, whereas c course they actually involve material removal. Th overall strategy is to start with those features deepest i the component and work out towards the surface. Th process is guided by a set of 'critics' constantly on th look out for possible feature interactions, which ma result in deferring work on part of the componer (Husbands *et al.* 1990), and by high-level consideratio regarding datums and such like. Once a feature has bee chosen, a finishing process to achieve its desired fin state is inferred. The details of this are discussed late This finishing process leaves an intermediate feature wi various inexact properties, such as a range of possib surface finishes. This models the fact that most finishi processes can only sensibly be started from a state wi a given range of properties. For instance, it is high undesirable to end up grinding down a very rou uneven surface. A roughing process is then chosen manufacture the intermediate feature. Rememberi that an exhaustive set of possible manufacturing routes required. for any given finishing process. any number compatible roughing processes may exist. Thus network like the one shown in Figure 5 is built u keeping track of the interactions between finishing a roughing processes for each feature. These networks c be readily extended to allow an arbitrary number of su finishing and sub-roughing processes, and hence int

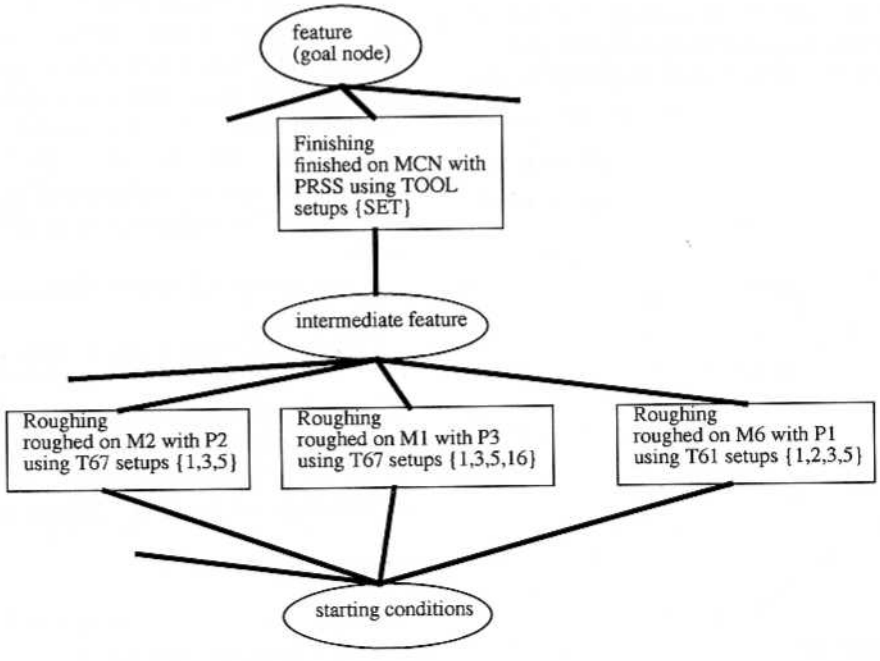


Figure 5. Fragment of planning network.

mediate features, to be handled. Each route on the network, from starting conditions to final feature, has its own subsidiary information attached, such as machining parameters and cost. In complex cases it may be desirable to weight the different routes or to remove certain nodes or connections. It is often found, when building up the network, that a possible roughing operation is exactly the same as the finishing operation it is connected to via an intermediate feature. In this case the roughing node and its connections are removed and a connection made directly from the starting conditions to the finishing node. This tells us that it is feasible to machine out the feature by using the single process. Various kinds of links between the sub-networks of different features are built up by the planner.

A full description of the plan space generation algorithm is outside the scope of this paper but see Husbands *et al.* (1990). The latest implementation is in C++ (hence object-oriented) and makes use of the ACIS solids modeller. The component model is continually updated as features, intermediate features and super-features are dealt with. The core mechanism for process, machine, tool and setup selection is as follows. Feature objects have a list of possible manufacturing processes associated with them; process objects are interrogated by the feature objects to see if they are suitable. Process objects have a list of machine types usually capable of performing them; process objects interrogate the machine class to find actual machines that can be used. In a similar way the tools and possible setups for a given [feature/process/machine] combination are found. The solid model is used to check for access and possible feature interactions.

