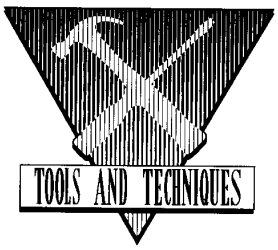

Resource Allocation in Distributed Factory Scheduling

Katia P. Sycara, Steven F. Roth, Norman Sadeh, and Mark S. Fox
Carnegie Mellon University

Both the operations research and the artificial intelligence communities have investigated factory scheduling in depth,[1,2] but almost noone has researched distributed scheduling. While a few researchers in distributed AI have studied this area,[3,4] they have focused on problems where machine agents contend only for computational resources, such as computer time and communication bandwidth.[5,6] In most real-world situations, however, allocation of noncomputational resources — for example, machines — must be made to various areas of the factory so that orders can be scheduled. Factory managers need mechanisms to control cooperative distributed-resource allocation over time — that is, the distributed scheduling of resources. We have initiated the Cortes project to construct a decentralized, heterogeneous, multiagent production control system with which to study the impact of different production control architectures. In our system, factory areas are modeled as machine agents, which perform the distributed factory scheduling. Agents run on separate workstations and communicate via networked message passing.

Both pragmatic and theoretical considerations motivated our investigations. For example, 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 inherently distributed, so their various parts must be scheduled to optimize the performance of the whole. The inherently decentralized nature of the activities being coordinated requires a corresponding decentralization of planning and control mechanisms. From the theoretical viewpoint, we are interested in investigating coordination techniques as available information degrades or becomes incomplete, and as each machine agent's need to react increases due to potentially conflicting scheduling decisions of other agents.

Our initial work focused on formalizing the concept of constraint-directed search within a centralized framework.[7] We identified a set of so-called texture measures that quantify several characteristics of the problem space being searched. We hypothesized that these textures can predict the impact of local decisions on system goals and that they express the expectations of agents' resource needs. We developed these textures into heuristics that direct the searches conducted by scheduling agents. We successfully applied these heuristics to direct the searches of a centralized activity-based scheduler, achieving good schedule quality and minimizing the likelihood of backtracking.[7] Our current model uses textures to direct distributed scheduling, optimizing decisions in the global search space by allowing agents to focus and direct searches in their individual search spaces.

Distributed scheduling is a process carried out by a group of agents, each of which has

0885/9000/91/0200-0029 $1.00 © 1991 IEEE

Manufacturing organizations are inherently distributed, so their various parts must be scheduled to optimize the performance of the whole.

- limited knowledge of the environment,
- limited knowledge of the constraints and intentions of other agents, and
- a limited number and amount of resources to produce a system solution.

Many agents might share these resources. Agents make local decisions about assigning resources to specific activities at specific time intervals, and a complete order schedule is cooperatively created by incrementally merging agents' partial schedules. No single agent has a global system view. The distributed AI system arrives at global solutions by interleaving local computations with the information exchanged among agents. The system goal is to find schedules that are not only feasible, but that also optimize a global objective — for example, minimizing order tardiness or work in process. The global objective to be optimized reflects the quality of schedule produced.

Another equally important concern is the efficiency of schedule generation. Since backtracking is a fact of life in scheduling, systems need approaches that incrementally produce schedules while minimizing backtracking. Our approach, based on texture measures, considers both optimization components — the quality of schedules as well as the efficiency of schedule generation.

Distributed scheduling has the following characteristics:

- The global system goal is to schedule a set of activities in a distributed fashion to optimize global criteria and minimize backtracking.
- To achieve global solutions, agents must consistently allocate resources needed to perform system activities. Because resources have limited capacity, system conflicts arise in the form of contention over optimal allocation of shared resources.
- Due to a limited communication bandwidth, it is not possible to exchange a complete set of specific constraints.
- Because of conflicts over shared resources, agents generally cannot optimize the scheduling of their assigned orders using only local information.
- All orders must be scheduled; in other words, agents cannot drop any local goals.
- Certain constraints (for example, the precedence constraints among the operations of an order) cannot be relaxed.
- When local computations produce infeasible schedules, the agent has to backtrack and try again. Backtracking can invalidate resource reservations that other agents have made, causing major ripple effects on the network.

At each stage of scheduling, agents need mechanisms to predict and evaluate the impact of scheduling decisions on global system goals, to form expectations and predictions about the resource needs and behavior of other agents, and to help focus attention opportunistically on specific parts of their search space. Good predictive measures are very important for several reasons. Because of ripple effects, backtracking is more costly when it involves many agents. Second, an agent might ask another agent to estimate its needs for resources that it might not otherwise consider until it has made many other reservations. Third, since they operate asynchronously, agents must predict and take into consideration the future needs of other agents that are in an earlier stage of scheduling. Finally, since communication is costly, predictive measures must be robust and accurate over many scheduling decisions.

The distributed factory scheduling model

Consider a set of scheduling agents, $\Gamma = \{\alpha, \beta, ...\}$. Each agent α is responsible for scheduling a set of orders $O^\alpha = \{O_1^\alpha, ..., O_N^\alpha\}$. Each order O_1^α consists of a set of activities $A^{l\alpha} = \{A_1^{l\alpha}, ..., A_n^{l\alpha}\}$ to be scheduled according to a process plan, or routing. The plan specifies a partial ordering among these activities — for example, $A_v^{l\alpha}$ before $A_w^{l\alpha}$. Each order also has a release date and a latest acceptable completion date, which might actually be later than the ideal due date. Each activity $A_k^{l\alpha}$ requires one or more resources $R_{k,i}^{l\alpha}$ $(1 \le i \le p_k^{l\alpha})$, and each resource can have one or more alternatives (that is, substitutable resources) $R_{k,i,j}^{l\alpha} (1 \le j \le q_{k,i}^{l\alpha})$. The system contains a finite number of resources. Some resources are required by only one agent and are said to be local to that agent. Other resources are shared, in the sense that they can be allocated to different agents at different times.

We distinguish between two types of constraints: activity precedence constraints and capacity constraints. Activity precedence constraints, together with order release dates and latest acceptable completion dates, restrict the set of acceptable start times of each activity. Capacity constraints restrict the number of activities to which a resource can be allocated at any one time to the capacity of that resource. For the sake of simplicity in this article, we only consider resources with a capacity of 1. Typically, limited resource capacity induces interactions between orders that are simultaneously competing for the same resource. These interactions can take place between the orders of one agent or different agents.

