Distributed Constrained Heuristic Search

K. Sycara, S. Roth, N. Sadeh, and M. Fox.
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper we present a model of decentralized problem solving, called Distributed Constrained Heuristic Search (DCHS) that provides both structure and focus in individual agent search spaces so as to optimize decisions in the global space. The model achieves this by integrating decentralized constraint satisfaction and heuristic search. It is a formalism suitable for describing a large set of DAI problems. We introduce the notion of textures that allow agents to operate in an asynchronous concurrent manner. The employment of textures coupled with distributed asynchronous backjumping (DAB), a type of distributed dependency-directed backtracking that we have developed, enables agents to instantiate variables in such a way as to substantially reduce backtracking. We have experimentally tested our approach in the domain of decentralized job-shop scheduling. A formulation of distributed job-shop scheduling as a DCHS is presented as well as experimental results.

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.

1. Introduction

In this paper, we present a framework for formalizing a set of DAI problems by extending Constrained Heuristic Search (CHS) [Fox 89] to multi-agent environments. A *distributed Constrained Heuristic Search (DCHS* problem) is a CHS problem where the solution is the result of cooperative multi-agent problem solving. The CHS problem solving paradigm addresses a subclass of problems that can be solved through state space search. Similarly, DCHS addresses a subclass of DAI problems that can be solved through distributed search. This methodological commitment is consistent with other research that formulates DAI problems in terms of search. We are currently engaged in investigating and further developing the CHS model in both single and multi-agent settings.

The CHS problem solving model provides both structure and focus to search in the problem space. The model achieves this by combining the process of constraint satisfaction (CSP) with heuristic search (HS). The resulting model both reduces search complexity and provides a more formal explanation of the nature and power of heuristics in problem solving. Although both CSP and HS have been extensively studied for single agent problem solving, with the notable exception of [Yokoo 90], there have been no attempts at studying these formalisms in a multi-agent setting. However, formal investigations of distributed CSP and HS problem solving models is very important because, as has been pointed out in [Yokoo 90], (1) Various DAI problems can be formulated as distributed CSP or distributed HS problems (e.g., [Lesser 90, Conry 88, Kuwabara 89]) and (2) distributed CSP and HS models can provide formal frameworks for investigating various DAI issues and methodologies, such as decision making coherence (e.g., [Durfee 87]) and organizational re-design (e.g., [Ishida 90]).

CHS integrates the synthetic capabilities of heuristics search with the structural characteristics of constraint satisfaction techniques. Constraint satisfaction algorithms are viewed as taking giant steps, not creating new objects, but reducing the entire space of objects to a satisficing set. (This assumes the ability to enumerate a set of objects from which to choose.) On the other hand, search techniques can be synthetic in that they incrementally construct a solution as part of the search process. In formulating the framework for distributed CHS, we are concerned with the principles behind how knowledge can be used to structure and guide asynchronous distributed search in the problem space of individual agents so as to optimize overall problem solving of the multi-agent system[1].

In this paper, we first review the elements of the CHS problem solving model. We then introduce the definition of DCHS and give the general process by which agents perform asynchronous DCHS. Section 4 formulates the distributed job-shop scheduling problem as a DCHS. The variable and value ordering heuristics that have been developed are presented in detail. Section 4.3 presents distributed asynchronous backjumping (DAB). Section 5 presents the communication protocol that allows concurrent asynchronous problem solving by the agents. Section 6 presents experimental results and section 7 presents concluding remarks.

---

[1] A problem space is composed of an initial state that defines the problem's initial conditions, a set of operators that generate new states and an evaluation function that identifies solution states.

2. Overview of Constrained Heuristic Search

CHS augments the definition of a problem space, composed of states, operators and an evaluation function, by refining a state to include:

1. **Problem Topology**: Provides a structural characterization of a problem.
2. **Problem Textures**: Provide measures of a problem topology that allows search to be focused in a way that reduces backtracking.
3. **Problem Objective**: Defines an objective function for rating alternative solutions that satisfy a goal description.

This model allows us to (1) view problem solving as constraint optimization, thus taking advantage of optimization techniques, (2), incorporate heuristic search, thus allowing the dynamic modification of the constraint model, and (3) extend constraint satisfaction to the larger class of optimization problems.

We define problem topology as a graph G, composed of vertices V and edges E:

$V = N \cup C \cup S$

where

N is a set of variables $\{n_1, n_2, ..., n_m\}$

C is a set of constraints $\{c_1, c_2, ..., c_n\}$

S is a set of satisfiability specifications $\{s_1, s_2, ..., s_o]$

Each variable in N may be a vector of variables whose domains may be finite/infinite and continuous/discrete. Constraints are n-ary predicates over variables vertices. A constraint predicate is true iff the instantiations of these variable vertices are compatible with each other. A satisfiability specification vertex groups constraints into sets of type AND, OR, or XOR. An XOR satisfaction set denotes that only one constraint in the set must be satisfied. Edges link constraint vertices to variable vertices, and satisfiability specifications to constraints. Finding a solution to a CHS problem consists in finding an assignment of values to all variables that satisfies all constraints and all satisfiability specifications.

