An Investigation into Distributed Constraint-directed Factory Scheduling

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

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

[1]We present an approach to focus search in a distributed system in individual agent search spaces so as to optimize decisions in the global space. The chosen domain is distributed factory scheduling. The importance of distributed decision-making in factory environments arises from the fact that factories are inherently distributed, and from the need of effective responsiveness to change. The approach relies on a set of *texture measures* that quantify several characteristics of the space being searched. These texture measures play four important roles in distributed search: (1) they focus the attention of an agent to globally critical decision points in its local search space, (2) they provide guidance in making a particular decision at a decision point, (3) they are good predictive measures of the impact of local decisions on system goals, and (4) they are used to model beliefs and intentions of other agents. The development of the presented texture measures is the result of extensive experimentation in a single agent setting. We have completed the implementation of a distributed testbed and are currently performing experiments involving multiple agents.

AI TOPIC: Reasoning, Search
DOMAIN AREA: Factory Scheduling
LANGUAGE TOOL: Common Lisp and KnowledgeCraft
STATUS: Three versions of the single agent system have been completed and extensive experimentation performed as to schedule quality and system performance. The distributed testbed has been completed. Communication and negotiation protocols are being researched. Experimentation with the distributed multi-agent system is currently under way.
EFFORT: Single-agent system: 2 person years. Multi-agent system: 1.5 person years.
IMPACT: This research is of both theoretical and practical interest. Providing algorithms and protocols that enable efficient distributed search is of major importance in many domains. In the practical arena, realizing the goal of efficient distributed scheduling will enable factories to increase performance and robustness in the face of the unpredictability of the factory environment.

1. Introduction

In this paper we present mechanisms that enable a set of agents to focus and direct search in their individual search spaces so as to optimize decisions in the global search space. Our investigation is conducted in the domain of distributed scheduling and in particular in distributed factory scheduling. Factory scheduling has been the subject of intense investigation by both Operations Research and AI communities (e.g., [1, 2, 3]). With few exceptions [4, 5], 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., [6, 7]). 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. A first version of the system has been implemented that allows for both planning and scheduling agents. Planning agents start with a set of orders to be produced. Each planning agent generates a process plan for each of its orders, and sends the plan to a scheduling agent along with a release date, a due date, and a set of preferences to be optimized. The scheduling agents have to attempt to build a good schedule for each of the orders that they receive. The architecture allows for two types of agent interactions: interactions between scheduling agents competing for shared resources, and interactions between planning and scheduling agents. In particular a scheduling agent may request an alternative process plan from a planning agent if it finds it too difficult to schedule the one that it currently has. In this paper, we concentrate on the interactions among the schedulers.

In investigating decentralized multi-agent scheduling, we are motivated both by pragmatic and theoretical considerations. Scheduling takes place at all levels of a manufacturing organization (from scheduling the factory floor to scheduling meetings of the board of directors). Manufacturing organizations are naturally distributed so there is need to schedule the parts of an organization in such a way as to optimize the performance of the whole. Typically a factory is divided into work areas that might share resources needed (e.g., machines, fixtures, operators) to schedule orders. The inherently decentralized nature of the activities that are being coordinated requires a corresponding decentralization of planning and control mechanisms. From the

[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.

To appear in proceedings "The Sixth Conference on Artificial Intelligence Applicatio
Fess Parker's Red Lion Resort, Santa Barbara, California. March 5-9, 1990

theoretical point of view, we are interested in investigating coordination techniques within our framework (1) as the available information degrades (becomes incomplete) (2) as the need for reactivity on the part of each agent grows due to possibly conflicting scheduling decisions of other agents.

Distributed scheduling is a process carried out by a group of agents each of which has (a) limited knowledge of the environment, (b) limited knowledge of the constraints and intentions of other agents, and (c) limited number and amount of resources that are required to produce a system solution. Some of these resources may be shared among many agents. Global system solutions are arrived at by interleaving of local computations and information exchange among the agents. There is no single agent with a global system view. In such an environment, schedules are constructed in an *incremental fashion*. Agents make local decisions about assignments of resources to particular activities at particular time intervals and a complete schedule for an order is formed by incrementally merging partial schedules for the order. 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. Our approach, based on so-called *texture measures* endeavors to take into consideration both optimization components (quality of schedule and efficiency of schedule generation).

