DISTRIBUTING PRODUCTION CONTROL

K. Sycara, S. Roth, N. Sadeh, and M. Fox

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

The importance of distributed decision-making in factory environments arises from the fact that factories are inherently distributed, and from the need of effective local reactive capabilities to unexpected events on the factory floor. In this paper, we present an Artificial Intelligence (AI) approach to distributed factory floor control that (a) allows for flexible grouping of resources into work areas, and (b) efficiently generates good schedules for the whole factory. The approach relies on a *micro-opportunistic* scheduling technique where the granularity of decision making is the operation. In particular the scheduling method used relies on a combination of Constraint Satisfaction and Heuristic Search. To guide and focus opportunistic scheduling, we have developed a set of scheduling heuristics that enable a scheduler to select the next operation to schedule and the next resource reservation to make for the selected operation. These heuristics have been experimentally tested in a single scheduler setting and have been found to produce good schedules efficiently. We have extended the use of these heuristics in a distributed setting and describe preliminary results of experiments.

1.0 INTRODUCTION

In this paper, we present AI mechanisms to distribute the scheduling task in the factory environment. It is common to divide a factory into departments, e.g., tooling, nc programming, and materials, where each department is responsible for the planning and scheduling of its own operations. For example, the factory floor scheduler is responsible for sequencing and scheduling of operations required to produce an order. A scheduler's focus is on the allocation of machines based upon available capacity. In order for operations to be performed, the scheduler must request resources from other departments, e.g., tools and materials, which fall outside of its purview. The tooling department can be viewed as a mini-factory; as tools are worn out, or their designs revised, it is necessary to alter them. The tooling department has its own machining facilities which it must schedule in order to meet the tooling needs of the factory. Consequently, the synchronization of operations is required if tools are to be available on time.

A similar problem exists at the plant level. Each manufacturing plant produces for or consumes resources from the plants which logically follow or precede it respectively. In order for the plants to operate smoothly, resource requirements must be negotiated and their operations synchronized. Although the plants are supposed to be cooperative, they tend not to be. Because the level of cooperation between

[1]This research has been supported, in part, by the Defense Advance Research Projects Agency under contract #F30602-88-C-0001, and in part by grants from McDonnell Aircraft Company and Digital Equipment Corporation.

Appeared in The Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, May 90

the plants is not fixed, plants need to explicitly reason about their goals and utilities while interacting with others.

Factory scheduling has been the subject of intense investigation by both Operations Research and AI communities (e.g., [12, 7, 5]). With few exceptions [9, 13], there has been almost no research in distributed scheduling. On the other hand, the Distributed AI community has focused its attention on problems where agents contend only for computational resources, such as computer time and communication bandwidth (e.g., [1, 2]). In most real world situations, however, allocation of (non-computational) resources that are needed by a planner to carry out actions in a plan is of central concern. Hence, approaches and mechanisms are needed to allow for cooperative distributed resource allocation over time (i.e., distributed scheduling of resources). The CORTES project has undertaken the construction of a decentralized heterogeneous multiagent production control system in order to study the impact of different production control architectures. Each agent in the CORTES project consists of the following modules: (a) detailed scheduler, (b) a set of dispatchers, (c) uncertainty analyzer, (d) factory floor model, and (e) individual machines. These modules perform a set of tasks encompassing generation of a detailed schedule for incoming orders, dispatching of operations to machines, monitoring operation execution, and taking into consideration the uncertainty in the factory environment (e.g., machine failures). In addition, the factory floor model is dynamically updated to reflect the current state of affairs. In this paper, we concentrate on the interactions among detailed schedulers. Future work will integrate the uncertainty analysis and execution monitoring capabilities into the distributed environment.