We distinguish between two types of problem topologies:

Definition 1: A *completely structured problem* is one in which all non-redundant vertices and edges are known a priori.

This is true of all CSP formulations.

Definition 2: A *partially structured problem* is one in which not all non-redundant vertices and edges are known prior to problem solving.

This definition tends to be true of problems in which synthesis is performed resulting in new variables and constraints (e.g. the generation of new subgoals during the planning process).

Operators in CHS have many roles: refining the problem by adding new variable and constraint vertices, reducing the number of solutions by reducing the domains of variables (e.g., assigning a value to a variable vertex), or reformulating the problem by relaxing constraints or omitting constraints and/or variables.

The general CSP is a well-known NP-complete problem [Garey 79]. There are however classes of CSPs that do not belong to NP, and for which efficient algorithms exist. The CHS

methodology is meant for those CSPs for which there is no efficient algorithm. A general paradigm for solving these problems consists in using Backtrack Search (BT) [Golomb 65, Bitner 75]. BT is an enumerative technique that incrementally builds a solution by instantiating one variable after another. Each time a new variable is instantiated, a new search state is created that corresponds to a more complete partial solution. If, in the process of building a solution, BT generates a partial solution that it cannot complete (because of constraint incompatibility), it has to undo one or several earlier decisions. Partial solutions that cannot be completed are often referred to as deadend states (in the search space). Because the general CSP is NP-complete, BT may require exponential time in the *worst-case*. Extensive experimentation with the centralized system [Sadeh 91], suggests that CHS provides a methodology for reducing the *average* complexity of BT by interleaving search with *local constraint propagation* and the computation of *texture-based heuristics*.

Local constraint propagation techniques are used to prune alternatives that have become impossible due to earlier decisions made to reach the current search state. By propagating the effects of earlier commitments as soon as possible, CHS reduces the chances of making decisions that are incompatible with these earlier commitments [Mackworth 85].

Typically, pruning the search space can only be done efficiently on a local basis [Nadel 88]. Hence local constraint propagation techniques are not sufficient to guarantee backtrack-free search. In order to avoid backtracking as much as possible as well as reduce its impact when it cannot be avoided, we need techniques for focusing the problem solver’s attention opportunistically to promising decisions. In CHS, for search to be well focused, that is to decide where in the problem topology an operator is to be applied, there must be features of the topology that differentiate one subgraph from another, and these features must be related to the goals of the problem. These features take the form of different types of constraint interactions. CHS analyzes the pruned problem space in order to determine critical variables, promising values for these variables, promising search states to backtrack to, etc. The results of this analysis are summarized in a set of textures that characterize different types of constraint interactions in the search space. We have identified and are experimenting with seven such problem textures: Value Goodness, Constraint tightness, Variable Goodness, Variable Tightness, Constraint Reliance, and Variable Tightness with respect to a set of constraints [Fox 89]. These textures are operationalized by a set of heuristics to decide which variable to instantiate next (so-called variable ordering heuristics), which value to assign to a variable (so-called value ordering heuristics), which assignment to undo in order to recover from a deadend, etc. These textures generalize the notion of constraint satisfiability or looseness defined by [Nadel 88][2] and apply to both CHSs (and CSPs) with discrete and continuous variables. We have generalized these textures so they can be used in Distributed CHS.

3. Distributed CHS

A distributed CHS problem is a problem where the variables are distributed among a set of agents. Each agent is responsible for a set of variables and their instantiation. Constraints and satisfiability conditions exist among variables under the jurisdiction of different agents. The instantiation of variables must satisfy these constraints and satisfiability specifications. We say

[2]Keng and Yun, [Keng 89] have defined related "criticality" and "cruciality" measures.

that a distributed CHS problem is solved iff an assignment is found of values to all variables of all agents such that all constraints and satisfiability specifications are simultaneously satisfied.

Distributed DCHS 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 requirements of other agents, and (c) limited number and amount of resources that are required to produce a system solution. 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, DCHS is an incremental process. Agents make local decisions about assignments of values to particular variables at particular times during problem solving and a complete solution is formed by incrementally merging partial solutions.

The Distributed CHS problem has the following characteristics:

- The global system goal is to assign in a distributed fashion a set of values to a set of variables so that all constraints are satisfied and backtracking is minimized.
- To achieve global solutions, agents have to make consistent variable instantiations. In our model, the variable instantiations are made concurrently and asynchronously. Each agent instantiates the variables under its control and communicates the variable values to agents that need to know these values.
- Due to limited communication bandwidth, it is not possible for agents to exchange a complete set of specific constraints.
- Because each agent has incomplete information, it is in general impossible for each agent to assign consistent values to variables such that constraints are satisfied using only local information.
- Local computations could produce inconsistent value assignments, ie value assignments that lead to constraint violations. When this happens one or more agent(s) has(have) to backtrack and try again. Backtracking can have major ripple effects on the network since it may invalidate value assignments that other agents have made. Moreover, asynchronous backtracking runs the risk of computing irrelevant action based on obsolete information.

