A genetic algorithm for the sequence coordination problem between two stages of a production system

Carlo Meloni

Dipartimento di Informatica e Automazione - Università Roma Tre

E-mail: meloni@dia.uniroma3.it

Feb. 22, 2001

Abstract

This paper deals with the coordination issues in multi–stage production systems and supply chains. Supply chain management is a relevant topic in logistics and operations management research, even though the *scheduling* of a supply chain did not yet receive the same attention so far. At the operational level, a number of interesting coordination problems exists, which have not yet been effectively modeled. These problems concern the way in which two consecutive stages should organize their internal work taking into account the requirements of the other stage. A central problem of operations management for multi–stage production systems is to decide how to sequence different parts which have to be processed in each stage, and what tools and materials to allocate to each stage in order to minimize the overall number of setups. Mainly, the focus of this paper is on the sequence coordination problem in a two–stage production system. A case arising in a real production system is considered, and a graph theoretical model for the problem is proposed. The model lead to a more general problem on graphs. A heuristic approach based on a genetic algorithm scheme is proposed. The computational experiments concern a wide set of instances involving also real industrial data.

1 Introduction

This paper addresses a problem arising in the coordination between two consecutive stages of a supply chain. Although supply chains are a hot topic in logistics and operations management research, the *scheduling* of supply chains has not received quite the same

attention so far, as pointed out in the recent comprehensive review by Thomas and Griffin (1996). At the operational level, a number of interesting coordination problems exist which have not yet been effectively modeled. These problems concern the way in which two consecutive stages should organize their internal work, taking into account the requirements of the other stage.

The models considered in this paper arise from an application in a real industrial context, namely a plant in which a large number of different slabs of wood are cut, painted and assembled to build kitchen furniture. In different departments of the plant, items are grouped according to different attributes. In the cutting department, parts are grouped according to their shape and material, in the subsequent painting department parts are grouped according to their color, and finally the assembly of the furniture is organized on the basis of kitchen models. This paper analyzes the coordination issues between the first two stages, and therefore the interest is in only two attributes, which for simplicity are called *shape* and *color*.

In both departments, a *set-up* occurs when the attribute of a new part changes. If a part must be cut having a different shape from the previous one, cutting blades and machinery must be reconfigured. Similarly, in the painting station, when a new color is used, the equipment as well as the pallets must be thoroughly cleaned in order to eliminate all the residuals of the previous color. Hence, in both cases costs are incurred in terms of time and manpower.

In this application, limited interstage buffering between the two stages forces the sequences in which the items are processed in the two stages to be coordinated. In this paper, we analyze the case in which the two departments follow the same common sequence. Clearly, each department would like to minimize its own total set–up cost, but the objectives of the two departments may be conflicting.

Some additional issues may exist in real life situations. For instance, not all set-ups may have the same cost. In the painting department, switching from yellow to blue may require less time than from blue to yellow. In some cases, there may even be hard precedence constraints that limit the set of feasible sequences in the two departments. On the other hand, there are some fixed activities involved in a changeover which are very much the same, regardless of the particular changeover, and the contribution of these fixed activities to the set-up cost may be extremely relevant. Hence, the total number of set–ups is usually a meaningful index of performance.

In this paper, the problem in its basic version is analyzed, i.e., when all the setups in each department and across departments have the same cost and no resequencing is possible between the two stages. Hence, the focus is on the *number* of set–ups, and the objective is to establish a *single* common sequence of items which both stages will follow. Even so, the problems turn out to be hard and call for a heuristic solution approach.

Each item to be produced has its own shape and color. All the items having the same shape and color form a single *batch*. In fact, there is no convenience, on either side, in splitting such batches. In the first (second) department, a set–up is paid when the new batch has a different shape (color) from the previous one. Otherwise, no set–up is incurred. Note that since the objective is to minimize the number of setups, the actual cardinality of each batch is of no interest here.

