Intelligent Disruption Recovery for Decentralized Manufacturing Systems

Thomas Tsukada Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

This paper considers the problems of intelligent response to disruptions in a decentralized manufacturing system. In the absence of intelligent coordination, the response to a disruption of one part of the system may cause a disruption in another part of the system. This paper presents a model for the problem of recovering from this kind of disruption, and presents a solution approach based upon the idea of negotiation from the field of distributed artificial intelligence. The model and solution approach are evaluated in the domain of job shop rescheduling.

1 Introduction

Decentralized process control is fast becoming an important concept in manufacturing systems research. While advances in computer hardware have made possible centralized control of a large manufacturing process at one powerful computer, advances in distributed computing are allowing the decentralization of control among a less expensive coordinated group of less powerful computers. Decentralization has several advantages, allowing better fault tolerance, easier modifiability, and parallelism. Decentralized approaches also take advantage of the distributed nature of the manufacturing process. Decentralized systems have been discussed in [6, 8].

Flexible manufacturing systems must be able to recover from disruptions without halting the entire system. In a decentralized system, intelligent run-time coordination is required for this flexibility. In a heterarchical cellular manufacturing system, for example, if a disruption occurs at a cell, that cell's controller must take some action to respond. However, this action may affect some other cell, causing a disruption in that cell, requiring action from its controller. In such a way, a disruption at one cell may propagate through the whole cellular system.

A good example of this type of propagation of disruptions can be seen in the rescheduling of a cellular manufacturing system. If one cell suffers a disruption, such as a down machine or the arrival of a new unexpected job, then the jobs at that cell must be rescheduled. However, because of precedence constraints or resource sharing, this rescheduling may disrupt the schedules of other cells, by the late arrival of parts or resource unavailability. Thus, a disruption in the schedule of one cell may result in the schedule disruptions at other cells which were otherwise undisrupted.

The problem that we address in this paper is that of decentralized disruption recovery, how a decentralized manufacturing system can deter the propagation of disruptions through communication and cooperation among local controllers. The effect that an action taken by a local controller has at a remote cell depends not only upon the action but also on the state at the other cell. Thus, when choosing a good response to a local disruption, a controller must take into consideration the states at other cells; some form of communication and cooperation is required to deter the propagation of disruptions. To approach this problem, we propose the use of distributed planning theory and distributed constraint satisfaction theory from the field of distributed artificial intelligence (DAI). The domain in which we consider this problem is that of rescheduling in cellular manufacturing systems.

The remainder of this paper is organized as follows. In Section 2, we present a formal model of the decentralized disruption recovery problem, and a general approach to this problem. In Section 3, we examine this problem in the domain of job shop scheduling, and present an algorithm for rescheduling in a cellular job shop. Numerical results of this algorithm are presented and discussed in Section 4. Section 5 concludes

The work described in this paper was supported in part by the NSF Under Grant IRI-9209031.

this paper.

2 The decentralized disruption recovery problem

Manufacturing cells can affect one another in many different ways: shared resources, transfer of parts, control over the shop floor environment, and others. In this section we present a very general model of the decentralized disruption recovery problem, and a general approach to this problem based on the idea of *negotiation* from DAI. This general model and approach do not provide solutions to any specific problems, but do provide an outline for developing solution algorithms in more specific application domains.

2.1 Model

We model disruption recovery in a manufacturing system of n loosely coupled manufacturing cells. Let $C = \{C_1, C_2, \ldots, C_n\}$ be the set of cells. For each cell C_i, there is a set $S_i = \{s_{i1}, \ldots, s_{im}\}$ of states. For each cell C_i, there is a set $D_i \subset S_i$, which is the set of *disrupted* states. A state in S_i which is not in D_i is a *safe* state. There is a set $A_i = \{a_{i1}, \ldots, a_{ip}\}$ of disruption recovery actions which can be taken at cell C_i. A disruption is an event which causes a cell to be in a disrupted state. For a disruption d at cell C_i, let $R_{di} \subset A_i$ be the set of proper recovery actions that cell C_i can take to recover from disruption d.

For each pair of cells C_i and C_k, $k \neq i$, there is a transition function $\mathcal{F}_{ik} : A_i \times S_k \rightarrow S_k$ which describes how an action taken by cell C_i affects cell C_k. Thus, if cell C_k is in state s_{kl} and cell C_i takes action a_{ij}, then cell C_k will be put into state $\mathcal{F}_{ik}(a_{ij}, s_{kl})$.