There are two remarks in order here with respect to backtracking. First, the standard chronological backtracking [Mackworth 85] instantiates variables in some sequential order. In general backtracking search is exponential in the worst-case. The situation is worse in distributed asynchronous backtracking. One way to increase backtracking efficiency is through various forms of dependency-directed backtracking [deKleer 87, Gaschnig 79, Dechter 89]. We have implemented a variant of dependency directed backtracking for distributed, asynchronous problem solving, called distributed asynchronous backjumping (DAB) that substantially reduces distributed search (see sections 4.3 and 6).

Second, both empirical and analytical studies in the CSP literature show that it is possible to reduce backtracking by properly sequencing the order in which variables are instantiated. As a consequence, a lot of research in CSP has concentrated on developing good variable and value ordering heuristics. A good variable ordering heuristic is to instantiate the variables that are most tightly constrained. This way, the system avoids investing a lot of time building partial solutions that cannot be completed because some difficult variables had not yet been instantiated.

A good value ordering heuristic to reduce backtracking is a least-constraining one, namely one that leaves as much room as possible to other variables and their values so that the current partial solution can be completed without backtracking.

Because of the incomplete and asynchronously changing information in the distributed case, the agents at each stage of problem solving require additional constraint instantiation guidance in the form of mechanisms to (a) predict and evaluate the impact of their decisions on global system goals, (b) form expectations and predictions about the constraints of other agents, and (c) help focus the attention of the agents on particular parts of their search space asynchronously and opportunistically. Having good predictive measures is very important in the distributed case because:

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

The textures that have been identified in the previous section have been generalized to work for DCHS. Our hypothesis is that these textures are good predictive measures of the impact of local decisions on system goals and express expectations of the difficulty of satisfying constraints at various parts of the search space of agents. We have operationalized these textures into a set of heuristics that direct search in individual agent spaces.

Each agent's DCHS asynchronous problem solving contains the following steps:

- An initial state is defined composed of a problem topology,
- Constraint propagation is performed within the state,
- Texture measures and the problem objective are evaluated for the state's topology,
- Operators are matched against the state's topology, and
- A variable node/operator pair is selected and the operator is applied.

The application of an operator results in either adding structure to the topology, further restricting the domain of a variable, or reformulating the problem (e.g., relaxation). The results of operator application are then propagated both within the agent (local constraint propagation step) and across agents (a constraint communication step followed by a local constraint propagation step). If an inconsistency (i.e. constraint violation) is detected during propagation, the agent employs DAB. Otherwise the agent moves on and looks for a new variable node/operator pair to apply. The process goes on until all variables of all agents have been assigned consistent values that satisfy all constraints and satisfiability specifications.

By allowing search to be performed concurrently and asynchronously by several problem solving agents, we introduce two groups of tradeoffs:

1. A group of *design tradeoffs*, which is found under one form or another in the

design of most distributed systems. In order to maintain search efficiency at a level comparable to that in the centralized setting (and hence achieve higher overall responsiveness), agents in the distributed system need to maintain a high degree of global system awareness. Within the centralized CHS setting, system awareness is achieved via local constraint propagation and texture computation. Unfortunately the limited communication bandwidth of a distributed system restricts the amount of information that can be passed between portions of the search space that are under the control of different agents. In other words the limited communication bandwidth generally prevents agents from attaining a level of awareness similar to that in a centralized system. As a consequence there is a tradeoff between the amount of distribution and the ability of the system to maintain a search efficiency that entails an overall increase in system responsiveness. For instance, in the factory scheduling domain, it is a good practice to start scheduling bottleneck resources first. Depending on the number of scheduling agents, and the way jobs are distributed between these agents, it is more or less difficult for the system as a whole to identify the bottleneck resources, and coordinate the construction of the schedule around the scheduling of these bottlenecks. Most distributed systems have to deal with this tradeoff under one form or another, as it generally determines the proper level of distribution in the system for a given bandwidth (or the necessary bandwidth given a predetermined number of agents), proper ways to partition the search space among a set of agents, etc.

2. A group of *tactical tradeoffs*: Given a specific distributed system with a predetermined number of agents, a fixed communication bandwidth, and a partitioned search space, whose portions have been preallocated to different agents in the system, there is still a large number of tactical parameters that one can play with in order to increase overall system responsiveness. In the distributed CHS setting, these parameters include the selection of the local constraint propagation mechanisms, the texture-based heuristics, and the synchronization protocols. All the tradeoffs influencing the selection of these parameters in the centralized setting are still present, but they have changed: they have become more complex due to the limited bandwidth of the system and the ability of the agents to perform some computations asynchronously. Consider for instance the relation between variable and value ordering heuristics. When several agents are allowed to asynchronously instantiate variables, they lose some of the benefits of a good variable ordering. This problem can be remedied in two ways: either by using a less constraining value ordering heuristic (this would be equivalent to relaxing the due dates of jobs requiring a bottleneck resource) and/or by introducing some synchronization between the agents in order to enforce some degree of system-wide variable ordering (e.g. enforcing that bottlenecks are scheduled first by properly synchronizing the scheduling agents). Clearly both approaches have their inconveniences and put additional strains on the communication bandwidth. A less constraining value ordering heuristic can only be allowed via additional computational efforts locally and/or via additional inter-agent communication. Worse, using less constraining values typically translates into poorer solutions. On the other hand, enforcing synchronization between the agents can only be achieved via additional inter-agent communication, and restricts the amount of concurrent processing in the system.