For each activity, we associate preference functions that

map each possible start time and each possible resource alternative onto a preference. These preferences arise from global organizational goals, including

- reducing order tardiness (meeting due dates),
- reducing order earliness (reducing the finished-goods inventory),
- reducing order flowtime (reducing the in-process inventory),
- using accurate machines, and
- performing certain activities during some shifts rather than others.[7]

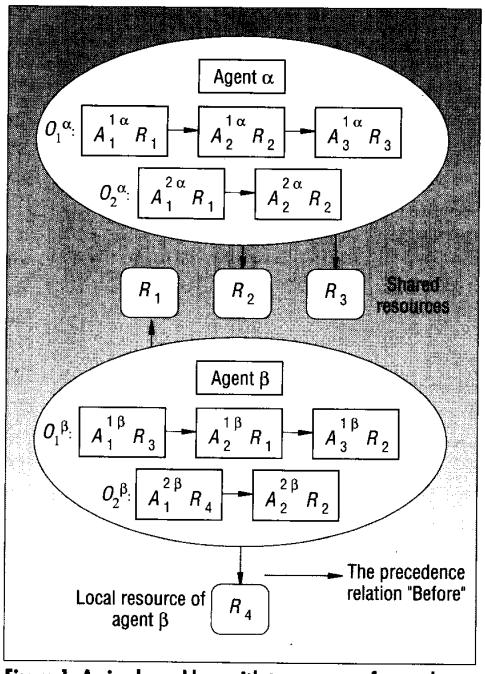

Figure 1. A simple problem with two agents, four orders, and four resources.

In the cooperative setting we are assuming here, the sum of these preferences over all agents in the system and over all activities scheduled by these agents defines a global-objective function to be optimized. We can view the sum of these preferences over all the activities under one agent's responsibility as that agent's local view of the global-objective function. In other words, individual agents do not know the global-objective function. Furthermore, because agents compete for shared resources, it is not enough for individual agents to try to optimize their own local preferences. Instead, agents need to consider the preferences of other agents when they schedule their activities.

Figure 1 shows a simple example with two agents, α and β. Agent α has two orders, O_1^α and O_2^α. Agent β has two orders as well, O_1^β and O_2^β. The figure shows each order's required activities and resource requirements. For instance, order O_1^α has a process plan with three activities: $A_1^{1\alpha}$ (which requires resource R_1), $A_2^{1\alpha}$ (which requires resource R_2), and $A_3^{1\alpha}$ (which requires resource R_3). The arrows between activities represent precedence constraints — for example, $A_1^{1\alpha}$ has to be performed before $A_2^{1\alpha}$.

In this example, each activity has only one resource requirement, and each requirement has only one alternative (for example, $R_{1,1}^{1\alpha} = R_1$). In other words, the only variables in this problem are the activity start times. For the sake of simplicity, we will also assume that time has been represented in discrete units with a granularity of 1, that all activities have the same duration of three time units, that all orders are released at time 0 and have to be completed by time 15. Resources R_1, R_2, and R_3 are shared, since they are required by both agents, and R_4 is local to agent β. We also assume that agent α is responsible for monitoring R_2 and R_3. Agent β monitors R_1 and its own local resource, R_4. R_2 is the only resource required by four activities (one activity in each order). All other resources are required by fewer activities.

Our model treats each activity $A_k^{l\alpha}$ as an aggregate variable (or vector of variables). It also defines a value as a reservation for an activity, consisting of a start time and a set of resources for that activity (that is, one resource $R_{k,i,j}^{l\alpha}$ for each resource requirement $R_{k,i}^{l\alpha}$ of $A_k^{l\alpha}$, $1 \le i \le p_k^{l\alpha}$).

By iteratively selecting an activity to be scheduled and a reservation (value) for that activity, each agent asynchronously and incrementally builds a schedule for the orders assigned to that agent. Each time an agent schedules a new activity, new constraints are added to the distributed Cortes system that reflect the new activity reservation. The system propagates these new constraints both within the agent (in a local constraint propagation step) and across agents (a constraint communication step followed by a local constraint propagation step). The system backtracks if it detects an inconsistency (that is, a constraint violation) during propagation. Otherwise, the agent moves on and looks for a new activity to schedule and a reservation for that activity. The process continues until all activities have been successfully scheduled.

If an agent could always make sure that the next reservation to be assigned will not result in some constraint violation — forcing that agent or others to undo earlier decisions — the system could schedule without backtracking. Because scheduling is NP-hard, most researchers believe that such look-ahead cannot be performed efficiently. Our system uses the constraint propagation mechanism described by LePape and Smith.[8] For the sake of efficiency, this mechanism does not try to guarantee total consistency, but instead looks for two types of inconsistency that are easy to spot: violations of precedence constraints within an order, and violations of capacity constraints between a group of scheduled activities and one unscheduled activity.

This mechanism does not immediately detect violations of capacity constraints between unscheduled activities. These are usually detected later, when the system has scheduled all but one of the activities involved in the conflict. This type of conflict appears to be more difficult to detect (possibly requiring exponential time). Under these

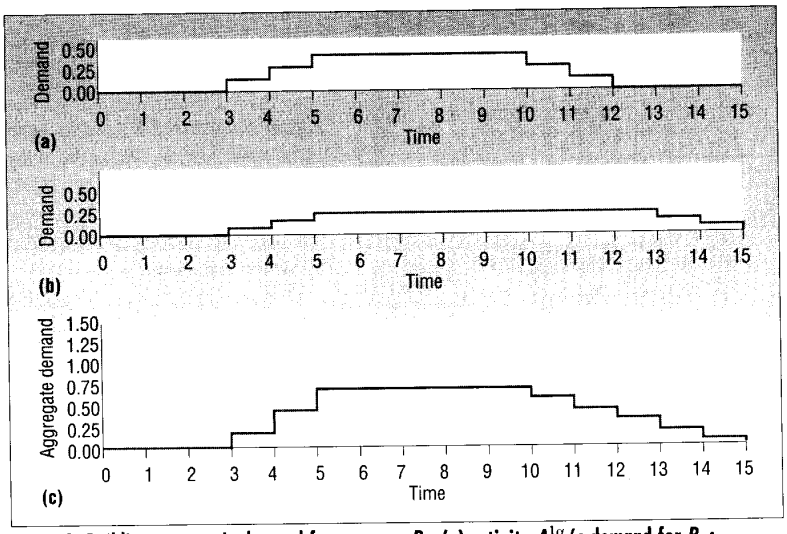

Figure 2. Building agent α's demand for resource R_2: (a) activity $A_2^{1\alpha}$'s demand for R_2; (b) activity $A_2^{2\alpha}$'s demand for R_2; (c) agent α's demand for R_2.