For each action a_{ij}, let $M_{ij} = \{C_k : \exists s_{kl} \in S_k, \mathcal{F}_{ik}(a_{ij}, s_{kl}) \in D_k\}$. That is, M_{ij} is the set of cells which could possibly be disrupted if cell i takes action a_{ij}.

We call $x = (x_1, x_2, \ldots, x_n)$ the system state if cell C_i is in state s_{ix_i}, for all i. Given a system state x, let $N_{ij} = \{C_k : \mathcal{F}_{ik}(a_{ij}, s_{kx_k}) \in D_k\}$. N_{ij} is the set of cells which will be disrupted if cell C_i takes action a_{ij}, given system state x.

In order to recover from a disruption d at cell C_i without propagating a disruption, we want to find an action a_{ih} such that $a_{ih} \in R_{di}$, and $N_{ih} = \emptyset$. We assume that the controller at cell C_i knows the set M_{ij} for every disruption recovery action in A_{ij}, but that it does not know the system state, and thus cannot determine, without communication, what the set N_{ij} is for any of the disruption recovery actions in A_{ij}. For example, cell C_i may know that it shares a particular resource with cell C_x. Without communication, however it will not know whether use of this resource at any given time will conflict with C_x's use of it.

2.2 Outline of approach

This model does not include a central controller. DAI research has for the past decade studied similar problems in which a group of distributed intelligent agents must interact to solve a problem. Distributed planning, an important area of DAI, involves the construction of a global plan through the interaction of problem-solving agents, where there is no central coordination of the agents' activities, and no individual agent has complete knowledge of the problem to be solved. Negotiation among agents is an important part of distributed planning [1, 5]. We apply ideas about negotiation and intelligent interaction among agents in the following approach to the decentralized disruption recovery problem.

When a disruption d has been detected and identified at cell C_i, C_i uses some heuristic H to pick a fault recovery action $a_{ih_1} \in R_{di}$. Cell C_i then sends a proposal message to all members of M_{ih_1}, proposing action a_{ih_1} (recall that M_{ih_1} is the set of cells that can potentially be disrupted by cell C_i taking action a_{ih_1}.) Cell C_i then waits for replies from members of M_{ih_1}. If all the replies are ok replies, then $N_{ih_1} = \emptyset$, so cell C_i takes action a_{ih_1}, and then presumably the disruption has been handled.

If, however, there is a not-ok reply, then $N_{ih_1} \neq \emptyset$, so the following loop is executed, starting with $j = 1$. At the beginning of the loop, if cell C_i determines that further negotiation will be useless (either because a solution cannot be found, or because of time constraints), then cell C_i chooses (using some measure) the best of $\{a_{ih_1} \ldots a_{ih_j}\}$, and takes that action (thus resolving the local disruption, but causing one or more disruptions at other cells). Otherwise j is incremented, and a new a_{ih_j} is chosen using heuristic H, and proposed to members of M_{ih_j}. If all the replies are ok messages, then $N_{ih_j} = \emptyset$, so cell C_i takes action a_{ih_j}, and the disruption has been handled. If not, then $N_{ih_j} \neq \emptyset$, so the loop is executed again.

When a cell C_k ($k \neq i$) receives a proposal message from cell C_i proposing action a_{ih_j}, it determines whether the action a_{ih_j} will cause a disruption at C_k. If not, then cell $C_k \notin N_{ih_j}$, so C_k returns a ok reply message to cell C_i. Otherwise, $C_k \in N_{ih_j}$, so the cell returns a not-ok reply message, perhaps along with some information I_{kj}, to cell C_i. This information I_{kj} is likely to be some indication of what state C_k is in,

but often a full description of this state will be too expensive to send. This information is used at cell C_i by the heuristic H in order to determine a better proposal.

The preceding approach is only a simple outline of an algorithm for solving the decentralized disruption recovery problem. Obviously, the heuristic H, the information I_{kj}, the means of determining when further negotiation will be useful or not, and the means of determining the best of the rejected proposed actions, are very domain-specific, and cannot be more fully described given this very general model. Nevertheless, this approach will be useful in determining decentralized disruption recovery algorithms for more specified domains. Results from the study of distributed constraint satisfaction problems suggest the use of variable and value heuristics [7] and resource demand profiles [10] for determining what possible solutions to try first.

This approach is somewhat different from distributed planning. In distributed planning, multiple agents cooperate to solve problems which they have in common. In our approach, agents cooperate in order to prevent local problems from becoming common problems. When cooperating with a disrupted cell, undisrupted cells do not help the disrupted cell find a good solution; they merely help it find a solution which will not disrupt themselves.