In this paper, we assume a distributed architecture and focus on the study of some of the tactical tradeoffs discussed above. While the study of some of the design tradeoffs identified

above has been given some attention in the literature (e.g. [Fox 81, Cammarata 83, Durfee 85]), we do not know of any prior study of the tactical tradeoffs discussed in this paper, namely those involved in distributing the CSP/CHS paradigm. The next section demonstrates the application of the distributed CHS model to the problem of distributed job shop scheduling.

4. Distributed CHS Job-Shop Scheduling

Factory scheduling has been the subject of intense investigation by both Operations Research and AI communities (e.g., [Baker 74, Rinnooy Kan 76, Graves 81, Fox 83, Smith et. al. 86]). With few exceptions [Parunak 86, Smith&Hynynen 87], there has been almost no research in distributed scheduling. On the other hand, the Distributed AI community has focused its attention primarily on problems where agents contend only for computational resources, such as computer time and communication bandwidth (e.g., [Cammarata 83, Durfee 87]). In most real world situations, however, allocation of (non-computational) resources that are needed by an agent 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 distributed job shop scheduling problem is viewed as a distributed CHS problem where each activity is an aggregate variable whose values are reservations. Our activity-based approach to job-shop scheduling relies on the combination of local constraint propagation techniques with texture-based heuristic search [Sadeh 91]. A reservation consists of a start time and a set of resources to be allocated to the activity. Each activity constitutes a variable vertex in the problem topology. Activity precedence constraints are binary constraints represented by constraint vertices connected to two activity variable vertices. A capacity constraint vertex is associated with each resource and connected to all the variable vertices representing activities that can possibly use the resource. Each capacity constraint ensures that the corresponding resource will not be allocated to more than one activity at any given time.

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. These interactions can take place either between the order of a same agent or between the orders of different agents.

With each activity, we associate preference functions that map each possible start time and each possible resource alternative onto a preference. These preferences [Fox 83, Sadeh 91] arise from global organizational goals such as reducing order tardiness (i.e. meeting due dates), reducing order earliness (i.e. finished goods 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.

Figure 4-1 displays a simple example with 2 agents: α, and β. Agent α has two orders: o_1^{α} and o_2^{α}. Agent β also has two orders: o_1^{β} and o_2^{β}. The activities required by each order are specified along with their resource requirements. For instance,order o_1^{α} has a process plan with 3 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 the activities represent the precedence constraints (e.g. $A_1^{1\alpha}$ has to be performed before $A_2^{1\alpha}$,). In this simple example, each activity has only one resource requirement, for each of which there is only one alternative (e.g. $R_{11}^{1\alpha}=R_1$). In other words the only variables in this problem are the activity start times. We further assume that time has been discretized with a granularity of 1, that all the activities have the same duration, namely 3 time units, that all orders are released at time 0 and have to be completed by time 15[3]. Resources R_1, R_2, and R_3 are shared in this example, since they are required by both agent α and agent β. On the other hand, resource R_4 is local to agent β. Notice that resource R_2 is the only resource to be required by 4 activities (one in each order). All other resources are required by fewer activities. Later we will see that this resource is the main bottleneck of the problem. This example will also be used to illustrate how agents α and β coordinate to identify this bottleneck and avoid making conflicting reservations when scheduling operations requiring a shared resource.

In our model we view each activity $A_k^{l\alpha}$ as an aggregate *variable* (or vector of variables). A *value* is a reservation for an activity. It consists of a start time and a set of resources for that activity (i.e. one resource $R_{kij}^{l\alpha}$ for each resource requirement $R_{ki}^{l\alpha}$ of $A_k^{l\alpha}$, $1 \le i \le p_k^{l\alpha}$).

Each agent asynchronously builds a schedule for the orders he has been assigned. 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 system that reflect the new activity reservation. These new constraints are propagated both within the agent (local constraint propagation step) and across agents (a constraint communication step followed

---

[3]None of these assumptions is required by our scheduling system. They simply make the example easier to understand.