The distributed scheduling problem has the following characteristics:

- The global system goal is to schedule in a distributed fashion a set of activities (operations) to optimize a set of global criteria and minimize backtracking.
- To achieve global solutions, agents 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, it is not possible to exchange a complete set of specific constraints.
- Because of the conflicts over the shared resources it is in general impossible for each agent to optimize the scheduling of its assigned orders using only local information.
- Because the scheduling problem is NP hard it is impossible for each agent to have precomputed a set of local schedules that it then could exchange and try to make compatible with the schedules of other agents.
- All the given orders have to be scheduled. In other words, agents cannot drop any local goals.
- 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 network since since it may invalidate resource reservations that other agents have made.

A consequence of the above characteristic is that agents, 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 agents, and (3) help focus the attention of the agents on particular parts of their search space opportunistically. Having good predictive measures is very important in the distributed case because:

1. backtracking is more costly when it involves many agents (ripple effects)
2. agents may be called upon to estimate their resource needs for resources that they may not need to consider until they have made many other reservations
3. since the agents operate asynchronously, they have to predict and take into consideration in their local decision-making the *future* needs of other agents that are in an earlier stage of scheduling
4. since communication is costly the predictive measures must be robust/predictive over many scheduling decisions

Our initial work [8] has focused on the formalization of the concept of constraint-directed search within a centralized framework. In particular we have identified a set of texture measures that quantify several characteristics of the problem space within which search is being performed. Our hypothesis is that these textures are good predictive measures of the impact of local decisions on system goals and express expectations of the resource needs of agents. We have operationalized these textures into a set of heuristics that direct the search of the scheduling agents. These heuristics have been successfully applied to direct the search of a centralized activity-based scheduler [9], so as to achieve good schedule quality and minimize the likelihood of backtracking. In this paper we discuss how these textures can be used to direct distributed scheduling.

2. The Model

Formally, we will say that we have a set of scheduling agents, $\Gamma=\{\alpha, \beta, ...\}$. Each agent α is responsible for the scheduling of a set of orders $O^{\alpha}=\{o_1^{\alpha},...,o_{N_\alpha}^{\alpha}\}$. Each order o_l^{α} consists of a set of activities $A^{l\alpha}=\{A_1^{l\alpha},...,A_{n_{l\alpha}}^{l\alpha}\}$ to be scheduled according to a process plan (i.e. process routing) which specifies a partial ordering among these activities (e.g. $A_p^{l\alpha}$ BEFORE $A_q^{l\alpha}$). Additionally an order has a release date and a latest acceptable completion date, which may actually be later than the ideal due date. Each activity $A_k^{l\alpha}$ also requires one or several resources $R_{ki}^{l\alpha}$ $(1 \le i \le p_k^{l\alpha})$, for each of which there may be one or several alternatives (i.e. substitutable resources) $R_{kij}^{l\alpha}$ $(1 \le j \le q_{ki}^{l\alpha})$. There is a finite number of resources available in the system. Some resources are only required by one agent, and are said to be *local* to that agent. Other resources are *shared*, in the sense that they may be allocated to different agents at different times.

We distinguish between two types of constraints: activity precedence constraints and capacity constraints. The activity precedence constraints together with the order release dates and latest acceptable completion dates restrict the set of acceptable start times of each activity. The capacity constraints restrict the number of activities that a resource can be allocated to at any moment in time to the capacity of that resource. For the sake of simplicity, we only consider resources with unary capacity in this paper. Typically the limited capacity of the resources induces interactions between orders competing for the possession of the

same resource at the same time.

With each activity, we associate utility functions that map each possible start time and each possible resource alternatives onto a utility value (i.e. preference). These utilities [10, 11] arise from global organizational goals 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. In the cooperative setting assumed in this paper, the sum of these preferences over all the agents in the system and over all the activities to be scheduled by each of these agents defines a common objective function to be optimized. The sum of these preferences over all the activities under the responsibility of a single agent can be seen as the agent's local view of the global objective function. In other words, the global objective function is not known by any single agent. Furthermore, because they compete for a set of shared resources, it is not sufficient for an agent to try to optimize his own local preferences. Instead, agents need to consider the preferences of other agents when they schedule their activities. This is accomplished via a communication protocol, which we describe in section 3.