3 The rescheduling domain

In this paper, we study the disruption recovery problem in the domain of job shop scheduling in a cellular manufacturing system. The scheduling domain is chosen to investigate this problem, because it contains easily definable interactions among cells, in the form of precedence constraints among jobs.

One important and often overlooked aspect of scheduling is the actual execution of an already-constructed schedule (a *preschedule*). Oftentimes, the real situation on the shop floor is different from that assumed in the scheduling process. Machines may break down, new unexpected jobs may arrive, release dates may change, etc. Such unexpected events may render a preschedule infeasible. Thus, if there is no provision for dealing with such events, even an optimal scheduling method may be completely irrelevant.

One approach to handling unexpected events is dynamic scheduling, in which no preschedule is constructed. All scheduling decisions are made at run-time, by dispatch rules [4], or by least-commitment opportunistic planning [9]. The advantages of dynamic scheduling are that it is computationally easy, and that it is robust in unpredictable environments. Another approach is to construct a new schedule when events render the old one infeasible. One very fast way of doing this is to "push back" the existing schedule until it becomes feasible. This method is widely used in practice, but very often produces an inefficient schedule. Another method, much more slow and resource-intensive, is to construct a completely new schedule using the same (presumably good) scheduler which produced the preschedule.

These approaches do not make use of a good preschedule. We choose instead to follow the matchup scheduling approach of Bean et al. [3]. In this approach, when unexpected events disrupt the preschedule, the scheduler attempts to schedule production so that the system can return to ("match up with") the original preschedule. Thus, the good preschedule need not be discarded when disruptions occur. The preschedule must be robust enough so that the system can return to it in a reasonable amount of time, and the system must be rescheduled efficiently until it matches up with the preschedule. The matchup scheduling paradigm reduces the problem of recovering from disruption to the narrower problem of returning to the preschedule.

3.1 The decentralized job shop rescheduling problem

We consider a job shop scheduling problem in which tardiness is the performance measure. When precedence constraints are considered, the problem of rescheduling a cell in a decentralized cellular manufacturing system easily fits into the model presented in Section 2. The set S_i of states for cell C_i is the set of all possible schedules for C_i, given the set of jobs which must be processed at C_i. Those states which represent infeasible schedules are the *disrupted* states. The set of actions A_i is the set of all feasible rescheduling actions for C_i, given the characteristics of the machines and jobs at C_i. Thus, A_i is as big as the set of all feasible schedules for C_i.

Following the matchup scheduling paradigm discussed above, we assume that each cell has a preschedule which is followed in absence of any disruptions. The rescheduling action a_{ij} of cell C_i may disrupt the schedule of another cell if C_i in the new schedule delays processing of any job which must precede jobs at other cells. We assume that if completion of a job on C_i must precede some other job at another cell, C_i will know the identity of this other cell, but not its scheduling details. Thus M_{ij} is the set of cells having

jobs which must succeed one of the jobs delayed at C_i by rescheduling action a_{ij}, and C_i knows M_{ij}.

3.2 Algorithm for decentralized job shop rescheduling

If the preschedule of a cell becomes infeasible because of a disruption, then the cell must reschedule. As in the matchup scheduling paradigm, we assume that there is some finite set of jobs which can be rescheduled in response to the disruption, such that the remainder of the cells schedule can remain intact. We assume that this set of jobs has already been identified by one of the methods described in [3]. Let B denote this set of jobs.

To solve these rescheduling problems, we take the approach previously outlined in Section 2.2. Because the propagation of a disruption to another cell can occur when a job is completed late, we would like to have a heuristic which helps produce a schedule in which every job which has successors is done on time. One heuristic which we make use of in this algorithm is the MDD (modified due date) dispatching rule. The modified due date for a job at a given time is the maximum of its processing time and the time remaining before its due time. This rule has been proven to be a good simple heuristic to keep tardiness low [2]. Likewise, the information communicated in the not-ok reply from a remote node should indicate by what times those jobs are expected to have been completed. Thus, we use the following rescheduling algorithm.