Distributed factory scheduling is a process carried out by a group of schedulers each of which knows only the orders and resource availability in the work area under its responsibility. Since some of the resources may need to be shared among different schedulers, the schedulers must interleave local scheduling computations with information exchange with other schedulers. Distributed schedules are constructed in an incremental fashion by cooperation between the schedulers that are responsible for scheduling particular sets of resources. The system goal is to find schedules that are not simply feasible but attempt to optimize some global objective function (e.g., minimize order tardiness, minimize work in process). The global objective function to be optimized reflects the quality of schedule that is produced. Another, equally important concern, is the efficiency of schedule generation. Since backtracking is a fact of life in scheduling, there is a need for approaches that incrementally produce schedules while minimizing backtracking.

We have been able to efficiently construct good schedules in a centralized setting by an approach that uses a combination of Constraint Satisfaction and Heuristic Search. In contrast to macro-opportunistic constraint-directed scheduling [8], where the focus of the scheduler opportunistically switches between a resource perspective and an order perspective, our approach is micro-opportunistic since (a) the unit of decision, the operation, is finer-grained, and (b) both a resource and order perspective are considered in conjunction at every scheduling decision point. To guide the scheduling process we have identified a set of domain-independent heuristics. They enable a scheduler to select the next operation to schedule and also the next resource/time interval assignment to be made for the selected operation. In other words, the heuristics enable the scheduler to do *ordering of operations* and *ordering of reservations*. They are respectively called variable and value ordering heuristics. These heuristics have been developed for a centralized scheduler after extensive experimentation to ascertain their goodness, both in terms of backtracking minimization and quality of schedules produced [11]. Our hypothesis is that these heuristics are good predictive measures of the impact of local scheduling decisions on the overall goals for the factory and express expectations of the resource needs of agents.

The distributed factory scheduling problem has a set of characteristics that impose requirements on a distributed scheduling system. These characteristics include:

- The global goal is to schedule in a distributed fashion a set of operations to optimize a set of production criteria.
- To achieve global solutions, schedulers have to make consistent allocations of resources that are needed to perform system activities. Because resources have limited capacity, conflicts in the system arise in the form of contention over optimal allocation of shared resources.
- Due to limited communication bandwidth and organizational reluctance, it is not possible to

exchange a complete set of specific constraints.

- Because of the conflicts over the shared resources, it is generally impossible for any scheduler to optimize the scheduling of its assigned orders without exchanging information with other schedulers.
- Certain constraints (e.g., precedence constraints among the operations of an order) cannot be relaxed.
- Local computations could produce infeasible schedules. When this happens the agent has to backtrack and try again. Backtracking can have major ripple effects on the scheduling system since it may invalidate resource reservations that other schedulers have made.

A consequence of the above characteristic is that schedulers, at each stage of scheduling, need mechanisms to (1) predict and evaluate the impact of scheduling decisions on global system goals, (2) form expectations and predictions about the resource needs and behavior of other schedulers, and (3) help focus the attention of individual schedulers to globally critical decision points, and (4) provide guidance in making a particular scheduling decision at a decision point. Our hypothesis is that the variable and value ordering heuristics that are presented in this paper have these desired properties. To test our hypothesis we have completed the implementation of a distributed testbed and are currently performing experiments involving multiple schedulers.

The rest of the paper is organized as follows: Section 2 presents the micro-opportunistic approach to scheduling and the variable and value ordering heuristics in the context of a single scheduler. Section 3 presents the extension of these heuristics to a distributed setting and their use in the communication protocol employed in our system. Section 4 presents an example of distributed scheduling. Section 5 presents experimental results and section 6 concluding remarks.

2.0 SCHEDULING BY A SINGLE AGENT

Each scheduling agent is responsible for scheduling a set of orders. The task of the schedulers is to cooperatively optimize over the whole factory a set of organizational goals [3], such as reducing order tardiness (i.e. meeting due dates), reducing order earliness (i.e. finished good inventory), reducing order flowtime (i.e. in-process inventory), using accurate machines, performing some activities during some shifts rather than others, etc. These organizational goals are represented as sums of utilities that express the desired tradeoffs in the optimization function [10]. Since each scheduler knows only the operations that it must schedule and their associated utilities, the schedulers have to exchange useful information concerning their scheduling decisions and utilities.