The output from this process is a large number of interconnected networks like the one shown in Figure 5. A manufacturing plan 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 between sub-operations. The algorithm also discovers ordering constraints in the processing of the various features, intermediate features and super-features. This results in the output of a partial ordering graph like the one shown in Figure 6. Each of the symbols refers to a particular feature, intermediate feature or super-feature.

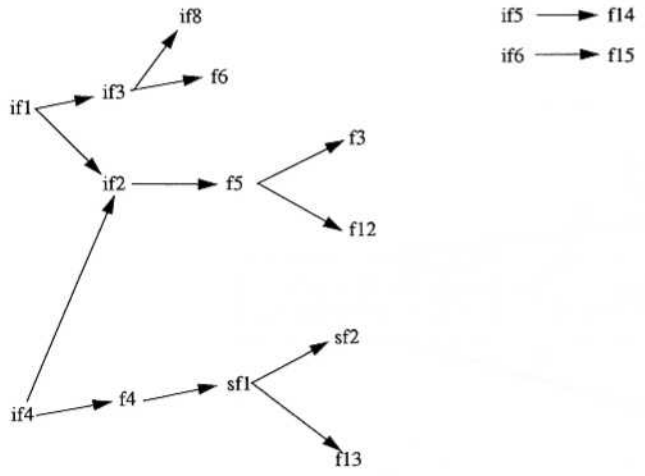


Figure 6. Anteriority constraints.

6. GAs for process plan optimization

The next step in understanding the details of the full ecosystems model is to look at the GA-based optimization of a single process plan. For the rest of this paper only machining and setup costs will be considered. This is to simplify the formulae. There is no loss in generality, it is straightforward to see how tool change and machine transfer costs could be included, but for the moment a machine selection implies a given process and tool too.

6.1. *An obvious approach*

An obvious string encoding of a process plan is:

$$f_1 m_1 s_1 f_2 m_2 s_2 \ldots f_i m_i s_i \ldots F_N m_N s_N$$

This is a simple order-based encoding to be read from left to right: f_i refers to the ith feature to be processed and m_i and s_i refer to the machine and setup to use for that operation. Each f_i could take on the value of any of the actual features (intermediate features, super-features) in the component as long as no ordering constraints are broken. The initial population, generated at random, would include many different orderings of the actual features and many different machine and setup combinations for the features. The genotypes (genetic encodings) could be implemented as integer strings if features. machines and setups were all given unique integer codes.

A suitable cost function for this encoding is

$$COST_0 = \sum_{i=1}^{i=N} (M_0(f_i, m_i) + S_0(m_i, s_i, m_{i-1}, s_{i-1})) \quad (1)$$

Where $M_0(f_i, m_i)$ is the basic machining cost for processing f_i on m_i. This value could be previously calculated according to standard formulae and stored in a table. S_0 is the setup cost function, defined as:

$$S_0(m_i, s_i, m_{i-1}, s_{i-1}) = \begin{cases} 0 & \text{if } m_i = m_{i-1} \text{ and } s_i = s_{i-1} \\ setup(m_i, s_i) & \text{otherwise.} \end{cases} \quad (2)$$

Where $setup(m_i, s_i)$ is a standard function. In English, a setup cost is always incurred unless the last feature to be processed used the same machine and setup.

This encoding and cost function provide the basis for the operation of a genetic algorithm like the one described in Section 3. However, because the strings in the population will have the features in different orders, simple crossover will nearly always produce illegal offspring with some features missing and some represented twice. Added to this is the problem of interdependent finishing and roughing operations being split up. Hence a modified crossover operator, which repairs the offspring to make it legal, would have to be used. This would be something like the PMX operator described by Goldberg and Lingle (1985), but would be significantly more computationally expensive because the partial ordering and planning networks would have to be continually checked to avoid contravening anteriority constraints and operation dependencies. Hence a rather more sophisticated encoding and cost function have been developed. (See Vancza and Marcus (1991) for another interesting alternative approach to this problem.)