When the local node recognizes a disruption and determines that job set B must be rescheduled:

```
for all j_i ∈ B
  d_i := ∞;
reschedule B by MDD;
if c'_i ≤ c_i ∀j_i ∈ P
  then stop; {new schedule is good}
else
  while not done
    do begin
      for each job j_i ∈ P
        send a proposal message to each cell in S_i
          proposing that j_i be completed at c'_i;
      wait for responses;
      if all replies are ok messages
        then stop; {new schedule is good}
      else
        for each job j_i for which
        a not-ok reply was received
          d_i := the value from the not-ok reply;
        reschedule B by smallest d_i
          using MDD to break ties;
    end; {while}
```

When a remote cell C_j receives a message proposing that c_i be completed at c'_i:

```
if c'_i ≤ t_ij
  C_j will return an ok message;
else
  C_j will return a not-ok reply
    along with value t_ij;
```

where:

P is the set of jobs in B that must precede some job at another cell;
S_i is the set of cells which have jobs which job j_i must precede;
d_i = the due time assigned to job j_i;
t_{ij} = the time that cell C_j expects j_i to have completed;
c_i = completion time for job j_i in the original schedule;
c'_i = completion time for job j_i in the latest new schedule;

In this algorithm, the disrupted cell uses communication to find out when other cells are expecting jobs at the disrupted cell to complete. The disrupted cell then assigns appropriate deadlines to the jobs to be rescheduled, and reschedules in order to meet those deadlines. This algorithm terminates when a deadline had been assigned to each job in P, regardless of whether the resulting schedule will disrupt another cell's schedule. Thus, this algorithm does not always find a solution which does not propagate disruptions, and there are many cases in which such a solution does not exist. If no solution is found, however, the information gained by the disrupted cell will help reduce the disruption caused to other cells.

3.3 An example

The very simple example in Figure 1 illustrates this problem. The original preschedule is shown in (a). This schedule is rendered infeasible by a disruption at machine A3 of cell A from time 0 to time 2. In (b), the schedule for cell A is pushed back from the disruption. This, however, causes part 6 to be delivered to cell B, disrupting cell B's schedule. By our simple negotiation algorithm, cell A proposes the pushed-back schedule to cell B, is informed by cell B that part 6

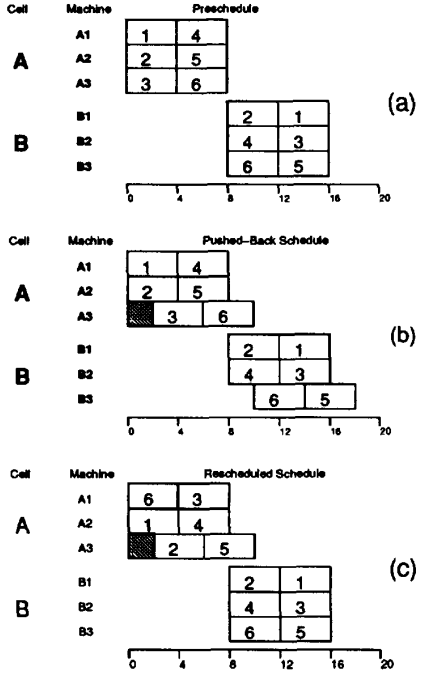

Figure 1: An example

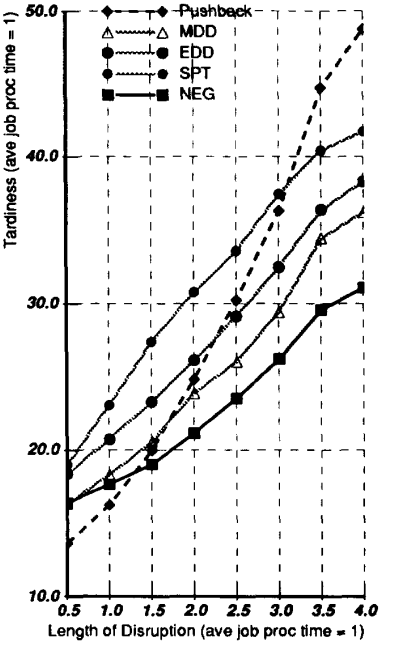

Figure 2: Tardiness vs disruption length

must complete processing at cell A by time 8. With this new information, cell A produces the schedule in (c), which does not disrupt cell B's schedule.

4 Experimental results and discussion

In order to evaluate our approach, we ran several experiments comparing the performance of our algorithm with that of commonly used dispatching rules, for several rescheduling problems. In our experiments, we assume a four-cell system, in which each cell has two identical machines, and twelve jobs are scheduled at each cell. Simple precedence constraints are provided by requiring that each of two jobs in each of the last three cells by preceded by one of the jobs of the first cell. Jobs have release times and due times. The performance measure is tardiness, and a preschedule for each job set was generated by a non-optimal branch and bound beam search, using MDD as a heuristic.