Each order consists of a set of operations that must be scheduled according to a given process plan. Additionally an order has a release date and a latest acceptable completion date, which may actually be later than the ideal due date. In order to accomplish an operation, a set of resources (e.g., a machine, fixtures, a human operator) are needed. There may be resource alternatives, more or less preferable, that can be used to perform an operation. There is limited resource capacity in each factory. Some resources are only required by one scheduler, and are said to be *local* to that scheduler. Other resources, such as tools and fixtures, are *shared*, in the sense that they may be allocated to different schedulers at different times. Each scheduler operates under a set of constraints. The most important constraint types are precedence constraints and capacity constraints. Precedence constraints refer to the temporal relations among operations and they are specified in the process plan. Precedence constraints along with order release dates and latest acceptable completion dates restrict the sequencing of operations and their allowable start times. Capacity constraints refer to available resource capacity and restrict the number of operations that can be simultaneously performed on a resource. For the purposes of this paper, we assume unary capacity for each resource.

Each agent builds a schedule for the orders it must schedule in a *micro-opportunistic* fashion. The process of micro-opportunistic scheduling is a combination of Constraint Satisfaction and Heuristic Search [4] and consists of iteratively selecting an operation to be scheduled and a reservation for the

selected operation. We call this process micro-opportunistic since (a) the unit at which decisions are made is the operation, and (b) the resource perspective and order perspective [8] are considered simultaneously at the finer granularity level of operation scheduling. Each time a new operation is scheduled, new constraints are added to the scheduler's initial scheduling constraints that reflect the new reservation. These new constraints are then propagated (consistency checking). If an inconsistency (i.e. constraint violation) is detected during propagation, the system backtracks. Otherwise the scheduler moves on and looks for a new operation to schedule and a reservation for that operation. The process terminates when all operations have been successfully scheduled.

If a scheduler could always make sure that the reservation that it is going to assign to an activity will not result in some constraint violation forcing it or other schedulers to undo some earlier decisions, scheduling could be performed without backtracking. Because scheduling is NP-hard, it is commonly believed that such look-ahead cannot be performed cheaply. The most efficient constraint propagation techniques developed so far for scheduling [6] guarantee consistency with respect to precedence constraints but not with respect to capacity constraints. In other words, there are situations with insufficient capacity that may go undetected by this constraint propagation technique. As a result, a reservation made by a scheduler may force other schedulers or the scheduler itself to backtrack later on. Hence, it is important to construct schedules in a way that reduces the chances of having to backtrack and minimizes the work to be undone when backtracking occurs. This is accomplished via two techniques, known as *variable* (i.e. operation) and *value* (i.e. reservation) ordering heuristics.

2.1 Ordering of Operations to Schedule

The variable ordering heuristic assigns a *criticality measure* to each unscheduled operation. The heuristic consists of scheduling first the operation with the highest criticality. An operation is critical if its resource requirements are likely to conflict with the resource requirements of other operations. The criticality measure approximates the likelihood that the operation will be involved in a conflict. By scheduling its most critical activity first, a scheduler reduces its chances of wasting time building partial schedules that cannot be completed (i.e. it will reduce both the frequency and the damage of backtracking). In order to identify critical operations, we use a method described in [10, 11]. A scheduler starts by building for each unscheduled operation and for each resource that the operation needs a probabilistic *operation demand.* This demand expresses the probability that the operation will use the resource at some time t. Operations that contend for a resource over a narrow time window, have higher operation demands on that resource. In a second step, each scheduler aggregates its operation demands as a function of time, thereby obtaining its *scheduler demand.* This demand reflects the need of the scheduler for a resource as a function of time. The individual scheduler demands are aggregated for the whole system thereby producing *aggregate* demands that indicate the degree of contention among schedulers for each of the (shared) resources in the system, as a function of time. Time intervals over which a resource's aggregate demand is very high correspond to violations of capacity constraints that are likely to go undetected by the constraint propagation mechanism. The contribution of an operation's demand to the aggregate demand for a resource over a highly contended-for time interval is the criticality measure for that operation.