Each given sequence of the batches results in a number of set–ups to be paid by each department. In this paper two different objectives are considerd, namely:

- MiTS: Minimize the total number of set–ups in the two departments
- MiMS: Minimize the maximum number of set–ups paid by either department

While the former objective corresponds to the maximization of overall utility, the latter may better capture the need to balance the set–ups between the two departments.
The manufacturing example previously described, requires to solve two different problems, namely MiTS and MiMS. Actually, the MiTS problem corresponds to solve the *Minimum cardinality Dominating Trail Set* (MDTS) problem on bipartite graphs, while MiMS problem is not combinatorially characterized yet. In what follows heuristic approaches for both problems are proposed.
Notice that, given the bipartite graph $T = (S, C, A)$, the MDTS problem can always be solved finding the hamiltonian path of minimum weight on a complete graph $K_{|A|}$ (with one node for each edge of T), in which the weight associated with each edge (x, y) of $K_{|A|}$ is 0 if $x \in A$ and $y \in A$ are adjacent in T and 1 otherwise.
Clearly, there is a one-to-one correspondence between hamiltonian paths in $K_{|A|}$ and batch sequences, the weight of a hamiltonian path being equal to the number of global changeovers in the corresponding sequence.
However, the size of graph $K_{|A|}$ can easily be too large to find the minimum weight hamiltonian path within the time constraints practitioners are usually faced with. Let the *density* of T be defined as $|A|/|S||C|$. A problem with 50 shapes and 50 colors and density 0.4 results in a graph $K_{|A|}$ with 1000 nodes. Moreover, it is well known that instances of the traveling salesman problem with 0-1 costs are particularly hard.
Finally, in real industrial applications as those in the examples, it may be important to have a method which is time–limited, in particular if the sequencing problem has dynamic aspects. In fact, in the case of large instances, by the time the problem is *solved*, it may have changed (e.g. more new parts may have arrived), so a method taht takes too long time may be inappropriate whatever the quality of the solutions it produces for a static problem. For these reasons we do not consider formulations based on TSP.
An alternative 0-1 integer programming formulation of these problems is suggested in Crama et al. (1996). The size of these formulations are smaller then those obtained in the case of

Hamiltonian path or TSP formulations. However, their linear programming relaxation provides a lower bound extremely weak. Hence, improvements are needed in the formulations in order to make it effective. These improvements may consist in some vaild constraints to add to the formulations, i.e. lower bounds on the objective function.
The difficulties presented in order to obtain an effective ILP formulation suggests to consider an heuristic approach to MDTS as well as to MiMS and MiTS problems.

2 A Genetic Algorithm approach

The concepts of genetic algorithms (GAs) have been applied successfully to problems involving pattern recognition, classification and other problems in the domain of Artificial Intelligence. Some literature in the recent years reports about applications to large combinatorial problems. This section attempts to apply GAs to two such problems, namely the MDTS (MiTS problem) and MiMS problems introduced early in this paper.
GAs have many possible choices of control parameters, but the emphasis here is on showing the feasibility of using GAs for such problems by producing a working algorithm, rather than on a search for the optimal parameter settings.
The basic concept of GAs are briefly described, then a GA scheme is developed for the sequencing problems related to the coordination in the manufacturing context. This approach provides a means to deal with the MiTS and MiMS problems. The performance on MiTS problem may be compared with those of the DOMWALKS heuristic proposed in literature (Agnetis et al.).
A part of the research community is involved in the design and development of heuristics for NP–hard problems. One such problem, known to be NP–hard, is the permutation flowshop sequencing problem in which n jobs have to be processed (in the same order) on m machines. The object is to find the permutation of jobs which will minimize the makespan, i.e. the time at which the last job is completed on machine m. For this problem Reeves (1995) proposes a GA solution scheme that perform relatively well on large instances. The proposed GA approach, which starts from the Reeves scheme, is dedicated to the permutation sequencing problems MDTS, MiTS and MiMS. In particular, these problems are viewed as special cases of permutation scheduling on a single processor in which the edges of the graph are the jobs to be processed, and the objective is to minimize the number of setups as discussed previously.
Nowadays much has been published on applications of GAs to combinatorial optimization and sequencing. Several genetic *operators* can be identified, the most commonly used ones being crossover (an exchange of sections of the parents' chromosomes), and mutation (a random modification of the chromosome). In the context of finding the optimal solution

to a large combinatorial problem, a GA works by maintaining a population of M solutions which are potential *parents*, whose *fitness values* have been calculated. In Holland's original GA, one parent is selected on a fitness basis (i.e. the better is the fitness value, the higher is the probability of it being chosen), while if crossover is to be applied, the other parent is chosen randomly. They are then *mated* by choosing a crossover point X at random, the offspring consisting of the pre–X section from one parent followed by the post–X section of the other. Offsprings replace a part of the existing population chosen by means of a probabilistic rule. The *reproduction* is iterativelly repeated. Various versions of this procedure are been proposed in literature and different strategies are been adopted.
As well as methodological or strategic decisions, there are parametric choices to do: different crossover and mutation probabilities, different population sizes an so on. It is also quite possible that the way the initial population is chosen will have a significant impact on the results.
In order to apply any GA to a sequencing problem, there is an obvious pratical difficulty. In most *traditional* GAs, the chromosomal representation is by means of a string of 0s and 1s, and the result of a genetic operator is still a valid chromosome. This is not the case if one uses a permutation to represent the solution of a problem, which is the natural representation for a sequencing problem. For example, if the parents are as shown below,