6.2. *A more subtle approach*

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

$$f_1 m_1 s_1 f_2 m_2 s_2 G f_3 m_3 s_3 f_4 m_4 s_4 f_5 m_5 s_5 G \ldots$$

Here f_i no longer refers to the ith feature to be processed in a plan, but to the ith feature in a fixed ordering scheme that groups together interdependent operations (e.g. roughing and finishing operations from the same planning network). Again m_i refers to the machine used to process that feature and s_i to the setup. Each group of interdependent operations is terminated by a special symbol (G in the example above). As long as the group terminators are the only legal crossover points, the simple crossover operator will always produce legal plans; each member of the population following the same encoding and hence the same feature ordering. If crossover were to occur within a group, data for dependent operations would be split up and illegal plans would probably occur on recombination (e.g. including incompatible roughing and finishing operations). 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 f_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. It is probably not obvious how this new encoding is to be interpreted as a plan. This will become clear in a little while.

Although this encoding allows the unmodified use of the computationally trivial simple crossover operator, it appears to ignore the ordering aspect of the search problem. In fact it does not, this has now been transferred into a rather more complex costing function. This function, $COST_1$, shown below, is applied to the genotype shown above after it has been translated into a linearized format that can be interpreted sequentially. This is achieved by regrouping the features, according to a fixed scheme, taking into account the anteriority constraints and resulting in an encoding equivalent to that described in Section 6.1. But note that here it is only used as an intermediate encoding and is not the genotype on which the genetic operators act. This translation is computationally inexpensive.

$$COST_1 = \sum_{i=1}^{i=N} (M_1(m_i, i) + S_1(s_i, i, m_i)) \qquad (3)$$

M_1 is exactly the same function as M_0 from the previous section. $COST_1$ performs a simulation of the execution of the plan. Its input data are an ordered set of (machine, setup) pairs, one for each operation. Ordered sets of operations to be processed using a particular machine/setup combination are built up on a 2D grid. $S_1(s_i, i, m_i)$ governs the way in which these sets are built up. The operations in any set can be performed in isolation from those in any other set. Such a set will be referred to as a 'stage' of a job in the rest of 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 (processing of features). The final ordering of features is quite different from that on the genotype and the intermediate encoding, but deterministically derived (by S_1) from the genotype.

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 $COST_1$ provides a mapping from the process plan genotype to its phenotype: one of the plans illustrated above. This involves a considerable interpretative process, analogous to the developmental process in nature: there is a complex route from DNA (genotype) to organism (phenotype). The essential workings of $COST_1$ are sequentially to process the transformed genome in order to group operations together in clusters which can then be treated as single units (stages). At the same time as the costs are found a final ordering for the operations is produced. This encoding and cost function combination effectively allow a search of the combined ordering

and machine/setup selection space. Unlike the first suggested encoding/cost-function combination, a complex, but computationally cheap, interpretative process allows the use of computationally very inexpensive genetic operators.

The definition of $S_1(s_i, i, m_i)$ is given below:

$$S_1(s_i, i, m_i) = \begin{cases} f(s_i, m_i) & \text{if } s, m \text{ combination not previously encountered} \\ f(s_i, m_i) & \text{if } i \text{ causes break-constraint in all grid sets,} \\ f(s_i, m_i) & \text{if } i \text{ causes set incompatibility in all grid sets,} \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Where $f(s_i, m_i)$ is a simple table look-up function. Full details of the 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 feature x is being processed by $COST_1$. 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, $S0$, exists on the simulation grid. A break-constraint occurs, in an attempt to add x to $S0$, under the following condition:

$$\exists z \exists y((y \in S0) \wedge (z \in Sn) \wedge (Sn \neq S0) \wedge (y \rightarrow z) \wedge (z \rightarrow x)) \quad (5)$$

Where $\rightarrow$ represents a partial ordering constraint, y and z are features and Sn is any set on the grid. (Note that $\rightarrow$ actually refers to the transitive closure of the binary ordering operator shown in Figure 6. That is, throughout the rest of this paper, $a \rightarrow b$ means that a lies somewhere before b in the overall ordering graph, quite possibly involving chains of the given atomic constraints.) So a break-constraint occurs for feature x with the (m_x, s_x) combination when there exists some feature y which has already used this same combination, and has the following additional property: owing to the ordering constraints on the problem there exists a third feature, z, which must be processed after y but before x and does not use the same setup/machine combination. When a break-constraint occurs it is not possible to process all those features linked to a particular machine/setup combination without changing machine and/or setup partway through to perform 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 feature x to grid set $S0$, a new set, with x as the first member, is started at the same grid location.