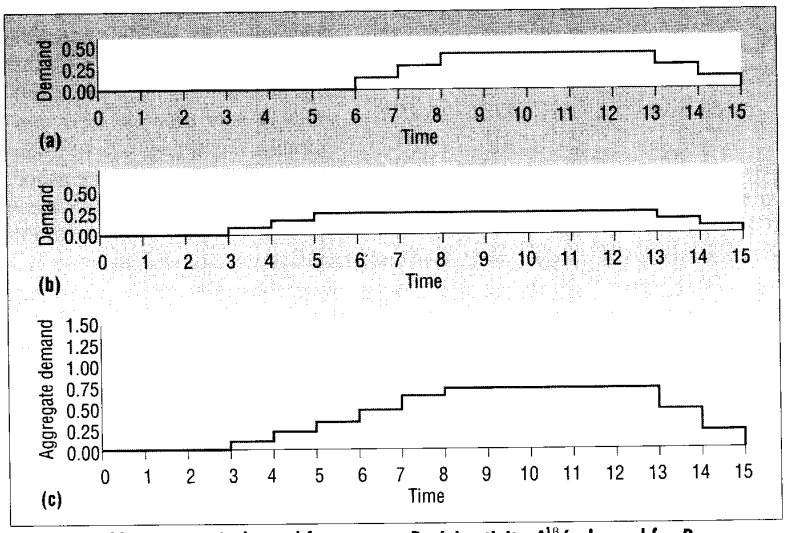

Figure 3. Building agent β's demand for resource R_2: (a) activity $A_3^{1\beta}$'s demand for R_2; (b) activity $A_3^{2\beta}$'s demand for R_2; (c) agent β's demand for R_2.

conditions, a reservation can force agents (even the agent who made the original reservation) to backtrack later. This was the case in the centralized version of our model and, because of the additional cost of communication, it is even more deleterious in the distributed case. Consequently, it is important to focus search in a way that reduces the chances of needing to backtrack and minimizes the work to be undone when backtracking occurs.

We accomplish this via two techniques, the variable-ordering heuristic (for activities) and the value-ordering heuristic (for reservations). The measures assigned by these techniques are examples of texture measures.

Ordering activities. The variable-ordering heuristic assigns a criticality measure to each unscheduled activity, so that the system schedules the activity with the highest criticality first. The criticality measure approximates the likelihood that the activity will be involved in a conflict. This technique only deals with conflicts that cannot be detected by constraint propagation. By scheduling the most critical activity first, agents reduce their chances of wasting time building partial schedules that cannot be completed, thereby reducing the frequency and the damage of backtracking.

Sadeh and Fox describe a technique to identify critical activities, whose resource requirements are likely to conflict with those of other activities.[7] The technique starts by building a probabilistic activity demand for each unscheduled activity. We determine the demand of activity $A_k^{l\alpha}$ for a resource $R_{k,i,j}^{l\alpha}$ at time t by calculating the number of possible reservations for $A_k^{l\alpha}$ that require $R_{k,i,j}^{l\alpha}$ at time t, divided by the total number of possible reservations for $A_k^{l\alpha}$. 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 demands.

In the technique's second step, agents aggregate their activity demands as a function of time, each obtaining an agent demand. This demand reflects an agent's need for a resource over time, given the activities still to be scheduled. We obtain an agent's demand for a resource at time t by simply summing the demands of all of that agent's unscheduled activities at time t. Because these probabilities do not account for limited resource capacities, their sum can be larger than 1.

Figures 2 and 3 illustrate the process by which agents α and β compute their total agent demands for resource R_2. For instance, agent α has two activities requiring R_2: $A_2^{1\alpha}$ and $A_2^{2\alpha}$. First, α computes the probabilistic demand of each of these

two activities (see Figure 2) and then sums these two numbers, producing α's total demand for R_2. Activities $A_2^{1\alpha}$ and $A_2^{2\alpha}$ have the same total demand, which equals their duration, but it has been spread over the different possible start times of each activity. Because $A_2^{1\alpha}$ has fewer possible start times than $A_2^{2\alpha}$, its demand is more compact than that of $A_2^{2\alpha}$.

Finally, for each shared resource, the system aggregates all agent demands, producing aggregate demands that indicate the degree of contention among agents for each shared resource as a function of time. Time intervals over which a resource's aggregate demand is very high correspond to capacity constraint violations that are likely to go undetected by constraint propagation. The contribution of an activity's demand to the aggregate demand for a resource over a highly contended time interval reflects that activity's reliance on that resource in that time interval. We define this as the criticality of the activity.

In our example, agent α communicates to β its total agent demand for resource R_1, and agent β communicates to α its total agent demands for R_2 and R_3. In the next step, α and β compute aggregate demand profiles for each shared resource, thereby obtaining the system's global demand for each resource as a function of time. Figure 4 illustrates the aggregation process for R_2: agent α aggregates its own demand for that resource together with the demand it received from agent β. The computed aggregate demands are then communicated back to each contributing agent. (Communication is unnecessary for R_4, since it is a local resource of β.) At the end of this phase, agent α has three aggregate demand curves (one for each of the three shared resources), and agent β has four (three for the shared resources, and one for its local resource). Figure 5 presents these curves. Resource R_2 has the largest peak of demand, indicating it is the resource for which the highest