2.1. Activity-based Scheduling Agents

In our model we view each activity $A_k^{l\alpha}$ as a *variable*. A variable's *value* corresponds to a reservation for an activity. A reservation consists of a start time and the set of resources needed by the activity (i.e. one resource $R_{kij}^{l\alpha}$ for each resource requirement $R_{ki}^{l\alpha}$, $1 \le i \le p_k^{l\alpha}$, of $A_k^{l\alpha}$).

Each agent asynchronously builds a schedule for the orders he must schedule. This is done incrementally by iteratively selecting an activity to be scheduled and a reservation for that activity. Each time a new activity is scheduled, new constraints are added to the agent's initial scheduling constraints that reflect the new activity 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 activity to schedule and a reservation for that activity. The process goes on until all activities have been successfully scheduled.

If an agent could always make sure that the reservation that he is going to assign to an activity will not result in some constraint violation forcing him or other agents 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 [12] for scheduling do not guarantee total consistency. In other words the reservation assigned by an agent to an activity may force other agents or the agent himself to backtrack later on[2]. Consequently it is important to focus search 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. activity) and *value* (i.e. reservation) ordering heuristics.

[2]This is already the case in the centralized version of the scheduling problem. Because of the additional cost of communication it is even more so in the distributed case.

The variable ordering heuristic assigns a *criticality measure* to each unscheduled activity; *the activity with the highest criticality is scheduled first.* The criticality measure approximates the likelihood that the activity will be involved in a conflict. The only conflicts that are accounted for in this measure are the ones that cannot be prevented by the constraint propagation mechanism. By scheduling his most critical activity first, an agent reduces his chances of wasting time building partial schedules that cannot be completed (i.e. it will reduce both the frequency and the damage of backtracking). The value ordering heuristic attempts to leave enough options open to the activities 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 activity to be scheduled. Both activity criticality and value goodness are examples of *texture measures*. The next two paragraphs briefly describe both of these measures[3]. A protocol to exchange the information required to compute these measures is then presented in the next section.

2.1.1. Activity Ordering

Each agent's constraint propagation mechanism is based on the technique described in [12]. It always ensures that unscheduled activities within an order can be scheduled without violating activity precedence constraints. This is not the case however for capacity constraints: there are situations with insufficient capacity that may go undetected by this constraint propagation technique. Accordingly a critical activity is one whose resource requirements are likely to conflict with the resource requirements of other activities. [11, 9] describes a technique to identify such activities. The technique starts by building for each unscheduled activity a probabilistic *activity demand*. An activity $A_k^{l\alpha}$'s demand for a resource $R_{kij}^{l\alpha}$ at time t is the probability that $A_k^{l\alpha}$ uses $R_{kij}^{l\alpha}$ at time t (to fulfill its resource requirement $R_{ki}^{l\alpha}$). Clearly activities with many possible start times and resource reservations tend to have smaller demands at any moment in time, while activities with fewer possible reservations tend to have higher ones. In a second step, each agent aggregates his activity demands as a function of time, thereby obtaining his *agent demand*. This demand reflects the need of the agent for a resource as a function of time[4]. Finally, agent demands are aggregated for the whole system thereby producing *aggregate* demands that indicate the degree of contention among agents for each of the (shared) resources in the system, in 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 activity's demand to the aggregate demand for a resource over a highly contended-for time interval is the *activity criticality*.

To choose the next activity to schedule, each agent looks for the resource/ time interval that it may need with the highest aggregate demand. He then picks his activity with the highest contribution (i.e. highest criticality) to the aggregate demand for that resource/time interval. In other words, each agent looks for the resource/time interval over which he has some demand that is

[3]For a more complete description of these measures, the reader is referred to [11, 9].

[4]Notice that, an agent's demand at some time t for a resource is obtained by simply summing the demand of all his unscheduled activities at time t. Because these probabilities do not account for limited capacity, their sum may actually be larger than 1

the most likely to be involved in a capacity constraint violation. He then picks his activity with the highest probability of being involved in the conflict.