To choose the next operation to schedule, each scheduler looks for the resource/time interval over which it has some demand that is the most likely to be involved in a capacity constraint violation. It then picks its activity with the highest probability of being involved in the conflict.

2.2 Ordering of Reservations for an Operation

Once a scheduler has selected the operation to schedule next, it must decide which reservation to assign to that operation. The value ordering heuristic attempts to leave enough options open to the operations that have not yet been scheduled in order to reduce the chances of backtracking. This is done by assigning a *goodness* measure to each possible reservation of the operation to be scheduled. We have identified a number of strategies that can be employed to operationalize the value ordering heuristic. In

particular, we distinguish between two extreme strategies:

1. **A Least Constraining Value Ordering Strategy (LCV):** One type of value ordering heuristic is a least constraining one. Schedulers using such heuristic attempt to select the reservation that is the least likely to prevent other operations to be scheduled. In other words a scheduler will select the reservation that will be the least constraining both to itself and to other schedulers. This heuristic results in *altruistic* behavior on the part of the scheduler.

2. **A "Greedy" Value Ordering Strategy (GV):** At the other extreme, a scheduler can select reservations based solely on its local preferences, i.e. irrespectively of its own future needs as well as those of other schedulers. This heuristic results in *egotistic/myopic* behavior on the part of the scheduler.

In between these two extremes, there is a continuum of strategies that combine features of both LCV and GV. These intermediate strategies attempt to factor in the contribution of a reservation to the global objective function together with the likelihood that selecting a reservation will result in backtracking (either locally or for another scheduler). Experiments in centralized scheduling [11] indicate that LCV-type heuristics are best at minimizing search, but usually result in poor schedules since reservations are selected irrespectively of their contribution to the objective function. Value ordering heuristics of the greedy type usually produce significantly better schedules, but they result in extra backtracking (i.e. search takes longer). The amount of extra backtracking is however significantly reduced when the GV value ordering heuristic is combined with the variable ordering heuristic described in the previous paragraph. Because schedulers in the decentralized case schedule in an asynchronous fashion, we expect the effect of the variable ordering heuristic to be weaker. Additionally the cost of backtracking in a distributed system is known to be higher than in a centralized one due to the overhead in coordinating schedulers whose earlier decisions are interdependent. Accordingly, we expect a higher need for least constraining behavior in a distributed scheduling environment. The ultimate choice of an (intermediate) strategy is likely to depend on such factors as the time available to come up with a solution, the load of the schedulers, and the amount of resource contention.

3.0 DISTRIBUTED SCHEDULING

The distributed CORTES system consists of several schedulers, each of which is given a set of orders to schedule. The orders may require resources that are also needed by other schedulers. Each resource in the system is monitored by a scheduler, and conversely, each scheduler is a monitor for some set of resources. The concept of monitoring schedulers is introduced in order to facilitate detection of capacity conflicts. Each resource monitor keeps track of the reservations that have been made for the resource it monitors. Capacity constraints are detected by the monitor when a scheduler requests a reservation for the monitored resource and for a time interval that has already been reserved. In addition, monitoring schedulers achieve more efficient inter-scheduler communication to inform schedulers of new reservations and changes in expected resource needs. They may be viewed as communication hubs for collecting and dispersing information.

We have identified two levels of interaction of the schedulers: the strategic level where aggregate information is communicated and the tactical level where information about specific entities is communicated. The information communicated at the strategic level is the demand profiles, with which schedulers calculate criticality measures for their decision making. At the tactical level, particular scheduling decisions are made and, if needed, negotiation takes place.