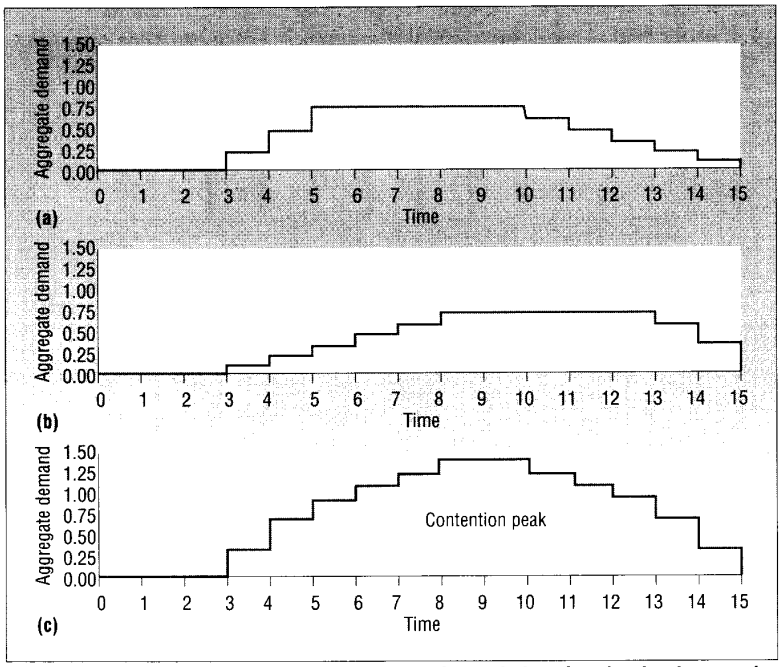

Figure 4. Agent α aggregates its demand for R_2 together with that of β, thereby obtaining the system's aggregate demand for R_2: (a) agent α's demand for R_2; (b) agent β's demand for R_2; (c) aggregate demand for resource R_2.

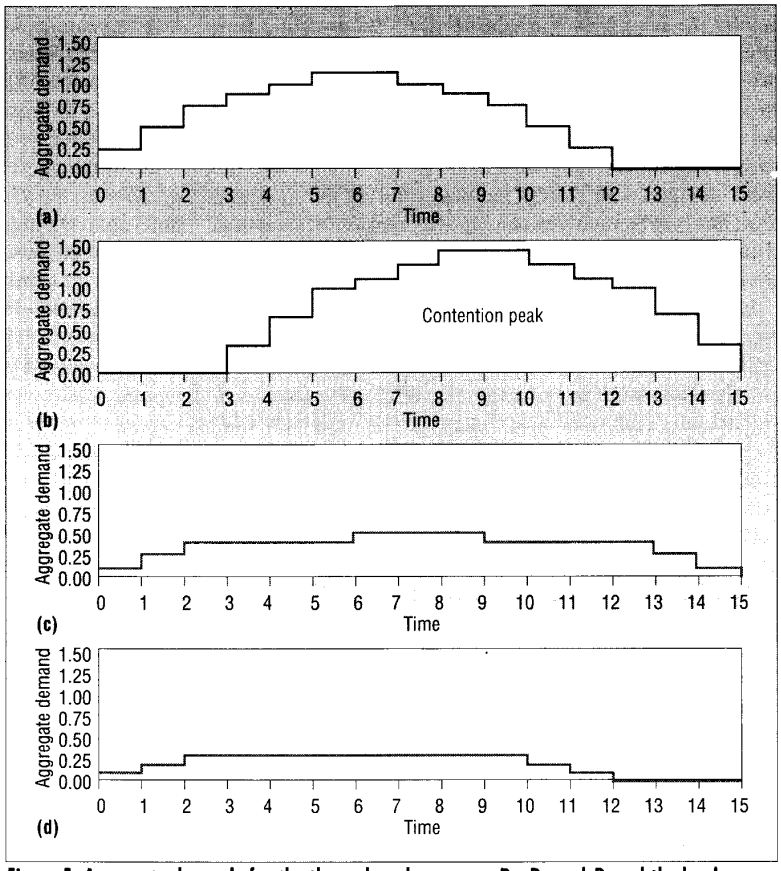

Figure 5. Aggregate demands for the three shared resources R_1, R_2, and R_3 and the local resource R_4: (a) aggregate demand for R_1; (b) aggregate demand for R_2; (c) aggregate demand for R_3; (d) aggregate demand for R_4.

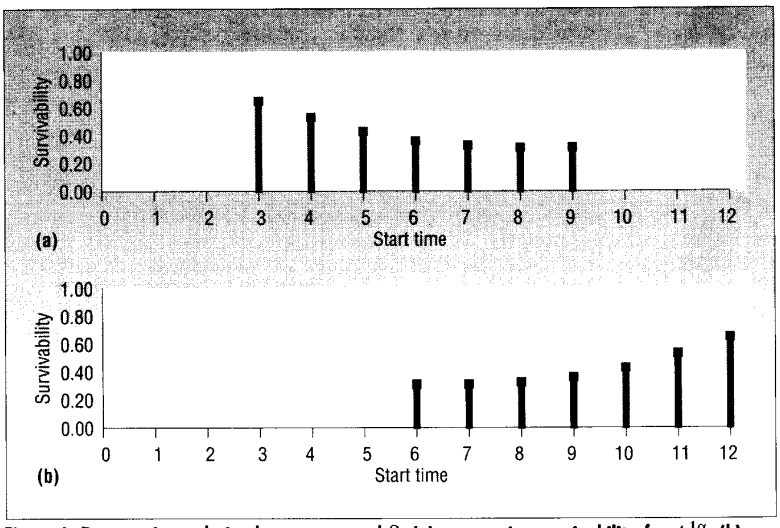

Figure 6. Reservation ordering by agent α and β: (a) reservation survivability for $A_2^{1\alpha}$; (b) reservation survivability for $A_3^{1\beta}$.

contention exists — that is, it is the main bottleneck resource.

To choose the next activity to schedule, each agent looks among the resources it might need within a certain time interval, and picks the one with the highest aggregate demand. It then picks the activity with the highest contribution (that is, the highest criticality) to the aggregate demand for that resource/time interval. In other words, each agent inspects all the resource/time intervals for which it still has some demand, and selects the time interval that it deems most likely to be involved in a capacity constraint violation. It then picks the activity with the highest probability of being involved in the conflict.

In our example, both agents α and β require resource R_2. Agents only consider time intervals whose duration equals the average duration of the activities requiring the resource — three time units in this example. While two time intervals actually qualify as most contended — [7,10] and [8,11] — we will assume that the agents both pick [8,11] as their most contended one. Agent α picks activity $A_2^{1\alpha}$ as its most critical activity, since it is the activity (among the two needing R_2) that relies most on that time interval. That is, the contribution of $A_2^{1\alpha}$ to the demand peak is larger than that of $A_2^{2\alpha}$ (see Figure 2). Similarly, agent β picks $A_3^{1\beta}$ as its most critical activity (see Figure 3).