2.1.2. Reservation Ordering

Once an agent has selected the activity to schedule next, he must decide which reservation to assign to that activity. Here several strategies can be considered. 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. Agents using such heuristic attempt to select the reservation that is the least likely to prevent other activities to be scheduled. In other words an agent will select the reservation that will be the least constraining both to himself and to other agents. This heuristic results in *altruistic* behavior on the part of the agent.
2. **A "Greedy" Value Ordering Strategy (GV)**: At the other extreme, an agent can select reservations based solely on his local preferences, i.e. irrespectively of his own future needs as well as those of other agents. This heuristic results in *egotistic/myopic* behavior on the part of the agent.

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 agent). Experiments in centralized scheduling [9] 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 agents 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 agents 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 agents, and the amount of resource contention.

3. Using Texture Measures in a Decentralized Multiagent Scheduling System

In the decentralized case, we have a set of agents that communicate in an asynchronous manner via message passing and each of which has a set of orders to schedule on a set of resources. Each order consists of several activities. Typically some of the resources are required by several agents and conversely, each agent requires some resources that are also needed by others. Which particular resources are shared may change with the set of orders to be scheduled. In our model, resources are passive objects that are *monitored* by active agents. Monitoring resources does not give an agent any preferential treatment concerning the allocation of the monitored resources but is simply a mechanism that enables the system to perform load balancing and efficient detection of capacity constraint violations. A capacity constraint violation (resource conflict) is detected when an agent requests a resource reservation for an activity for a time interval that is already reserved for another activity. Monitoring agents perform the additional tasks of (a) integrating certain pieces of information for shared resources (see step IV of protocol below) so as to avoid duplication of effort, which would be the case if all agents were doing this information integration, and (b) keeping the calendar of the resources they monitor. Typically, each agent in the system is a monitoring agent for some shared resources and conversely each resource is monitored by some agent. Since there is no single agent that has a global system view, the allocation of the shared resources must be done by collaboration of the agents that require these resources (the monitoring agent is usually one of those that require the shared resources)[5].

We have identified two levels of interaction of the agents: 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 out of which the agents 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 the scheduling agents build their schedules in a cooperative manner. The two texture measures identified in the previous section provide a framework for cooperation where the agents exchange demand profiles, and reservations. Demand profiles are aggregated periodically to compute textures that allow agents to form expectations about the resource demands of other agents. Because of communication overhead, the demand profile information is restricted. Subsets of the agents 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 agents 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 an agent backtracks, the change in scheduling reservations may ripple through to the other agents and cause them to change their reservations.

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

I. Each agent determines required resources by checking the process plans for the orders it has to schedule. It sends a message to each monitoring agent for needed shared resources informing it of the need[6].

[5]This model mirrors actual factory floor situations where the factory is divided into work areas that might share resources, such as machines, fixtures and operators in order to process orders.

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

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

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

IV. Based on the agent demands for a resource, the monitoring agent aggregates the agent demands to form the *aggregate demand* for the resource and sends it to all the agents that share the resource.

V. Upon receipt of the aggregate demand for each of the shared resources that it needs, an agent finds its most critical resource/time-interval pair and its most critical activity (the one with the greatest demand on this resource for this time interval). Since agents in general need to use a resource for different time intervals, the most critical activity and time interval for a resource will in general be different for different agents. The agent calculates a desired reservation for its most critical activity 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 agents for resource reservations, thus minimizing backtracking, or whether it wants to increase the quality of its schedule. The agent communicates this reservation request to the resource's monitoring agent.

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

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

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

1. The agent that has secured the reservation first, does not relinquish it. The second agent has to calculate an alternative, if there is one left, else he has to backtrack.
2. The agents negotiate a new mutually satisfactory assignment of time intervals to the competing activities. These new reservations are communicated to the monitoring agent who undoes the previous reservation and installs the new ones.

The system terminates when all activities of all agents 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 because of backtracking. This assumption has two consequences: 1) Once an agent has been granted a reservation, this reservation is not automatically undone because some other agent had to backtrack and may now need the reservation. In such situations, the two agents have to negotiate to decide whether the reservation will get undone. 2) If an agent 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 agents on these resources remain as they were. This policy may result in non-optimal reservation for other agents since it denies the other agents the opportunity to take advantage of the canceled reservations of the backtracking agent, but it results in less computationally intensive performance.