In each experiment, one of the machines of the first cell was disrupted for a certain period of time (during which the machine could not process any job). The first cell was then rescheduled, and any resulting schedule disruptions at other cells were handled by push-back. We compared the rescheduling performance of our negotiation-based algorithm (NEG) with that of push-back rescheduling, and rescheduling using three dispatch rules: MDD, EDD, and SPT. These methods were used for 20 different randomly generated job sets, and for 8 different disruption durations.

The results (averaged over the 20 job sets) are shown in Figure 2. For small disruptions, push-back rescheduling performed best, presumably because it retained most of the characteristics of the original preschedule. As disruptions become larger, our negotiation-based algorithm outperforms the others.

The reason that the negotiation-based algorithm performs better is illustrated in Figure 3. The best dispatch method, MDD, kept tardiness low on the disrupted cell, but suffered a high tardiness on the remote (undisrupted) cells. The negotiation-based algorithm, however, kept down tardiness at remote cells by trying to isolate the disruption at the disrupted cell. This was done at the cost of having a high tardiness at the disrupted cell. Additional tardiness at remote cells, however, is multiplied by the number of remote cells (in this case 3) in order to obtain total tardiness. This multiplicative effect demonstrates the cost of allow-

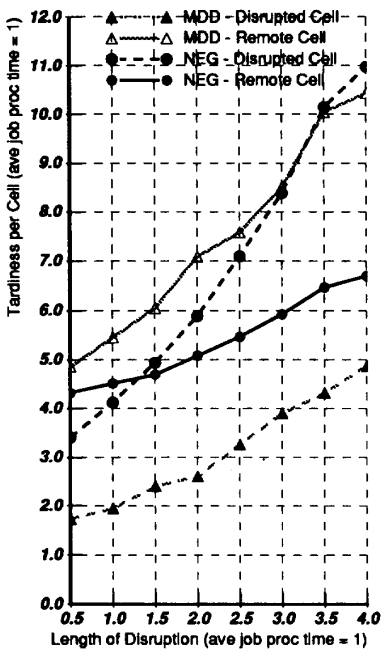

Figure 3: Cell tardiness vs disruption length

ing disruptions to propagate unhindered, and justifies incurring a larger local cost in order to limit these propagations.

These experiments also demonstrated limitations to the decentralized disruption recovery. Decentralized approaches generally work well for problems which are easily decomposable into almost independent subproblems. However, in problems constrained as these scheduling problems, constraints will probably make disruptions very hard to isolate. Even though our negotiation-based algorithm performed better than the other rescheduling methods did for large disruption size, it was still causing propagation of disruptions. Identifying how decomposable a problem is is the most important factor in how relevant these decentralized negotiation approaches are.

5 Conclusions

We have presented an approach to the problem of recovering from disruptions in a decentralized cellular manufacturing system. We are planning to continue this study for a system consisting of a larger number of cells, and a greater number of precedence constraints. This is just a part of the larger problem of providing intelligent coordination in a decentralized manufacturing system. Our future work will examine more rigorously the relevance and applicability of DAI techniques to this area.

References

[1] M. R. Adler et al. Conflict-resolution strategies for nonhierarchical distributed agents. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence Volume 2*, pages 139–162. Morgan Kaufmann, San Mateo, 1989.

[2] K. R. Baker and J. W. M. Bertrand. A dynamic priority rule for sequencing against due dates. *J. Opns. Mgmt.*, 3:37–42, 1982.

[3] J. C. Bean et al. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, May-June 1991.

[4] J. H. Blackstone et al. A state-of-the-art survey of dispatch rules for manufacturing job shop operations. *International Journal of Production Research*, 20(1):27–45, 1982.

[5] S. E. Conry et al. Multistage negotiation for distributed constraint satisfaction. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(6):1462–1477, November 1991.

[6] N. A. Duffie et al. Fault-tolerant heterarchical control of heterogeneous manufacturing system entities. *Journal of Manufacturing Systems*, 7(4):315–328, 1988.

[7] Y. Nishibe et al. Effects of heuristics in distributed constraint satisfaction problems. In *Proceedings Eleventh Internat'l Wrkshp on DAI*, pages 285–302, 1992.

[8] H. V. D. Parunak. Distributed artificial intelligence systems. In A. Kusiak, editor, *Artificial Intelligence Implications for Computer Integrated Manufacturing*, pages 225–251. IFS Ltd., 1988.

[9] S. F. Smith et al. An integrated framework for generating and revising factory schedules. *J. Opl. Res. Soc.*, 41(6):539–552, 1990.

[10] K. Sycara et al. Distributed constrained heuristic search. *IEEE Trans. on Systems, Man, and Cybernetics*, 21(6):1446–1461, November 1991.