Figure 4-1: A simple problem with 2 agents, 4 orders, and 4 resources.

by a local constraint propagation step). If an inconsistency (i.e. constraint violation) is detected during propagation, the system backtracks. Otherwise the agent 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 earlier decisions, scheduling could be performed without backtracking. Because scheduling is NP-hard, it is commonly believed that such look-ahead cannot be performed efficiently. Instead the constraint propagation mechanism used in our system is the one described in [LePape&Smith 87]. For sake of efficiency, this mechanism does not attempt to guarantee total consistency, but instead looks for two types of inconsistencies that are easy to spot: violation of precedence constraints within

an order, and violation of capacity constraints between a group of scheduled activities and one unscheduled activity. The conflicts that are not immediately detected by this mechanism correspond to violations of capacity constraints between several unscheduled activities[4]. This is because this last type of conflict appears to be more difficult to detect in general (possibly requiring exponential time). Under these conditions, a reservation assigned by an agent to an activity may force other agents or the agent himself to backtrack later on[5]. 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.

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 [6].

4.1. Variable Ordering Scheduling Heuristic

As was just pointed out earlier, there are situations with insufficient capacity that may go undetected for a while by the constraint propagation technique, thereby causing the system to backtrack later on. Accordingly a critical activity is one whose resource requirements are likely to conflict with the resource requirements of other activities. [Sadeh 91] 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 determined by the ratio of reservations that remain possible for $A_k^{l\alpha}$ and require using $R_{kij}^{l\alpha}$ at time t over the total number of reservations that remain possible 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 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 over time,

---

[4]These conflicts are only detected later on, typically when all but one of the activities involved in the conflict have been scheduled.

[5]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.

[6]For a more complete description of these measures, the reader is referred to [Sadeh 90].

given the activities that he still needs to schedule[7].

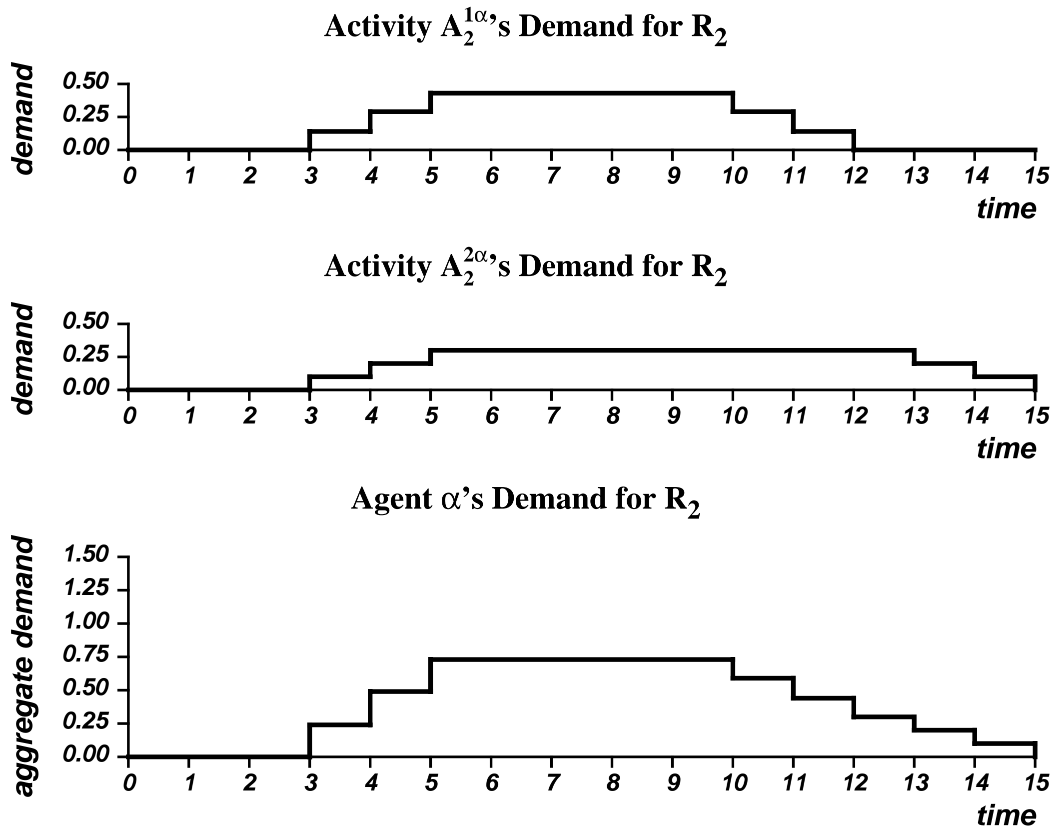

Figure 4-2: Building agent α's demand for resource R_2.

Figures 4-2 and 4-3 illustrate the process by which agent 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 $A_2^{2\alpha}$. In a first phase, the probabilistic demand of each of these two activities was computed by agent α, as illustrated in Figure 4-2. The computation is done as follows: Each agent calculates for each activity the set of remaining possible earliest-start-times and latest-start-times after existing reservations have already been taken into account. For example, activity $A_2^{1\alpha}$ has earliest-start-time of 3 and latest-start-time of 9 (to allow for scheduling of its preceding activity $A_1^{1\alpha}$, and its succeeding activity $A_3^{1\alpha}$). Similarly, activity $A_2^{2\alpha}$ has earliest-start-time of 3 and latest-start-time of 12 (to allow for scheduling its preceding activity $A_1^{2\alpha}$). Assuming that no reservations have been made yet, activity $A_2^{1\alpha}$. has 7 possible reservations on R_2 whereas activity $A_2^{2\alpha}$ has 10. Thus, the probabilistic demand for resource R_2 of $A_2^{1\alpha}$ for time interval [3, 4] is 1/7, for time interval [4, 5] it is 1/7 + 1/7=2/7, for interval [5, 10], it is 3/7, for interval [10, 11] it is 2/7 and for interval [11, 12] 1/7. The demand of activity $A_2^{2\alpha}$ is calculated in a similar fashion.