4. An Example

We present an example that illustrates the use of textures to predict resource utilization and make reservations in a distributed environment. For simplicity, the example assumes a two-agent system with a single shared resource, R1. We further assume that both agents have calculated demand profiles for R1 and have communicated them to R1's monitor-agent. 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 activity for each agent within this interval, and (4) application of heuristics by each agent to select a reservation for its critical activity.

The curves in Figure 1 show the aggregate-demand profile (1a), agent demand (1b) and activity 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 agent-demands at each time point for agents α, β and the demand of the monitoring agent. Recall that each agent demand is calculated from the activity demands for all its activities that use R1. Agents 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, agents α and β use this to select an activity to schedule. In this case, the agents are focusing on the same critical resource interval. Typically, agents often focus on different resources because they proceed asynchronously and have different resource requirements. Figures 1c and 1e show the activity demands for activities which have non-zero demand in the critical resource/time-interval. In this step, referred to as variable selection, each agent determines the activity with the greatest demand **within the critical resource/time-interval** [30, 40]. This is activity A_1^{α} for agent α and A_3^{β} for agent β (agent β has only one activity with non-zero demand in this interval). Note that the only interval considered at this point is [30, 40], so other activities can have much higher demand in other intervals, but are not selected.

Finally, each agent chooses a reservation for its selected activity (i.e. performs value-selection). Figures 1d and 1f illustrate how comparison of an activity'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 agent β's activity A_3^{β} is 0.7 (1.3 - .6), indicating that a reservation at this interval is likely to result in conflict with another activity. 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 activities is the minimum of the difference between the aggregate demand and the activity demand (for all