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, $S0$, of features on the grid,

$$\exists y((y \in Sn) \wedge (Sn \neq S0) \wedge (y \rightarrow x)) \wedge \exists z \exists w((z \in Sn) \wedge (w \in S0) \wedge (w \rightarrow z)) \quad (6)$$

Where y, w and z are features and Sn is any set on the grid. The sets $S0$ and Sn are incompatible since it is not possible to order them in relation to each other. A new set must be started at the same position as $S0$ on the grid and with x as the first member. At the end of the simulation all the sets on the grid are ordered. 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 $S_1(s_i, i, m_i)$, 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 (defined below), then S_i is ordered before S_j. Formally, $S_i \rightarrow S_j$ if:

$$\exists x \exists y((x \in S_i) \wedge (y \in S_j) \wedge (y \in A_x^{\text{ext}})) \quad (7)$$

Where S_i and S_j are grid sets, x and y are operations and A_x^{ext} is the complete set of all operations lying after x in the overall partial ordering (extended after-constraints list).

It is this encoding and cost function that are used in the full parallel ecosystems model as described in the next section. Full details of the functions can be found in Husbands (1988b). A GA for optimizing single process plans, employing this encoding and cost function, is described by Husbands (1988a), and Husbands *et al.* (1990).

7. Coevolution, arbitrators and emergent scheduling

Sufficient detail has now been accumulated to fill out the sketch of the ecosystems model given in Section 4. Interacting populations of separate 'species', the genotype of each encoding for the process plan of a particular component and making use of the coding scheme described in Section 6.2, convolve under the influence of selection, crossover and mutation. Selective pressure takes into account interactions between the different populations (process plans) and hence allows the simultaneous optimization of the plans for each component, and the emergence of a near optimal schedule.

Recall that a separate species of arbitrators is 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 that encodes a table indicating which population should have precedence at any particular stage (defined

earlier) 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 abitrator genome is a bit string of length $SN(N-1)/2$, where S = maximum number of stages possible in a plan and N = number of process plan organism populations. Each bit is uniquely identified with a particular population pairing and is interpreted according to the function:

$$f(n_1, n_2, k) = g\left[\frac{kN(N-1)}{2} + n_1(N-1) - \frac{n_1(n_1+1)}{2} + n_2 - 1\right] \quad (8)$$

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.

In the full model the cost, hence selection, functions for plan organisms involve two stages, for arbitrators just one. The first stage involves population-specific criteria (basic machining costs) and the second stage takes into account interactions between populations. The first stage cost function for the process plan organisms is of course, $COST_1$, described in detail in Section 6.2. The second phase of the cost function involves simulating the simultaneous execution of plans derived from stage one. Additional costs are incurred for waiting and going over due dates. When two plans require the same resource at the same time an arbitrator is used to resolve the problem. The arbitrators are costed according to the amount of waiting and the total elapsed time for a given simulation. The smaller these two values are, the fitter the arbitrator is. Hence the arbitrators, initially randomly generated, are allowed to coevolve with the plan organisms. Each individual's fitness is calculated according to its total cost. This means that selection pressure takes account of both optimization problems: interactions during phase two that increase an individual's cost will reduce its chances of reproduction, just as will a poor result from phase one of the costing.

The first implementation of this model, on a MIMD machine, had the various populations on separate processors and involved a complicated ranking mechanism to allow coevolution to produce useful results (Husbands and Mill 1991), global selection was employed. The second, more satisfactory, implementation spreads each population 'geographically' over a 2D toroidal grid, which is illustrated in Figure 7. Selection is local, individuals can mate only with those members of their own