Because they may contend for the same resources, it is important that schedulers build their schedules in a cooperative manner. The two texture measures identified in the previous section provide a framework for cooperation where the schedulers exchange demand profiles and reservations. Demand profiles are aggregated periodically to compute textures that allow schedulers to form expectations about the resource demands of other schedulers. Because of communication overhead, the demand profile information is restricted. Subsets of the schedulers communicate only demand profiles for the resources

that they share, although reservations on the non-shared resources may impact scheduling decisions on the shared ones. Since several schedulers are scheduling asynchronously, and the communicated demand profiles are only those of the subset of shared resources, there is higher uncertainty in the system. This uncertainty also varies in an inversely proportional manner with the frequency at which the demand profiles are communicated. Moreover, the cost of backtracking is greater, since if a scheduler backtracks, the change in scheduling reservations may ripple through to the other schedulers and cause them to change their reservations.

In particular, the multi-scheduler communication protocol is as follows:

I. Each scheduler determines required resources by checking the process plans for the orders it has to schedule. It sends a message to each monitoring scheduler for needed shared resources informing it of the need[2].

II. Each scheduler calculates its demand profile for the resources (local and shared) that it needs.

III. Each scheduler sends its demand profiles (*scheduler demands*) for the shared resources it needs to the scheduler(s) that monitor those shared resources.

IV. The monitoring scheduler aggregates the scheduler demands it receives to form the *aggregate demand* for a resource and sends it to all the schedulers that share the resource.

V. Upon receipt of the aggregate demand for each of the shared resources that it needs, a scheduler finds its most critical resource/time-interval pair and its most critical operation (the one with the greatest demand on this resource for this time interval). Since schedulers in general need to use a resource for different time intervals, the most critical operation and time interval for a resource will in general be different for different schedulers. The scheduler calculates a desired reservation for its most critical operation using either LCV (the least constraining time interval heuristic), or GV (the egotistic/myopic heuristic) depending on whether it wants to minimize conflicting with other schedulers for resource reservations, thus minimizing backtracking, or whether it wants to increase the quality of its schedule. The scheduler communicates this reservation request to the resource's monitoring scheduler.

VI. The monitoring scheduler upon receiving these reservation requests checks the resource calendar for the event of conflicting reservations (i.e. a scheduler requests a reservation on a resource for an already reserved time interval). There are two cases:

1. If there are no conflicts, the monitoring scheduler (a) communicates "Reservation OK" to the requesting scheduler, (b) marks the reservation on the resource calendar, and (c) communicates the reservation to all concerned schedulers (i.e. the schedulers that had sent positive demands on the resource).
2. If there is a conflict, it communicates the conflicting resource/time interval to the conflicting schedulers.

VII. Upon receipt of the information about the conflicting reservation, the conflicting schedulers have to decide how to resolve the conflict. There are two ways:

1. The scheduler that has secured the reservation first does not relinquish it. The second scheduler has to calculate an alternative, if there is one left, or backtrack otherwise.
2. The schedulers negotiate a new mutually satisfactory assignment of time intervals to the competing operations. These new reservations are communicated to the monitoring

[2]This is done so that the monitoring scheduler will know which schedulers will be sending demand profiles for a resource, so it can ascertain the completeness of the information before aggregation.

scheduler who undoes the previous reservation and installs the new ones.

The system terminates when all operations of all schedulers have been scheduled i.e. when all demands on resources become zero. In this version of the protocol we assume that reservations are not changed in one scheduler because of the need for backtracking in another. This assumption has two consequences: 1) Once a scheduler has been granted a reservation, this reservation is not automatically undone because some other scheduler had to backtrack and may now need the reservation. In such situations, the two schedulers have to negotiate to decide whether the reservation will get undone. 2) If a scheduler backtracks after it finds out that a schedule that it was constructing is infeasible (i.e., it cannot satisfy the problem constraints), it frees up resources but the reservation of other schedulers on these resources remain as they were. This policy may result in non-optimal reservation for other schedulers since it denies the other schedulers the opportunity to take advantage of the canceled reservations of the backtracking scheduler, but it results in less computationally intensive performance.