Ordering reservations. Once an agent has selected the activity to schedule next, it must decide which reservation to assign to that activity. The system can consider several strategies, including the following two extremes:

• A least-constraining-value ordering strategy (LCV) — Agents try to select the reservation least likely to prevent other activities from being scheduled by itself or by other agents. This heuristic results in the agent behaving altruistically.

• A greedy-value ordering strategy (GV) — At the other extreme, an agent can select reservations based solely on its local preferences, irrespective of its own future needs or those of other agents. This heuristic results in the agent behaving egotistically and myopically.

A continuum of strategies lies between these two extremes, combining features of both LCV and GV approaches. These intermediate strategies factor in a reservation's contribution to the global-objective function, together with the likelihood that selecting that reservation will result in backtracking (either locally or for another agent). Experiments in centralized scheduling indicate that LCV heuristics are best at minimizing searches but usually result in poor schedules, since this method selects reservations irrespective of their contribution to the objective function.[7] GV heuristics usually produce significantly better schedules, but result in extra backtracking, so the search takes longer.

However, we can reduce backtracking considerably by combining the GV value-ordering heuristic with the variable-ordering heuristic, reflecting the compensatory relation between these heuristics. Because agents in the decentralized case schedule asynchronously, we expect the effect of the variable-ordering heuristic to be weaker. Also, the cost of backtracking in a distributed system is higher than in a centralized one due to the overhead in coordinating agents whose earlier decisions are interdependent. We expect a higher need for least-constraining behavior in a distributed scheduling environment. The ultimate choice of an intermediate strategy depends on such factors as the time available to come up with a solution, the load of the agents, and the amount of resource contention.

Figure 6 illustrates the behavior of agents α and β when they both use an LCV heuristic. Each agent assigns a survivability measure to each possible start time of the activity to be scheduled. This measure simply estimates the probability that assigning this start time to the activity will not result in a capacity constraint violation. For each possible start time, the agent looks at the time interval that it would need to reserve on R_2's calendar. The heuristic determines the likelihood of another activity requiring that same time interval by considering the number of other activities contributing to the demand for that time interval

and their total demand. Intervals in which other activities have low demands are usually the least likely to result in a conflict, so they correspond to the least constraining start times.[7]

Figures 2 and 3 show that the earlier agent α schedules activity $A_2^{1\alpha}$, the less likely this reservation will conflict with agent β's resource requirements (since β's demand for R_2 is the lowest over these time intervals). This is reflected by the survivability measures computed by α for the different possible start times of $A_2^{1\alpha}$. The least constraining of these start times is time 3. Accordingly, agent α decides to schedule $A_2^{1\alpha}$ to start at that time, reserving R_2 over the interval [3,6]. Similarly, agent β selects time 12 as the least constraining start time for $A_3^{1\beta}$, and reserves R_2 over the interval [12,15]. The LCV heuristic works as expected, and the agents end up selecting nonoverlapping intervals. All this occurs without the agents knowing about each other's activities. The agents only exchanged their total agent demands, not their individual activity demands.

Applying textures to multiagent scheduling

Additional theoretical concerns arose as we applied the texture approach to decentralized, multiagent, resource-constrained scheduling. We addressed several difficulties in the multiagent situation, including

- reasoning with incomplete information about the intentions and future behavior of other agents;
- scheduling with uncertain or changing information — even when detailed information regarding other agents' intentions is communicated, this information is not stable over time, since agents are scheduling asynchronously; and
- scheduling without coordinating agents helping to avoid conflicts and achieve global goals.

Multiagent scheduling with incomplete information. In a multiagent system, complete information is unavailable to each agent about the constraints, partial plans, schedules, and heuristic analyses of other agents. Only limited interagent communication can reasonably occur, since a narrow communication bandwidth requires some summarization and abstraction. Our approach expresses summarization information in terms of the texture measures that have been effective for centralized problems. Specifically, we represent an agent's intended resource use in terms of that agent's *demand density* for the resource for different time intervals; that is, the extent to which an agent expects to use a resource is expressed as the aggregate of its expectations that each of its activities will require the resource over the time period for which it is available. We further abstract all agent densities to produce an *aggregate density*, which represents the system-wide expectation for resource utilization over time.

The system goal is to find schedules that are not only feasible, but that also optimize a global objective — for example, minimizing order tardiness or work in process.

Because of the abstraction process, agents cannot consider the specific quantities underlying aggregate densities while solving problems (as they could in the centralized case). For example, while agents can determine that a high collective demand exists for a resource for some interval, they cannot determine whether that demand results from a single activity with high criticality or from many less-critical activities (or even if the demand originates with one agent or many).

Multiagent scheduling with rapidly changing information. The continuous, asynchronous behavior of agents can reduce the validity of exchanged information, regardless of how complete that information is. Therefore, an agent cannot depend on the certainty of information, because other agents' decisions interact with already constructed partial schedules as well as with future scheduling decisions, repeatedly producing new expectations. Also, because of the associated communication costs, agents cannot afford to communicate, update, and evaluate information with every change that occurs. The information communicated must remain robustly predictive in the face of communication lags.

Several aspects of the texture approach address this problem of rapid information obsolescence. First, texture measures produce relatively accurate early predictions of agent behavior, as long as all agents communicate their expectations at the beginning of scheduling and as long as constraints remain constant. Second, the uniform representation of expectations as densities and the incremental nature of activity scheduling allows the system to incorporate changes in expectations as soon as it receives them. Third, agents can monitor their own current expectations to determine when these have changed significantly from those that were last communicated.

Heuristics that minimize the undoing of planning decisions, thereby avoiding the ripple effects of backtracking, are crucial to maintaining stability in distributed scheduling. Variable-ordering strategies focus on the most constrained resources and activities first. Since these are activity-scheduling decision points for which the fewest initial alternatives exist, it becomes less likely that these will be undone later. Our LCV value-ordering heuristic helps produce stable, incrementally refined schedules, in that agents

In the texture approach, we can coordinate multiagent scheduling by having agents accept and adhere to shared decision-making policies.

can react flexibly to changes in other agents' future decision-making behavior (as reflected in changing demand densities) with minimal backtracking.

Agents can reduce the effects of rapid information obsolescence by monitoring their expectations to determine the most opportune time to communicate. Agents can initiate the exchange of densities either when they determine that their own plans for future decisions have changed significantly or when they infer that other agents' plans have changed. Agents determine when to communicate expected changes in their own plans by heuristically comparing their current densities with those last communicated to other agents. For example, we can use the mean-squared difference between the previous and current densities over time as a measure of uncertainty.