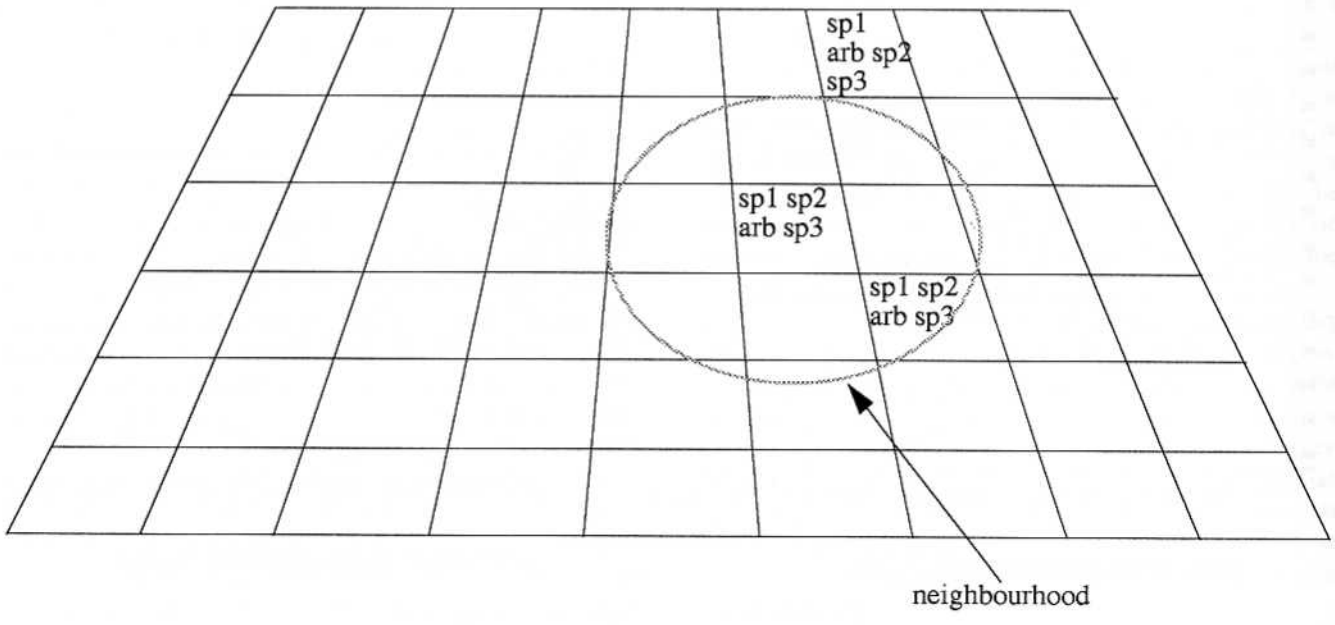
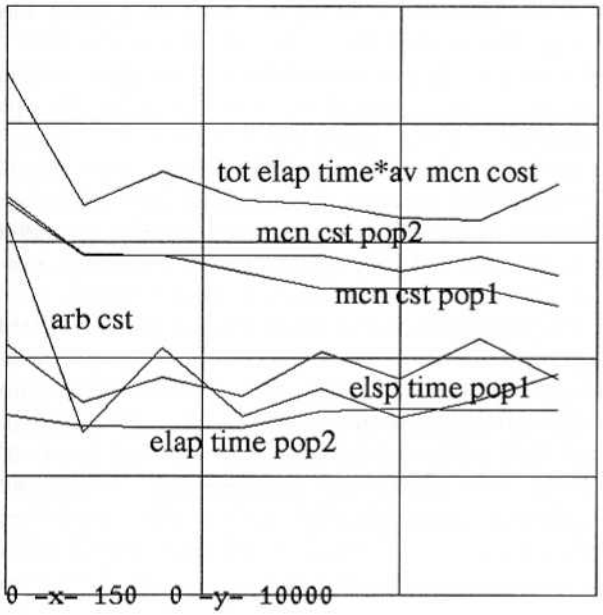
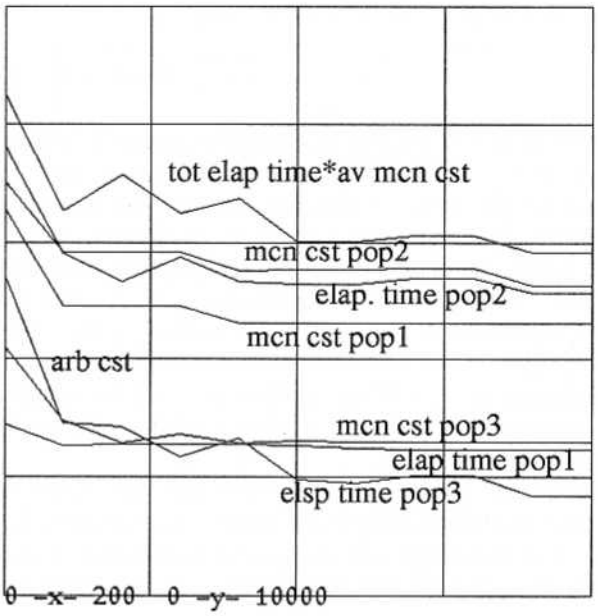