4.0 AN EXAMPLE

We present an example that illustrates the use of the heuristics to predict resource utilization and make reservations in a distributed environment. For simplicity, the example assumes a two-scheduler system with a single shared resource, R1. We further assume that both schedulers have calculated demand profiles for R1 and have communicated them to R1's monitor-scheduler. The remaining steps involve (1) calculation of the aggregate demand profile for R1, (2) determination of the most critical resource interval from the aggregate-demand profile, (3) determination of the most critical operation for each scheduler within this interval, and (4) application of heuristics by each scheduler to select a reservation for its critical operation.

The curves in Figure 1 show the aggregate-demand profile (1a), scheduler demand (1b) and operation profiles (1c, 1e). The vertical axis in each picture expresses demand and the horizontal axes intervals of time. Figure 1a illustrates the aggregate demand-density which is the simple sum of scheduler-demands at each time point for schedulers α, β. Recall that each scheduler demand is calculated from the operation demands for all its operations that use R1. Schedulers α and β detect that the most critical resource/time-interval for R1 occurs at interval [30, 40] (the peak of the aggregate-demand = 1.3). Had there been more resources, the aggregates for these would be calculated in a similar manner and the resource with an interval which has the most demand would have been determined.

Having selected the most critical resource/time-interval, schedulers α and β use this to select an operation to schedule. In this case, the schedulers are focusing on the same critical resource interval. Typically, schedulers oftens focus on different resources because they proceed asynchronously and have different resource requirements. Figures 1c and 1e show the operation demands for operations which have non-zero demand in the critical resource/time-interval. In this step, referred to as variable selection, each scheduler determines the operation with the greatest demand **within the critical resource/time-interval** [30, 40]. This is operation A_1^{α} for scheduler α and A_3^{β} for scheduler β (scheduler β has only one operation with non-zero demand in this interval). Note that the only interval considered at this point is [30, 40], so other operations can have much higher demand in other intervals, but are not selected.

Finally, each scheduler chooses a reservation for its selected operation (i.e. performs value-selection). Figures 1d and 1f illustrate how comparison of an operation's demand and the aggregate demand predict the degree of conflict that can be expected for each time interval. For example, at interval [30, 40] in figure 1f, the demand difference between the aggregate demand and scheduler β's operation A_3^{β} is 0.7 (1.3 - .6), indicating that a reservation at this interval is likely to result in conflict with another operation. In contrast, for interval [50, 60], the demand difference is only 0.3 (.5 - .2). In general, the interval for which a reservation is least likely to conflict with other operations is the minimum of the difference between the aggregate demand and the operation demand (for all intervals where the operation's demand is non-zero). The LCV heuristic, which selects the least constraining value, will select interval [50, 60] for operation A_3^{β}. Similarly, scheduler α will select interval [10, 20] (the