The two demands are then added up by the agent, thereby producing agent α's total demand

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

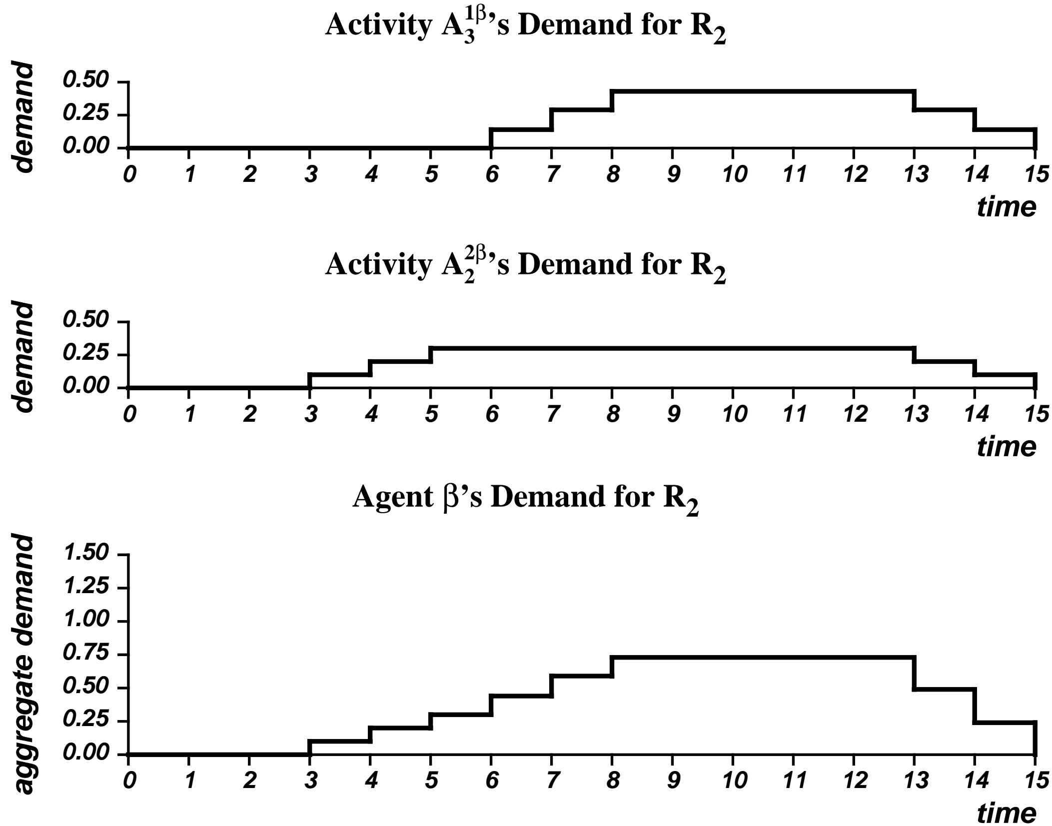

Figure 4-3: Building agent β's demand for resource R_2.

for R_2. Notice that $A_2^{1\alpha}$ and $A_2^{2\alpha}$ have the same total demand. This total demand is equal to 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, agent demands are aggregated for the whole system thereby producing *system-aggregate* demands that indicate the degree of contention among agents for each of the (shared) resources in the system as a function of time. Time intervals over which a resource's system 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 system's aggregate demand for a resource over a highly contended-for time interval reflects the reliance of the activity on the possession of that resource/ time interval. It is taken to be the *criticality of the activity*.

In our example, agent α and agent β communicate their agent demands for each of the shared resources to each other and aggregate them (i.e. take their sum) thereby obtaining the system's global demand for each resource as a function of time. So each agent knows the global systemwide demand at this point in time[8]. The exact communication protocol used by the

[8]To avoid the computational cost of (a) each agent exchanging its agent demand with every agent it shares resources with, and (b) replicating the systemwide aggregation computation in each agent, in the implemented system, a number of agents, called monitors, one for each resource, play the role of systemwide demand density aggregators.

scheduling agents is described in section 5. Figure 4-4 illustrates the aggregation process for resource R_2.[9]. At the end of this phase, agent α has three aggregate demand curves: one for each of the 3 shared resources, and agent β has 4 aggregate demand curves: the ones for the 3 shared resources and the one for its local resource R_4. These curves are represented in Figure 4-5. Notice that resource R_2 has the largest peak of demand, thereby indicating that it is the resource for which there is the highest contention (i.e. the main bottleneck resource).