Figure 7. Geographically distributed population. Tiny fragment of 2D toroidal grid. Member of each species at each grid location.

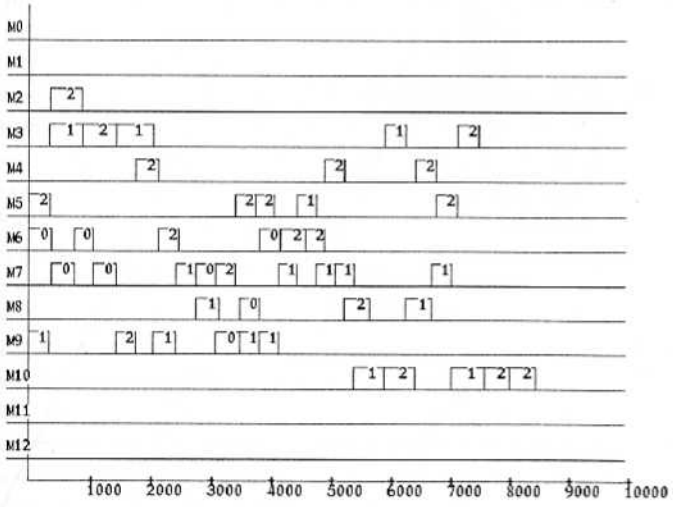
species in their local neighbourhood. The neighbourhood is defined in terms of a Gaussian distribution over distance from the individual; this results in a small number of individuals per neighbourhood. Neighbourhoods overlap, allowing information flow through the whole population without the need for global control. Selection works by using a simple ranking scheme within a neighbourhood: the fittest 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



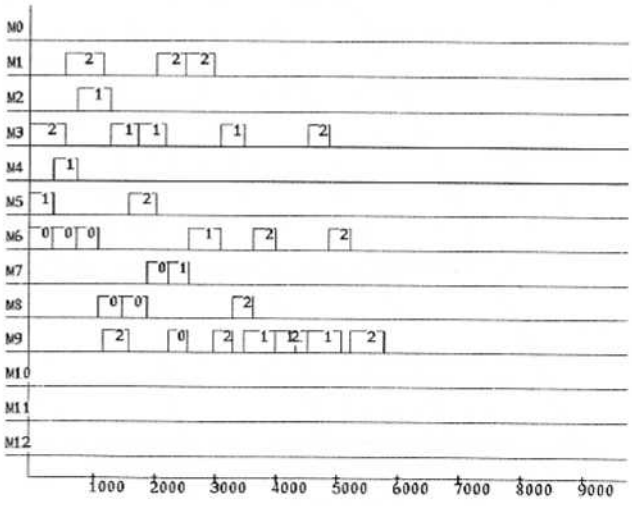
machining cost, total elapsed time against num generations (x axis)



machining cost, total elapsed time against num generations (x axis)



gantt chart, generation 0, 3 jobs



gantt chart, generation 180, 3 jobs

Figure 8. Results of coevolution model.

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 provides a highly parallel model that consistently provided better faster results than the first, less parallel, implementation.

Results of typical runs are shown in Figure 8. The graphs show how the machining costs ($COST_1$) 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 after a few tens of generations reflects the fact that machining costs can be decreased by putting more operations into a single stage. Clearly both optimization 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 eight candidate machines and each of these machines has six 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 10 jobs have been conducted. Very promising results have been obtained for this extremely complex optimization problem, never before attempted. The search spaces involved are unimaginably huge (greater than 10^{60}), but this model has exploited parallelism sufficiently to produce good results.