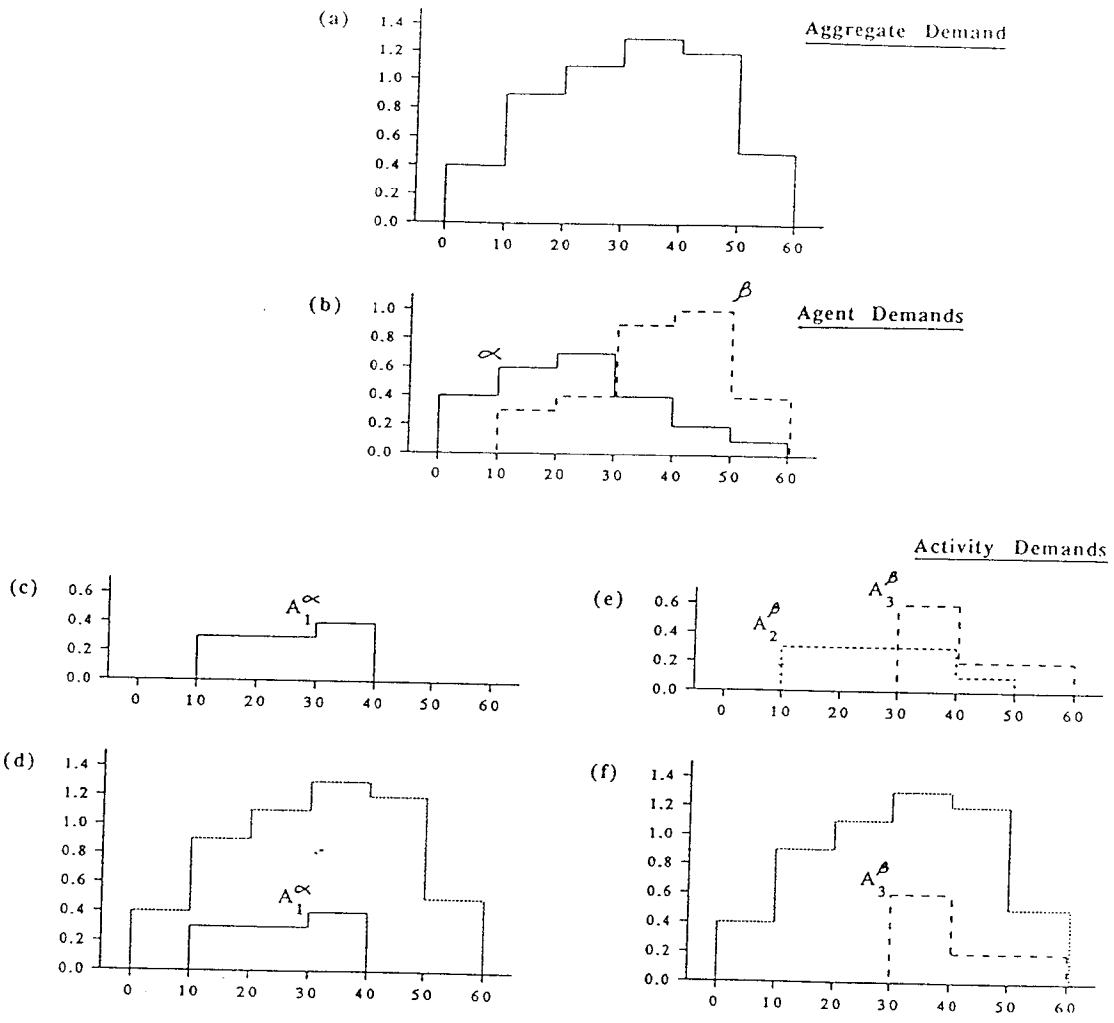

Figure 1: Demand for a single resource as a function of time. These curves are used for variable and value selection.

intervals where the activity's demand is non-zero). The LCV heuristic, which selects the least constraining value, will select interval [50, 60] for activity A_3^β. Similarly, agent α will select interval [10, 20] (the agents have selected non-overlapping time intervals).

5. Preliminary Experiments and Concluding Remarks

In this paper we have presented mechanisms to guide distributed search. The domain of investigation is distributed factory scheduling. In particular, we have presented measures of characteristics of a search space, called textures, that are used to focus the attention of agents during search and allow them to make good decisions both in terms of quality of system solution and performance. In addition, the textures express the impact of local decisions on system goals and allow agents to form expectations about the needs of others. We have presented two types of textures, their operationalization into variable and value ordering heuristics and their use in distributed problem solving. In addition, a communication protocol that enables the agents to coordinate their decisions has been presented. We have experimented with the variable and value ordering heuristics in a single agent case obtaining good results. A testbed has been implemented that allows for experimentation with a variety of distributed protocols that use variable and value ordering heuristics based on the probabilistic framework described in subsection 2.1. 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.

Preliminary experiments indicate that solutions to distributed scheduling problems can be found using these texture measures for several of the problems previously used to test the centralized scheduler. In particular, two important variables affecting system performance are (1) the degree to which schedulers share resources and (2) the frequency with which they exchange texture information. We have also discovered several cases in which

solutions were not found by the distributed system for problems which could be solved by the centralized scheduler. These occur primarily when one agent has made reservations which allow it to schedule all its activities, but creates an over-constrained problem for another agent. This situation is unresolvable under the current protocol, which does not provide a mechanism for agents to request reservations to be undone.

The preliminary results suggest the need for mechanisms to alter previous reservations even when they lead to local solutions by: requesting backtracking (undoing reservations and restarting search), requesting local shifts in reservations which do not produce conflicts, and/or allowing relaxation of constraints. These solutions will require the expansion of the negotiation component of the protocol described previously.

Acknowledgements

We wish to thank Joe Mattis for developing the KM Message Passing Utility, which was used to implement the distributed testbed. We also wish to acknowledge his work in developing the user-interfaces for both the distributed and centralized systems and his substantive contributions to the design and implementation of the testbed.

References

[1] 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.

[2] Ow, P.S., S.F. Smith, and A. Thiriez, "Reactive Plan Revision", *Proceedings AAAI-88*, St. Paul, Minnesota, August 1988.

[3] Karmarkar, U.S., "Alternatives for Batch Manufacturing Control", Tech. report QM 86-13, University of Rochester, June 1986.

[4] 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.

[5] 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.

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

[7] Durfee, E.H., *A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network*, PhD dissertation, COINS, University of Massachusetts, 1987.

[8] 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.

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

[10] Fox, M.S., *Constraint-Directed Search: A Case Study of Job Shop Scheduling*, PhD dissertation, Computer Science Department, Carnegie-Mellon University, 1983.

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

[12] 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)*, Elsevier Science Publishers, held in Sophia Antipolis, France, May 1987.