Parent 1: $(2, 1, 3, 4, 5, 6, 7)$ Offspring 1: $(2, 1, 3, 2, 5, 7, 6)$

Parent 2: $(4, 3, 1, 2, 5, 7, 6)$ Offspring 2: $(4, 3, 1, 4, 5, 6, 7)$

it is clear that choosing the crossover point between the third and the fourth symbol, the offsprings are *illegitimate* because symbols do not appear once in the sequence.
Different representation are possible, but it seemed easier to modify the idea of crossover and keep the permutation representation. Two modification are proposed by Reeves (1995). The first crossover chose (C1) consist in to select randomly one point X along the parents' chromosomes, to take the pre–X section of the first parent, and to fill up the chromosome by taking in order each *legitimate* symbol from the second parent. The application of this rule to previous example yields:

Parent 1: $(2, 1, 3, 4, 5, 6, 7)$ Offspring 1: $(2, 1, 3, 4, 5, 7, 6)$

Parent 2: $(4, 3, 1, 2, 5, 7, 6)$ Offspring 2: $(4, 3, 1, 2, 5, 6, 7)$

The idea behind C1 is that it preserves the absolute positions of the symbols taken from the first parent, and the relative positions of those from the second. It was conjectured that this would provide enough scope for modification of the chromosome without excessively disrupting it. Preliminary experiments showed that this crossover rule tended to converge rather too rapidly to a population of almost identical chromosomes (premature convergence), so a mutation operation is also required. In other words the crossover C1

emphasizes *exploitation* instead of *exploration*; using a mutation provides a way to restore the balance.

The premature convergence problem is one that has been noted by several workers in applying GAs, and various proposals have been made to prevent it. The method applyed here is to use an adaptive mutation rate p_m. At the start of the algorithm an high probability of mutation is allowed; this probability is slowly decreased as long as a reasonable diverse population exists. If the diversity becomes too small, the original high value is restored. Several measures of diversity may be adopted, our choice is to reset the mutation rate if the ratio $\frac{f_{min}}{f_{avg}}$ exceeded a value D_{th}, where f is the fitness value of each chromosome (that is related to the objective of the problem to solve).

Another approach (C2) considers two crossover points X_1 and X_2. The pre–X_1 section of the first parent is retained, and the *legitimate* symbols of the post–X_2 section of the second parent. The gaps are filled in by a random choice from symbols unused yet. It was expected that this would disrupt the chromosome much more than C1, so that mutation would not be needed, and this appeared to be true in experiments. Anyway, disruption seemed to be excessive, and as the solutions converged very slowly, it seems useful to concentrate on C1 rather to try to improve C2.

The selection of parents for the reproduction process (by means of the crossover operator), is a simple ranking mechanism for the first parent, while the second parent is selected randomly over the current population. At this aim, the population is ordered on the basis of the fitness of its elements. Howerver, the ranking selection mechanism involves only K elements of the population with a better fitness than the current average fitness. Such a mechanism works in accordance with the probability distribution $p([i]) = \frac{2i}{K(K+1)}$, where $[i]$ is the ith chromosome in ascending order of fitness (for example, descending order of MiTS or MiMS), and K is the number of elements involved in this selection.

This implies that the median value has a probability of $\frac{1}{K}$ of being selected, while the Kth (the fittest element) has a probability of $\frac{2}{(K+1)}$, about twice that of the median.

The ranking mechanism is also used for choosing elements to be eliminated at each iteration, i.e. substituted by a offspring. Since we already have a ranking of the chromosomes, it is easy to choose from elements which have a bad fitness.

As already indicated, a mutation operator was used to reduce the rate of convergence. In traditional GAs, mutation of a chromosome is performed by *flipping* each of its symbols from 0 to 1 (or vice versa) with a small probability. In our case, with a permutation representation, mutation needs to be defined differently.

Three types of mutation are considered. The first, an *exchange* mutation, was a simple exchange of two symbols in a permutation, chosen at random. The second, a *shift* mutation, i.e. a shift of one symbol (again, chosen randomly) a random number of places to the right or to the left. The third type is the *reversal* mutation consisting in change the selected

chromosome with its reverse. Note that in the latter case the fitness value remains the same.