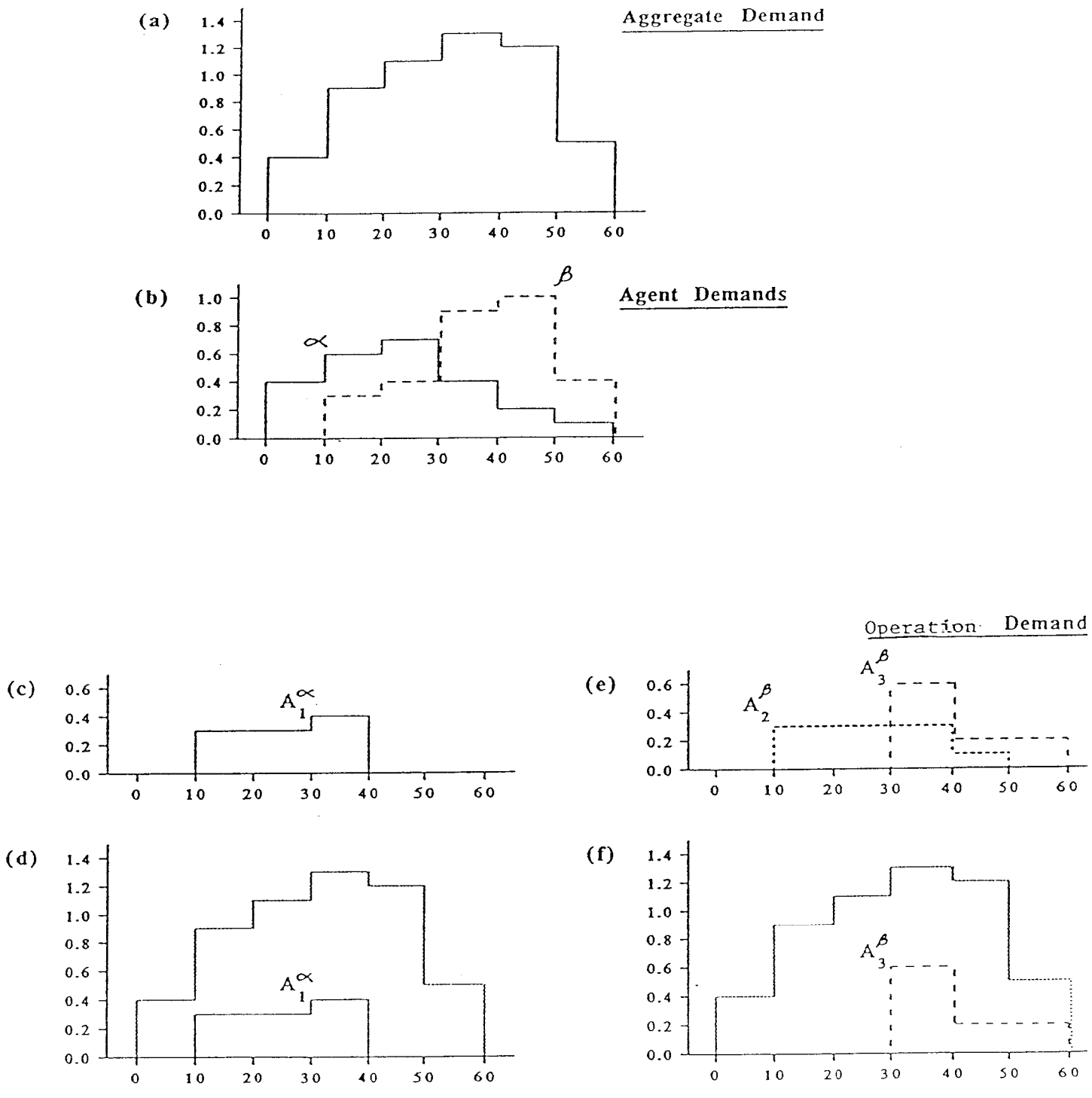

Figure 1: Demand for a single resource as a function of time. These curves are used for variable and value selection.

schedulers have selected non-overlapping time intervals).

5.0 EXPERIMENTAL RESULTS AND CONCLUDING REMARKS

In this paper we have presented a methodology for distributed factory scheduling. This methodology is based on an integration of constraint satisfaction and heuristic search. A set of domain-independent heuristics that have been identified and tested in a single scheduler setting have been extended to guide distributed decision-making. We have presented the use of these heuristics in a communication protocol that allows schedulers to coordinate their scheduling actions. A testbed has been implemented that allows for experimentation with a variety of distributed protocols that use variable and value ordering heuristics. The experiments are designed to give results concerning the role of the heuristics in the tradeoff of solution quality versus amount of search, and the influence of frequency of communication on schedule quality and system performance. The testbed is implemented in Knowledge Craft running on top of Common Lisp, and can be run either on a MICROVAX 3200 or on a VAX 8800 under VMS.