Agents must also reason about the collective effects of changes in the expectations of other agents. This is especially important when no single agent has changed its expectations sufficiently to warrant communication to others, but significant net changes have occurred across all agents. For example, an agent can request others to exchange densities when many reservations have been made on globally critical resource/time intervals. This lets agents take advantage of change, rather than adhering to rigid communication protocols.

The changeable and incomplete nature of expectations can have additional adverse effects, requiring extensions to previous search techniques. Since LCV is only a heuristic, it cannot always prevent agents from making reservations that cause constraint violations for other agents. For example, an agent might reserve a resource in an interval that was the only potential interval within which another activity (belonging to another agent) could be scheduled without causing other activities to be late (that is, causing other activities to violate their due-date constraints). In other words, the second agent's activity still could be scheduled on its required resource prior to its own due date, but not without delaying subsequent activities past their due dates.

Agents detect these indirectly produced violations of their own constraints during the normal course of consistency checking, which occurs after making each reservation. At these points, the system propagates temporal constraints and partially checks capacity constraints to determine the impact of the new reservation on the rest of the schedule. In a centralized system, we can attribute constraint violations to the most recent reservation, since the system validates all previous reservations as new ones are made. As a result, a centralized system can handle violations simply, using chronological backtracking to undo the last reservation for an activity and then trying another one for that activity. In a multiagent system, other agents can make reservations throughout an agent's search, making it difficult to determine which set of previous reservations were responsible for a constraint violation when it is eventually detected.

At this point, an agent must find the last set of reservations it made that, together with those made by other agents, does not violate constraints. A simple backtracking procedure will eventually find this state, but this is extremely inefficient, because chronological backtracking requires trying every value (reservation) for a variable (activity) before backtracking to a previous variable (trying a different reservation for a previous activity). If an earlier-scheduled reservation combines with other agents' reservations to cause a constraint violation, the system has to try all alternative reservations for all activities scheduled since then before it reaches the relevant point. This produces computational loads that increase exponentially with the number of activities searched.

To deal with this problem, we have developed a variation of backjumping[9,10] for uncertain, multiagent environments. Our approach involves iteratively undoing each activity's scheduled reservation and determining whether constraint violations remain, until the set of acceptable activity reservations has been partitioned. We do not try any alternative values for any one activity until we have determined this set.

In other words, whenever we find a constraint violation, we do not try to undo other agents' reservations. An agent can only undo the reservations it has made (in backward chronological order) until it finds a set of reservations that are still feasible with all other agents' reservations. The scheduling process then resumes for the activities remaining to be scheduled.

This procedure avoids testing alternate values for variables when violations already exist for values assigned to previously addressed variables. Our version of backjumping locates the appropriate search point by incurring a computational load that is a linear function of the number of variables already traversed. This is a tremendous saving over chronological backtracking, which is a multiplicative function of the number of variables traversed times the number of possible reservations to be tried on each.

Multiagent scheduling without coordinating agents. In the texture approach, we can coordinate multiagent scheduling by having agents accept and adhere to shared decision-making policies. Through three policies, agents support each others' attempts to solve their portions of the global scheduling problem. First, by applying LCV value-

ordering heuristics, agents use information about other agents' expectations to avoid overconstraints. Second, the system grants agent requests to reserve uncontested resources on a first-come, first-served basis. Also, agents promptly surrender reservations if they decide not to use them as a result of local constraint violations. Third, once an agent has made a reservation, that agent does not have to surrender it — one agent cannot ask another to backtrack. An important principle is that all agents assume they can best realize the global good by applying these policies and, therefore, do not depart from them to maximize local objective functions.

A communication protocol for coordinating multiagent scheduling. In the decentralized case, agents communicate asynchronously by passing messages, and each agent has a set of orders to schedule on a set of resources. Each order consists of several activities. Several agents usually need the same resources and, conversely, each agent needs some resources that are also needed by others. Which resources are shared can change with the set of orders to be scheduled. In our model, resources are passive objects monitored by active agents. Resource monitoring does not give an agent preferential treatment in resource allocation; it simply helps the system balance loads and detect capacity constraint violations. This detection occurs when an agent requests a resource reservation for a time interval that is already reserved for another activity. To avoid duplication of effort, only monitoring agents integrate certain information for shared resources and keep the calendar of the resources they monitor. Typically, each agent is a monitoring agent for some shared resources, and each resource is monitored by some agent. Since no single agent has a global system view, the system must allocate shared resources through collaboration of the agents that need these resources — the monitoring agent is usually one of those that require the shared resource. This model mirrors actual situations where the factory floor is divided into work areas that might share resources (for instance, machines, fixtures, and operators) to process orders.

We have identified two levels of agent interaction: the strategic level, where aggregate information is communicated, and the tactical level, where information about specific entities is communicated. Strategic information consists of the demand profiles out of which agents calculate criticality measures for their decision making. At the tactical level, agents make specific scheduling decisions and negotiate if needed.

Because they can contend for the same resources, scheduling agents must build their schedules cooperatively. The two texture measures we have identified provide a framework for cooperation in which agents exchange demand profiles and reservations. The system periodically

Several agents usually need the same resources and, conversely, each agent needs some resources that are also needed by others.

aggregates demand profiles and computes textures to help agents form expectations about other agents' resource demands. Because of communication overhead, the demand profile information is restricted. Agents communicate demand profiles for only the resources they share, although reservations on nonshared resources can affect scheduling decisions on the shared ones. Since several agents are scheduling asynchronously and the communicated demand profiles are limited to the subset of shared resources, this system architecture involves a higher uncertainty. This uncertainty also varies inversely with the frequency at which demand profiles are communicated. Moreover, the cost of backtracking is greater, since any scheduling change due to backtracking can ripple through to other agents and cause them to change their reservations.

Specifically, our multiagent communication protocol works as follows:

(1) Each agent determines its required resources by checking the process plans for the orders it has to schedule. It sends a message to all monitoring agents (as specified in a table) informing them that it will be using shared resources.

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

(3) Each agent determines whether its new demand profile differs significantly from its previous ones. If it has changed, the agent sends the new profile to the monitoring agent.

(4) The monitoring agent combines all agent demands and communicates the aggregate demand to all agents that share the resource. Except for the first time demands are exchanged, agents do not wait for aggregate demands to be computed and returned before continuing their scheduling operations, although they can postpone further scheduling if desired.

