

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.

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 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

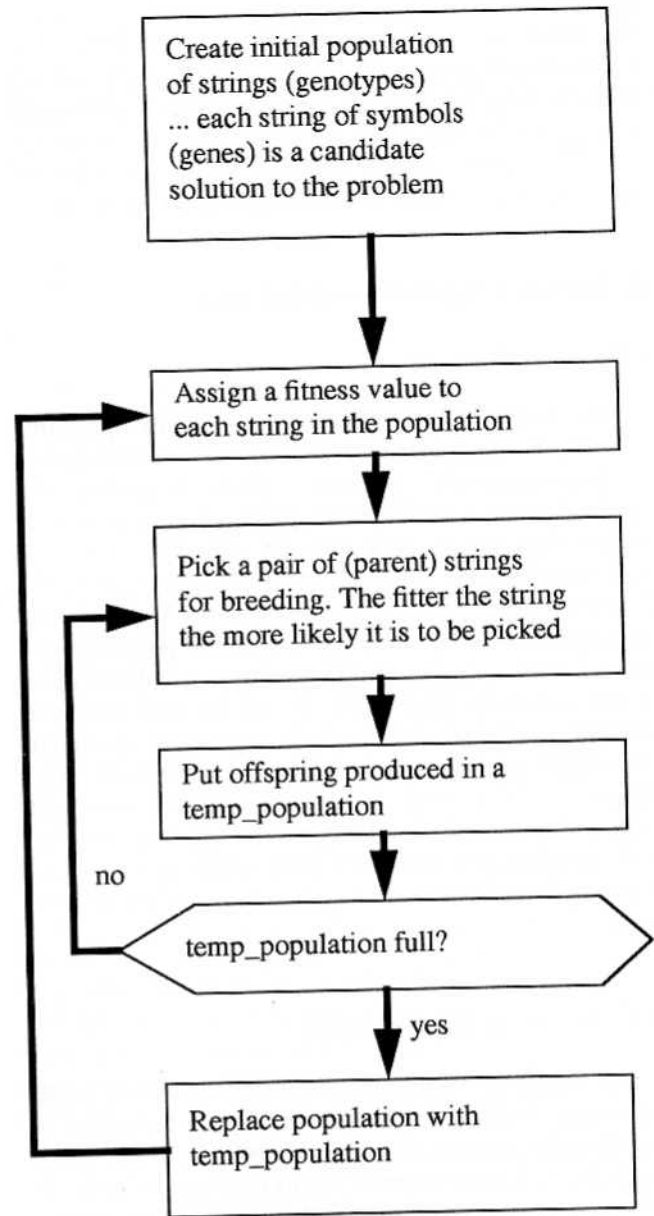


Figure 1. A simple genetic algorithm.

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.