8. Discussion

Davis has done some work on using GAs to solve JSS problems (Davis 1985), 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, but that approach cannot exploit the inherent parallelism of the problem in the same way that the work described here has. GA work on other scheduling problems has been done by Clevland and Smith (1989), and Nakano and Yamada (1991). Hilliard *et al.* (1987) have used a classifier system (Goldberg 1989a) to discover scheduling heuristics. That work may tie in with ongoing research on enabling the arbitrators to learn how to resolve a number of different types 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 transputer), 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 words the simulated environment will dynamically mirror the actual manufacturing environment. Various local selection and interaction schemes are to be investigated in a new extended implementation of the model. As well as taking into account the general dynamic nature of a manufacturing environment, future work will make use of job priorities, varying start times and batch sizes. In the dynamic situation it is undesirable to allow the populations to converge too strongly on a single solution; potentially useful partial solutions may be lost for good. Local selection partly counteracts this tendency, but it is likely that a stochastic cost function will be necessary to fight it fully. A stochastic objective function would inject noise into the model. As well as preventing strong convergence, this is actually likely to provide a more accurate cost-model of the manufacturing processes. The scaling-up properties of the model will be investigated by using it with a large number of components (about 50).

Interesting coevolutionary GA systems have recently been developed by others (Hillis 1990, Koza 1990) in very different applications. The author is not aware of any other parallel GA systems that allow cooperative and distributed problem-solving in the manner of the work described here.

It may be possible to extend the method to encompass the notion of a 'total manufacturing system' in which the manufacturing facility is no longer fixed. Properties of the manufacturing environment would also coevolve along with the component plans. Variations in the environment could range from minor configuration details to major changes in the layout, number, types and properties of cells or machines.

In conclusion, this paper has presented preliminary results from a highly parallel GA-based 'ecosystems' model, which allows the simultaneous optimization of the process plans of a number of components. At the same time a near-optimal schedule for the job-shop emerges. Underlying the method is a feature-based process plan space generator. This work is aimed at completely re-evaluating the classical JSS problem and giving an indication of the best route forward in advanced parallel GA applications.

Acknowledgements

Initial parts of this work were funded by SERC (ACME) grants GR/D 63103 and GR/E 04837. The author would like to acknowledge useful discussions on this work with Frank Mill, Stephen Warrington, Inman Harvey, Jonathan Salmon, Joszef Vancza and Matthew Realff.

References

BALAS, E., 1969, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research*, **17**, 941–957.

BELEW, R., and BOOKER, L., 1991, Proceedings of the Fourth International Conference on Genetic Algorithms (Morgan Kaufmann).

CARLIER, J., and PINSON, E., 1989, An algorithm for solving the job shop problem. *Management Science*, **35**(2), 164–176.

CHANG, T., and WYSK, R., 1985, *An Introduction to Automated Process Planning* (Prentice-Hall, Englewood Cliffs, NJ, USA).

CHAPMAN, D., 1985, Planning for conjunctive goals. *Technical Report AI-TR-802* (MIT AI Laboratories).

CHRISTOPHEDES, N., 1979, *Combinatorial Optimisation* (Wiley).

CLEVLAND, G., and SMITH, S., 1989, Using genetic algorithms to schedule flow shop releases. In D. Schaffer (ed.) *Proceedings of the Third International Conference on GAs* (Morgan Kaufmann) 160–169.

DAVIS, L., 1985, Job shop scheduling with genetic algorithms. In J. Grefenstette (ed.) *Proceedings of the International Conference on Genetic Algorithms and their Applications*, (Lawrence Erlbaum) 136–140.

DAVIS, L. (ed.), 1990, *The Handbook of Genetic Algorithms* (Van Nostrand Reinhold).

GAREY, M., and JOHNSON, D., 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman).

GAREY, M., JOHNSON, D., and SETHI, R., 1976, Complexity of flowshop and jobshop scheduling. *Mathematical Operations Research*, **1**.

GOLDBERG, D., 1989, *Genetic Algorithms* (Addison-Wesley).

GOLDBERG, D., and LINGLE, R., 1985, Alleles, loci and the travelling salesman problem. In J. Grefenstette (ed.) *Proceedings of the International Conference on GAs and their Applications*, (Lawrence Erlbaum) 154–159.