Another consideration deals with the selection of the initial population. Most GAs assume that the initial population is chosen completely at random. However, it is worth trying the effect of *seeding* the initial population with a good solution generated by a constructive heuristic. This is the case of MDTS and MiTS problems, that my be tackled with the heuristic DOMWALKS; while there is not a specific heuristic for MiMS, and it may be useful to try with the best MiTS solution as upper bound for this problem. This suggests an algorithmic scheme consisting of two different subsequent stages, one dedicated to MiTS and another to MiMS.

The GA scheme proposed has not a natural stopping point. In order to avoid too long computation time it is useful to introduce some stopping criteria. The simplest method consist to limit the number of algorithm iteration or the total computational time. This method may be refiend considering also the evolution of the algorithm and starting iteration/time counters only when iteration does not improve the population fitness (best value or average value). The stopping criteria may also take into account eventually known lower and upper bounds on the objective function (Meloni, 2000). The first avoids to continue in the search when an optimal sequence is already found, while the second may force the algorithm to continue the search in order to improve the current solution.

Experiments are made on a wide set of instances also from real industrial cases. After a preliminar phase for testing and tuning the algorithm, a particular setting of parameters is adopted and some innovations are introduced with respect to the basic version of the algorithm. We consider a variable population up to 100 individuals, a probability of mutation of 3.5%, and choosing (time by time) at random the mutation operator among those proposed (i.e., reverse, shift and swap).

A particular evolution scheme is implemented in which two genetic algorithms evolve in parallel (starting with different initial random population) in the first phase. This phase ends when the average fitness value of each population results between two theoretical bounds (UB and LB). In the second phase the populations are merged and a new genetic algorithm starts. Also the crossover operator is implemented in an improved version: when the chromosomes are almost identical (i.e., when the number of identical elements is greater than a fixed value S_{th}) one of them is reversed before the crossing over.

The algorithm solved the 78% of the real world instances. It takes always less than 30 seconds and an average computing time of 18.4 seconds. The results on the artificial instances allow us to point out the complementary behavior of the GA approach with respect to the DOMWALKS heuristics. In fact, GA performs well when the graph has a low density as shown on a large set of instances. The time required by GA is almost always higher than those taken by DOMWALKS. However, GA may obtain better results than DOMWALKS

in a subset of the instances.

3 Conclusions

These experiments suggest to consider an hybrid heuristic based on DOMWALKS and GA. In this new scheme GA will play the role of a local search tool receiving the solution of DOMWALKS in its population to obtain a *seeding effect*. This appears to be a fast and powerful algorithm for the considered problems. In fact, also the MiMS problem may be tackled by the same approach considering that the solution of the MiTS problem may be view as an upper bound. The proposed algorithms found ideal sequences on real-life instances and on most of a large sample of randomly generated instances. Even when an ideal sequence was not found, the number of global changeovers of the solution provided by the algorithm was always very small. These results suggest that future research should address topics which have not been covered here, such as: 1) the case in which set-ups have different costs in the two stages; 2) algorithms which account for both the total and the maximum number of set-ups across the two stages; 3) the extension of MiTS to include also another stage of the supply chain; 4) sequence-dependent set-up costs, as those occurring when, for instance, it takes less time to switch from a lighter to a darker color than viceversa.

4 Bibliography

Agnetis, A., Detti, P., Meloni, C., Pacciarelli D., (1999), *Set-up coordination between two stages of a supply chain*, Tech. Rep. n. 32–99, Dip. di Informatica e Sistemistica, Università La Sapienza, Roma. To appear in Annals of Operations Research.

Crama, Y., Oerlemans A.G., Spieksma F.C.R., (1996), *Production planning in automated manufacturing*, Springer, Berlin.

Holland, J.H., (1976), *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor MI.

Meloni C., (2000), *The splittance of a graph and the D-trail problem.* Università di Roma Tre, Dip. di Informatica e Automazione, RT-55-2000.

Reeves, C.R., (1993), *Modern heuristic tecniques for combinatorial problems*, Blackwell Scientific Publications, Oxford.

Reeves, C.R., (1995), *A genetic algorithm for flowshop sequencing*, Computers & Operations Research, 22, 5–13.

Thomas D.J., Griffin P.M., (1996),*Coordinated supply chain management*, European Journal of Operation Research, 94, 1–15.