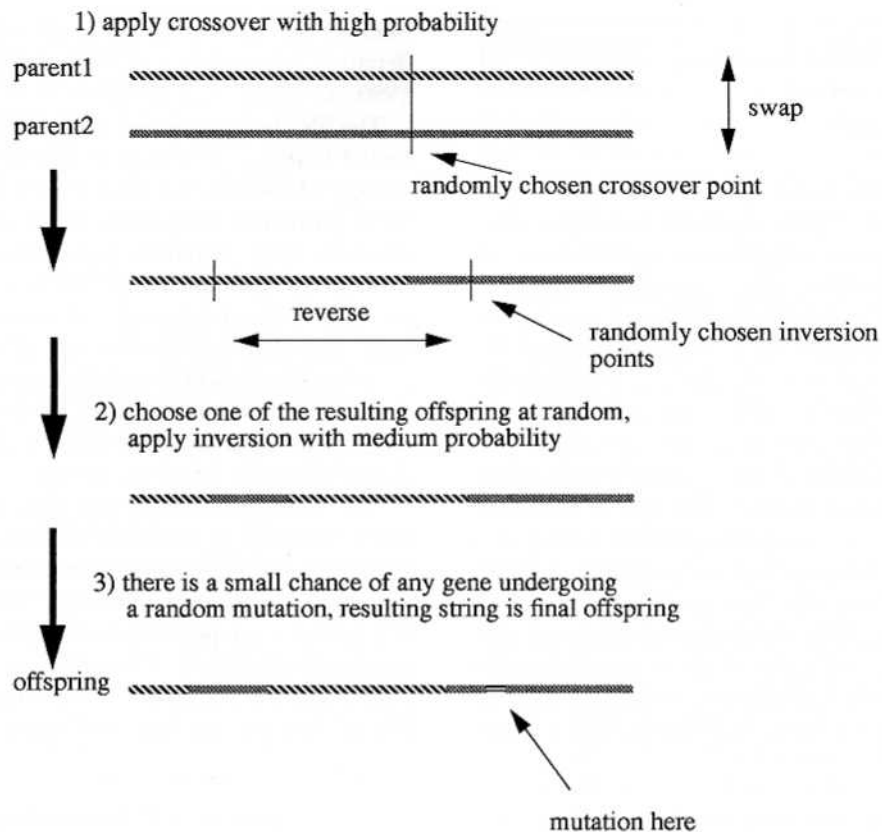


Figure 2. Application of the genetic operators.

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

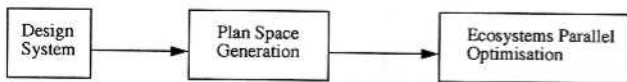


Figure 3. Overall approach.

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

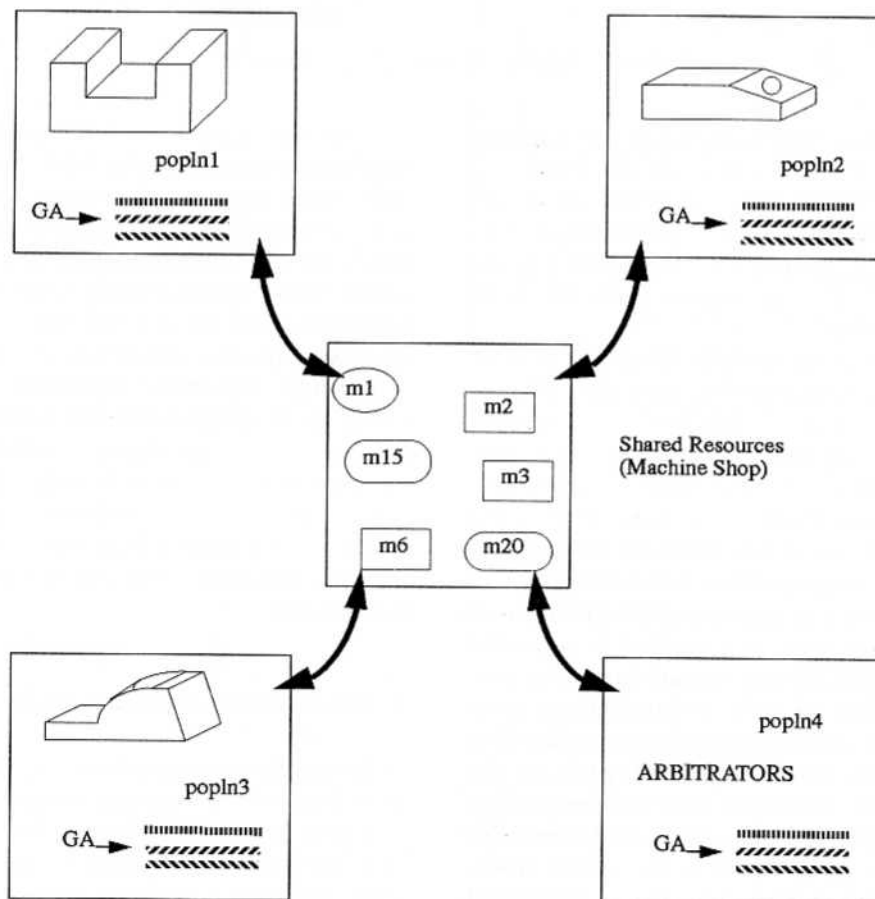


Figure 4. The ecosystems model.