GRAVES, S., 1981, A review of production scheduling. *Operations Research*, **29**(4), 646–667.

GREFENSTETTE, J. (ed.), 1985, *Proceedings of an International Conference on GAs and Their Applications* (Lawrence Erlbaum).

GREFENSTETTE, J. (ed.), 1987, *Proceedings of the Second International Conference on GAs*. (Lawrence Erlbaum).

HARVEY, I. 1992, Species adaptation genetic algorithms: a basis for a continuing SAGA. In F. Varela and P. Bourgine (eds) *Proceedings of the First European Conference on Artificial Life* (MIT Press).

HOLLAND, J., 1975, *Adaptation in Natural and Artificial Systems* (University of Michigan Press).

HILLIARD, M. *et al.*, 1987, A classifier based system for discovering scheduling heuristics. In J. Grefenstette (ed.) *Proceedings of the Second International Conference on GAs* (Lawrence Erlbaum) 231–235.

HILLIS, W. D., 1990, Co-evolving parasites improve simulated evolution as an optimisation procedure. *Physica D*, **42**, 228–234.

HUSBANDS, P., 1988a, The optimisation of process plans. *Technical Report 88-03* (Manufacturing Planning Group, Department of Mechanical Engineering, University of Edinburgh, UK).

HUSBANDS, P., 1988b, Further process plan optimisation work. *Technical Report 88-05* (Manufacturing Planning Group, Department of Mechanical Engineering, University of Edinburgh, UK).

HUSBANDS, P., 1992, Genetic algorithms in optimisation and adaptation. In L. Kronsjo and D. Shumsheruddin (eds) *Advances in Parallel Algorithms* (Blackwell Scientific Publications, Oxford) 227–276.

HUSBANDS, P., and MILL, F., 1991, Simulated co-evolution as the mechanism for emergent planning and scheduling. In R. Belew and L. Booker (eds) *Proceedings of the Fourth International Conference on Genetic Algorithms* (Morgan Kaufmann) 264–270.

HUSBANDS, P., MILL, F., and WARRINGTON, S., 1990, Generating optimal process plans from first principles. In E. Balagurusamy and J. Howe (eds) *Expert Systems for Management and Engineering* (Ellis Horwood, Chichester, UK) 130–152.

KOZA, J., 1990, Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. *Technical Report STAN-CS-90-1314* (Department of Computer Science, Stanford University, USA).

LIANG, M., and DUTTA, S., 1990, A mixed-integer programming approach to the machine loading and process planning problem in a process layout environment. *International Journal of Production Research*, **28**(8), 1471–1484.

MILL, F., PEDLEY, A., and SALMON, J., 1992, Representation problems in feature based approaches to design and process planning. *International Journal of Computer Integrated Manufacturing*, **6**, 27–33.

MUTH, J., and THOMPSON, G., 1963, *Industrial Scheduling* (Prentice-Hall, Englewood Cliffs, NJ, USA).

NAKANO, R., and YAMADA, T., 1991, Conventional genetic algorithms for job shop problems. In R. Belew and L. Booker (eds) *Proceedings of the Fourth International Conference on GAs* (Morgan Kaufmann) 474–479.

OW, P., and SMITH, S., 1988, Viewing scheduling as an opportunistic problem solving process. In R. Jareslow (ed.), *Annals of Operations Research*, **12**, (Baltzer Scientific Publishing).

SCHAFFER, J., (ed.), 1989, *Proceedings of the Third International Conference on GAs* (Morgan Kauffmann).

SCHWEFEL, H., and MANNER, R. (eds), 1991, *Parallel Problem Solving from Nature, Lecture Notes on Computer Science*, **496** (Springer Verlag).

SYCARA, K., ROTH, S., SADEH, N., and FOX, M., 1991, Resource allocation in distributed factory scheduing, *IEEE Expert*, (February 1991), 29–40.

TONSHOFF, H., BECKENDORFF, U., and ANDERS, N., 1989, FLEXPLAN - A concept for intelligent process planning and scheduling. In *Proceedings CIRP International Workshop on CAPP* (IFW, University, Hanover) 87–106.

VANCZA, J., and MARCUS, A., 1991, Genetic algorithms in process planning. *Computers in Industry*, **17**, 181–194.