(5) Using the most recent aggregate demand, each 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 typically need to use a resource for different time intervals, the most critical activity and time interval for a resource usually differs for different agents. Each agent communicates its reservation request to the resource's monitoring agent and awaits a response.

(6) The monitoring agent checks the resource calendar for resource availability. If the resource is available for the requested time interval, the monitoring agent

Texture measures allowed near-perfect performance (solving the problem in 40 states) when the texture information was updated frequently.

communicates "reservation OK" to the requesting agent, marks the reservation on the resource calendar, and communicates the reservation to all the agents that had sent positive demands on the resource. If the resource had already been reserved for the requested interval, the monitoring agent denies the request. The agent whose request is denied tries to substitute another reservation, if feasible; otherwise, it performs backjumping.

(7) Hearing that its request is granted, the agent performs consistency checking to search for constraint violations. If it detects none, the agent proceeds to step 2. Otherwise, it backjumps, withdrawing reservations until the system reaches a search state with no constraint violations. Any reservations withdrawn during this phase are communicated to the monitor for distribution to other agents. Once the system reaches a consistent state, the agent proceeds to step 2.

The system terminates when it has scheduled all the activities of all agents. We have based backjumping in this version of the protocol on two design decisions. First, once the system grants a reservation to an agent, it does not automatically undo this reservation when another agent (who had to backjump) needs it. This design decision can lead to situations where one agent solves its local scheduling problem but another agent cannot due to unresolvable constraint violations.

We also decided that when an agent backjumps and frees up resources, other agents' reservations on these resources remain as they were. This policy can result in nonoptimal reservations since agents cannot take advantage of canceled reservations, but it results in less computationally intensive performance.

Experimental results

We wanted to compare the performance of multiagent and centralized schedulers. We also wanted to

- determine the feasibility of the texture approach to multiagent scheduling,
- test specific mechanisms and parameters that influence system performance, and
- study the heuristics' roles in the tradeoff of solution quality versus amount of searching and the influence of frequency of texture communication on system performance.

We created our experiments based on problems found to be difficult in previous research on centralized scheduling.[7] These experiments reflect system performance with respect to search efficiency rather than schedule optimality. We implemented the testbed in Knowledge Craft running on top of Common Lisp. It can be run on networked MicroVAX 3200s under VMS.

Specifically, our experiments considered

- the effects of agents' incomplete knowledge of each other's plans — that is, the robustness of texture measures when aggregated across multiple agents and with the resulting loss of detailed information;
- the effects of rapidly changing expectations on performance — that is, the robustness of these measures with respect to delays in communicating densities; and
- the consequences of asynchronous scheduling (for example, asynchronously using variable-ordering strategies) without external coordination.

We selected all our experimental problems such that orders could be distributed evenly between two agents, all resources were shared by the two agents (high interagent resource coupling), every order used all resources, and problems ranged from 40 to 100 activities. Varying several properties in each problem, we ran more than 90 experiments. In each case, the dependent variable was the efficiency with which the scheduling system found a solution, expressed by the total number of states needed to reach a solution. For example, for a problem with 40 activities, the minimum number of states needed to assign a reservation to each activity is 40. Every reservation that needed to be redone added an additional state to the total. This let us compare a 40-activity one-agent problem to a pair of 20-activity problems solved simultaneously by two agents.

Problem versions differed in several ways. First, to establish a baseline, we created a one-agent system that resembled the two-agent system in every way except that aggregate densities were constructed from a single agent. This was still different from the original centralized system in that decisions were based on an abstract aggregate (for example, the aggregate did not include detailed information about the number of activities contributing to the densities). Furthermore, we were able to vary the frequency with which the aggregate was computed, thereby isolating the effect of uncertain expectations caused by infrequent and delayed communication of densities in the two-agent system.

We implemented several simplified versions of the heuristic used by agents to determine when to communicate their changed densities. In the zero-delay condition, any agent's reservation on any resource initiated the exchange of densities for all resources. For increased delays, agents independently exchanged densities for each resource

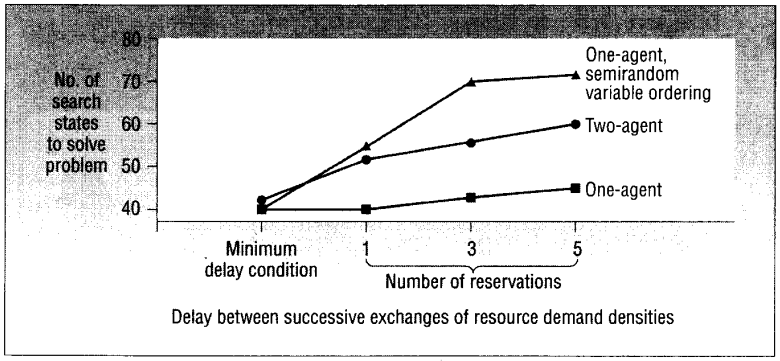

Figure 7. Experimental comparisons of one- and two-agent systems.

whenever N reservations were made ($N = 1, 3$, and 5). This gave us a way to observe the effects of wide variation in communication bandwidth in a two-agent system and comparable conditions in a one-agent system.

Another version of the one-agent system used a semirandom-version variable-ordering heuristic. We wanted to isolate and assess the effects of less accurate variable ordering that might occur in a multiagent system. Recall that our multiagent system performs variable ordering in parallel, each agent selecting the best activity to schedule from its subset of activities requiring a critical resource. Agents do not coordinate activity selection to ensure that the globally most critical ones are scheduled first. By selecting activities to schedule from those requiring the most critical resource/time interval, the semirandom heuristic narrows selection to a maximum of 20 percent of the activities in these problems. However, it randomly selects from this subset instead of selecting the activity with the greatest demand for the critical resource.

Finally, we created two scheduler versions to compare backtracking and backjumping search techniques. Figure 7 shows the results for a representative group of 40-activity experiments. Each data point is the average of three experimental runs, since exact experimental replication is impossible in asynchronous decentralized processing. The three curves represent the effects of increasing delays in updating aggregate demand densities for one- and two-agent configurations and for a one-agent case with a semirandom variable-ordering strategy.