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 builds and manipulates. These are planning networks like the one shown in Figure 5. In common with most generative process planners, the manufacturing processes are treated as material addition operations, whereas of course they actually involve material removal. The overall strategy is to start with those features deepest in the component and work out towards the surface. The process is guided by a set of 'critics' constantly on the look out for possible feature interactions, which may result in deferring work on part of the component (Husbands *et al.* 1990), and by high-level considerations regarding datums and such like. Once a feature has been chosen, a finishing process to achieve its desired final state is inferred. The details of this are discussed later. This finishing process leaves an intermediate feature with various inexact properties, such as a range of possible surface finishes. This models the fact that most finishing processes can only sensibly be started from a state with a given range of properties. For instance, it is highly undesirable to end up grinding down a very rough, uneven surface. A roughing process is then chosen to manufacture the intermediate feature. Remembering that an exhaustive set of possible manufacturing routes is required, for any given finishing process, any number of compatible roughing processes may exist. Thus a network like the one shown in Figure 5 is built up, keeping track of the interactions between finishing and roughing processes for each feature. These networks can be readily extended to allow an arbitrary number of sub-finishing and sub-roughing processes, and hence into

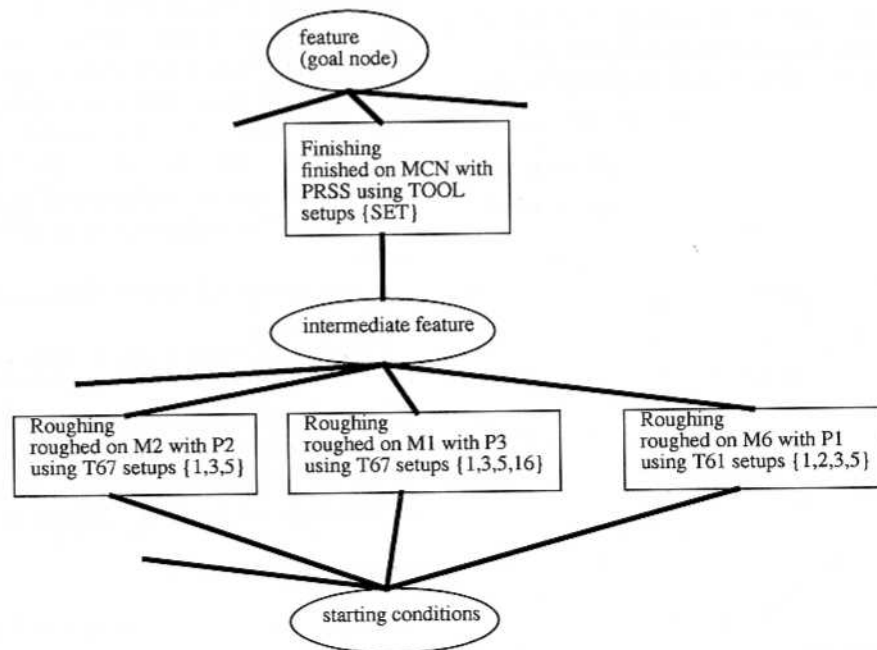


Figure 5. Fragment of planning network.

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

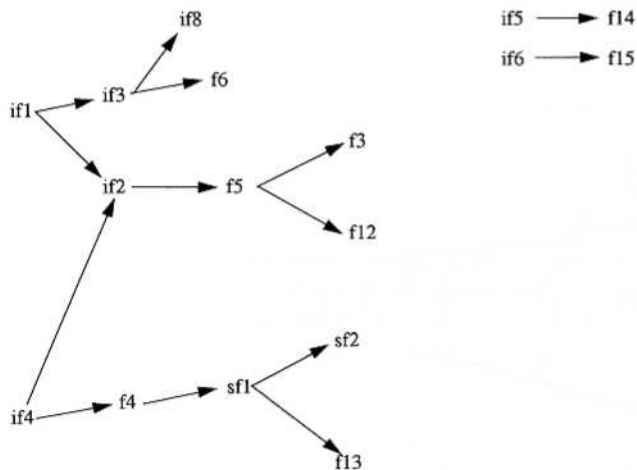


Figure 6. Anteriority constraints.

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.

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:

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

This is a simple order-based encoding to be read from left to right: f_i refers to the i th 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:

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

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:

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

Here f_i no longer refers to the i th feature to be processed in a plan, but to the i th 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.

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

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