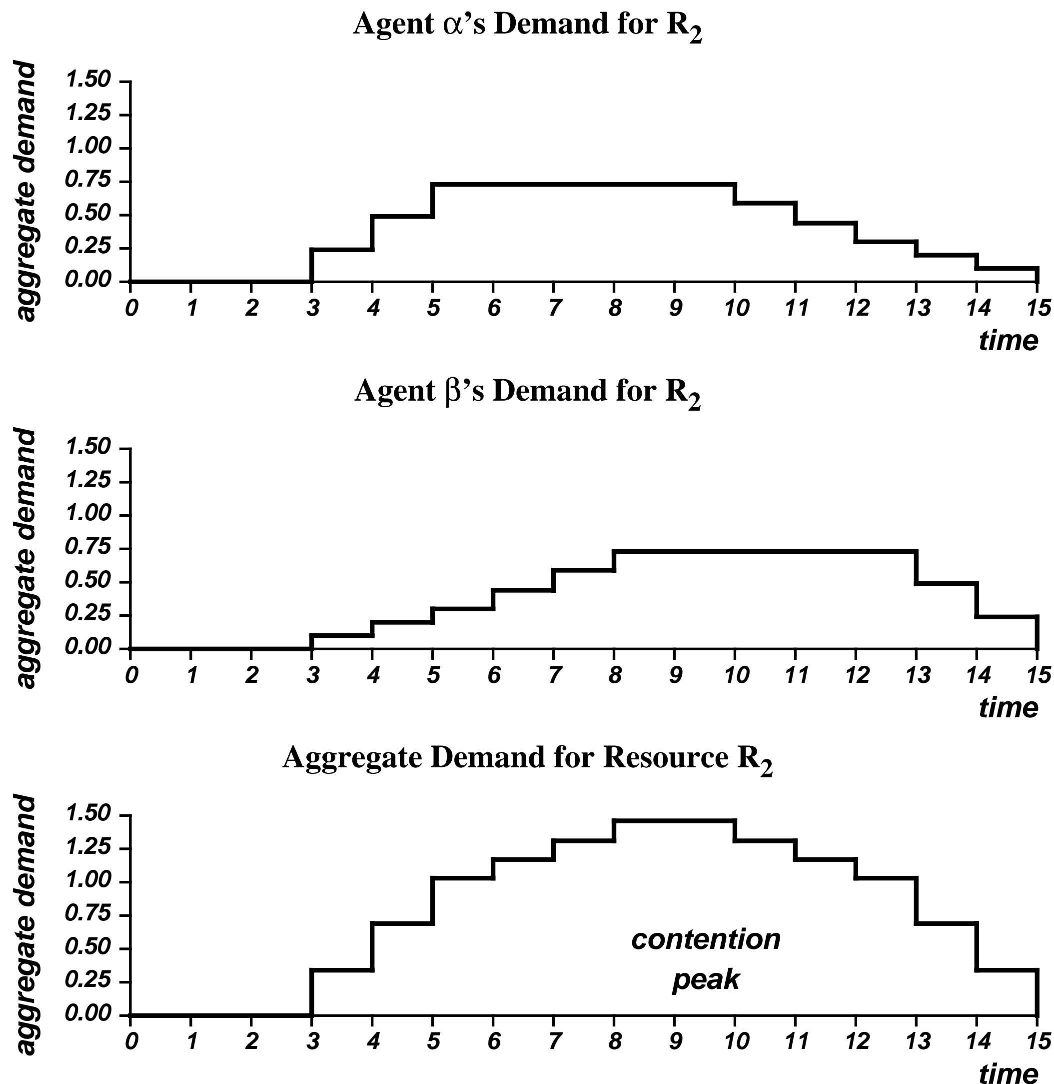

Figure 4-4: Agent and aggregate demands for resource R_2.

To choose the next activity to schedule, each agent first looks at all the resource/time intervals on which it has some non-zero demand and picks the one with the highest aggregate demand. In our example, both agents α and β happen to have their highest aggregate demand on resource R_2.

[9]Notice that in the case of resource R_4, there is no need for communication: R_4 is a local resource of agent β.