Texture measures allowed near-perfect performance (solving the problem in 40 states) when the texture information was updated frequently. Thus, despite incomplete information in the two-agent system, texture measures provide satisfactory summaries. Second, as expected, the performance of the multiagent system deteriorates as the communication of changing texture information is delayed. Since each agent uses current texture information to perform both variable and value ordering, it is likely that both these processes deteriorate.

The effects of delaying communication and computation of demand densities are greater for the two-agent than the one-agent system. This interaction may reflect the compensatory relation between variable and value ordering.[7] In the two-agent case, variable ordering might be less effective because variables are chosen asynchronously. When the system renews texture measures frequently, it can select effective values for variables, thereby using valid information to compensate for poorer variable ordering. However, delaying density communication weakens value ordering, and performance declines. In the one-agent system, variable ordering is more effective, so the delay does not hurt performance as much. Results of the one-agent semirandom variable-ordering test support this view: the effect of partially disabling variable ordering accelerates with increasing delay, as it does in the two-agent case. Two-agent performance is still better than the semirandom condition, suggesting that a variable-ordering strategy is robust in multiagent environments, which deal with incomplete, changeable information and asynchronous behavior without external coordination.

The semirandom condition is still highly selective relative to completely random variable ordering (in that only activities using the most critical resource/time interval are considered). In fact, we found that random variable ordering resulted in terrible performance, even in the one-agent case. Finding a solution required more than 500 states.

Finally, as expected, the backjumping strategy substantially reduced search in the two-agent system. Performance using chronological backtracking was highly variable and degraded exponentially with delayed communication. Searches exceeded 300 states when the system waited for three reservations before initiating communication.

Our research shows that texture measures can play four important roles in distributed search.

(1) They provide guidance in making decisions.

(2) They focus each agent's attention on globally critical decision points in its local search space.

(3) They are good predictive measures of the impact of local decisions on system goals.

(4) They can be used for modeling the beliefs and intentions of other agents.

Texture measures represent plan expectations robustly, even when more detailed information is lacking. Although

the frequency with which texture measures are communicated affects the efficiency with which schedules are produced, the measures are robust enough to complete scheduling efficiently.

Acknowledgments

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.

References

1. S.F. Smith, M.S. Fox, and P.S. Ow, "Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems," *AI Magazine*, Vol. 7, No. 4, 1986.

2. S.C. Graves, "A Review of Production Scheduling," *Operations Research*, Vol. 29, No. 4, 1981, pp. 646-675.

3. H.V. Parunak et al., "A Distributed Heuristic Strategy for Material Transportation," *Proc. 1986 Conf. Intelligent Systems and Machines*, 1986. This reference can be obtained from the authors.

4. S.F. Smith and J.E. Hynynen, "Integrated Decentralization of Production Management: An Approach for Factory Scheduling," *Proc. ASME Annual Winter Conf.: Symposium on Integrated and Intelligent Manufacturing*, American Society of Mechanical Engineers, New York, 1987.

5. S. Cammarata, D. McArthur, and R. Steeb, "Strategies of Cooperation in Distributed Problem Solving," *Proc. Eighth Int'l Joint Conf. AI (IJCAI-83)*, Morgan Kaufmann, San Mateo, Calif., 1983, pp. 767-770.

6. E.H. Durfee, *A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem-Solving Network*, doctoral dissertation, Dept. of Computer and Information Sciences, Univ. of Massachusetts, Amherst, Mass., 1987.

7. N. Sadeh and M.S. Fox, "Focus of Attention in an Activity-Based Scheduler," *Proc. NASA Conf. Space Telerobotics*, NASA, Bethesda, Md., 1989.

8. C. LePape and S.F. Smith, "Management of Temporal Constraints for Factory Scheduling," *TAIS 87: Proc. IFIP TC 8/WG 8.1 Working Conf. on Temporal Aspects in Information Systems*, Elsevier Science, New York, 1987.

9. J. Gaschnig, "Performance Measurement and Analysis of Certain Search Algorithms," Tech. Report CMU-CS-79-124, Computer Science Dept., Carnegie Mellon Univ., Pittsburgh, Penn., 1979.

10. R. Dechter, "Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition," *Artificial Intelligence*, Vol. 41, 1989, pp. 273-312.

Katia P. Sycara is a research scientist in Carnegie Mellon University's Robotics Institute and the director of the Laboratory for Enterprise Integration. Her research interests include knowledge-based systems, case-based reasoning, distributed systems, negotiation models for multiagent planning, and constraint-directed reasoning, with special emphasis on integrating operations research and AI methods to address manufacturing problems. She received her PhD in computer science from the Georgia Institute of Technology, her MS from the University of Wisconsin, and her BS from Brown University. She is a member of the IEEE Computer Society, AAAI, ACM, and the Institute for Management Science.

Steven F. Roth is director of the Information Presentation and Interfaces Lab at Carnegie Mellon's Robotics Institute. He received his PhD in cognitive psychology from the University of Pittsburgh and his AB in psychology from the State University of New York at Stony Brook. His interests have included applying AI technologies to project management, transportation scheduling, distributed factory scheduling, and computer-assisted instruction. He is currently working on the System for Automatic Graphics and Explanation, investigating intelligent interfaces that automatically design graphical displays of information. He is a member of AAAI, ACM, the American Psychological Association, and the American Educational Research Association.

Norman Sadeh is a PhD candidate in computer science at Carnegie Mellon University, where he expects to graduate in early 1991. He is a member of the Production Control Laboratory and the Manufacturing Architecture Laboratory at Carnegie Mellon's Center for Integrated Manufacturing Decision Systems, where he developed the Cortes factory scheduler. His research interests include the application of AI and operations research techniques to engineering and management problems, centralized and decentralized planning and scheduling, distributed problem solving, constraint satisfaction, and constrained optimization. He received an engineering degree in electrical engineering and physics from the École Polytechnique at the Université Libre de Bruxelles, and an MS in computer science from the University of Southern California. He is a fellow of the Belgian American Educational Foundation.

Mark S. Fox is associate professor of computer science and robotics at Carnegie Mellon University and director of the Center for Integrated Manufacturing Decision Systems. His research interests include knowledge representation, constraint-directed reasoning, and applications of AI to engineering and manufacturing problems. He received his PhD from Carnegie Mellon University and his BSc from the University of Toronto, both in computer science. He is a member of the IEEE, AAAI, ACM, the Society of Manufacturing Engineers, the Institute for Management Science, and the Canadian Society for Computational Studies of Intelligence.

The authors can be reached in care of Katia Sycara, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.