Our initial experiments were designed to determine whether schedules could be generated over a range of parameters which reflect the degree of uncertainty in the system. Uncertainty varied as a function of the frequency with which resource demands were communicated and as a function of the degree of resource sharing across schedulers (i.e. the number of shared resources relative to local resources). The more schedulers who use a resource, the more difficult it is for any one scheduler to be certain about the need for each resource. Finally, uncertainty also varied as a function of the amount of information that was exchanged about schedulers' resource demands. As expected, the ease of finding a solution was proportional to uncertainty, although solutions could be found over a range of problem difficulty.

Our preliminary results do point out the need for a negotiation mechanism to solve a subset of scheduling problems which are solvable by a centralized scheduler, but not always in a distributed environment. These cases arise occasionally because one scheduler has been successful in scheduling all its operations, but in doing so, has made reservations which make it impossible for another agent to achieve a complete schedule. Recall that in the communication protocol described in section 3, a scheduler did not automatically backtrack if another scheduler was unable to arrive at a feasible solution. This was necessary to avoid creating a tremendously large search space. Our testbed provides an opportunity to study the use of negotiation to initiate and control multi-scheduler backtracking. This involves experimenting with criteria for determining (1) when negotiation should occur, (2) the particular resources and reservations on which negotiation should focus, (3) the schedulers that should be involved, (4) the negotiation protocols needed to achieve a resolution, and (5) negotiation strategies with which schedulers can balance their own local goals with those of other schedulers.

REFERENCES

1. Cammarata, S. McArthur, D. and Steeb, R. Strategies of cooperation in distributed problem solving. IJCAI-83, IJCAI, Karlsruhe, W. Germany, 1983, pp. 767-770.

2. Durfee, E.H. *A Unified Approach to Dynamic Coordination: Plannign Actions and Interactions in a Distributed Problem Solving Network.* Ph.D. Th., COINS, University of Massachusetts, Amherst, MA., 1987.

3. Fox, M.S. *Constraint-Directed Search: A Case Study of Job Shop Scheduling.* Ph.D. Th., Computer Science Department, Carnegie-Mellon University, 1983.

4. Mark S. Fox, Norman Sadeh, and Can Baykan. Constrained Heuristic Search. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989, pp. 309-315.

5. Karmarker, U.S. Alternatives for Batch Manufacturing Control. Tech. Rept. QM 86-13, University of Rochester, June, 1986.

6. LePape, C. and S.F. Smith. Management of Temporal Constraints for Factory Scheduling. Proceedings IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems (TAIS 87), held in Sophia Antipolis, France, May, 1987.

7. Ow, P.S., S.F. Smith, and A. Thiriez. Reactive Plan Revision. Proceedings AAAI-88, St. Paul, Minnesota, August, 1988.

8. Ow, P.S. and S.F. Smith. "Two Design Principles for Knowledge-Based Systems". *Decision Sciences 18*, 3 (Summer 1987).

9. Parunak, H.V., P.W. Lozo, R. Judd, B.W. Irish. A Distributed Heuristic Strategy for Material Transportation. Proceedings 1986 Conference on Intelligent Systems and Machines, Rochester, Michigan, 1986.

10. N. Sadeh and M.S. Fox. Preference Propagation in Temporal/Capacity Constraint Graphs. Tech. Rept. CMU-CS-88-193, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1988. Also appears as Robotics Institute technical report CMU-RI-TR-89-2.

11. N. Sadeh and M.S. Fox. Focus of Attention in an Activity-based Scheduler. Proceedings of the NASA Conference on Space Telerobotics, 1989.

12. Stephen F. Smith and Peng Si Ow. The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks. Proceedings of the Ninth International Conference on Artificial Intelligence, 1985, pp. 1013-1015.

13. Smith, S.F. and J.E. Hynynen. Integrated Decentralization of Production Management: An Approach for Factory Scheduling. Proceedings ASME Annual Winter Conference: Symposium on Integrated and Intelligent Manufacturing, Boston, MA, December, 1987.