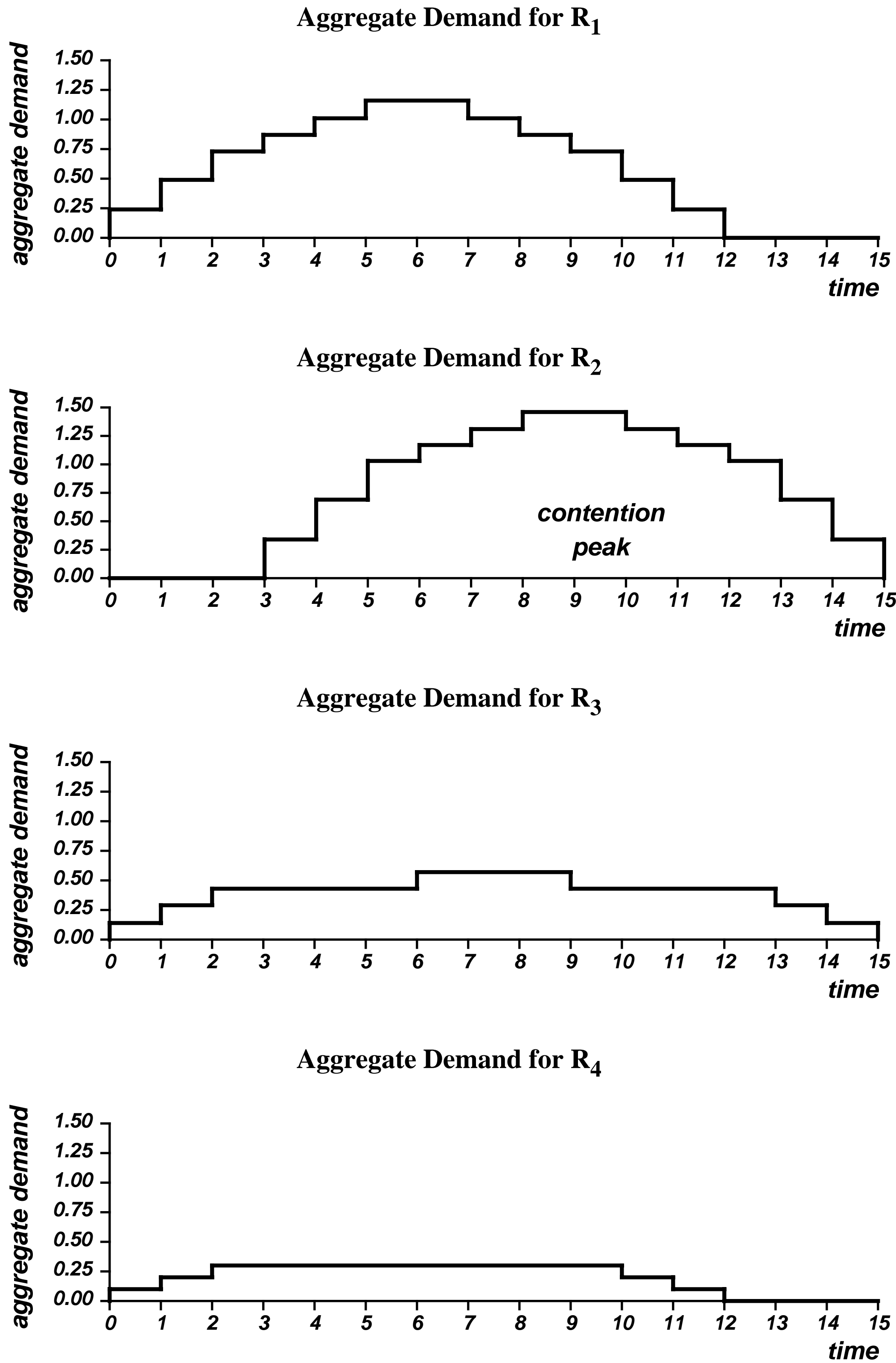

Figure 4-5: Aggregate demands for the three shared resource R_1, R_2, R_3 and the local resource R_4.

Therefore they both select time interval [8,11] as being their most contended one[10].

A second step to picking the activity to schedule next is for each agent to pick its activity with the highest contribution (i.e. highest criticality) to the aggregate demand for that resource/time interval. A higher demand contribution of an activity means that the activity is more likely to be involved in a capacity constraint conflict. In the example, agent α picks activity $A_2^{1\alpha}$ as its most critical activity, since it is the one (among its two activities contending for R_2 that relies most on the possession of that time interval (Figure 4-6) (i.e. the contribution of $A_2^{1\alpha}$ to the demand peak is larger than that of $A_2^{2\alpha}$). Similarly agent β picks $A_3^{1\beta}$ as its most critical activity.

4.2. Value Ordering Scheduling Heuristic

Once an agent has selected the activity to schedule next, it must decide which reservation to assign to that activity. Here several strategies can be considered. One type of value ordering heuristics is a least constraining one. The extremely small number of feasible solutions to a scheduling problem compared to the total number of schedules (including infeasible ones) that one can possibly generate is what has made least constraining value ordering heuristics so attractive. Agents using such heuristics attempt to select the reservation that is the least likely to cause constraint conflicts with reservations of other agents. In other words an agent will select the reservation that will be the least constraining both to itself and to other agents. This heuristic results in *altruistic* behavior on the part of the agent.[11] Because agents in the decentralized case schedule in an asynchronous fashion, and because of the high cost of backtracking in such distributed systems, we expect a higher need for least constraining behavior in a distributed scheduling environment. LCV is a least constraining value ordering heuristic where every reservation for an activity A_k, is rated according to the probability that it would not conflict with another activity's reservation, if one were to first schedule all the other remaining activities. This probability is approximated in our model by the ratio $\frac{D_{kij}(\tau)}{D_{R_{kij}}^{aggr}(\tau)}$, where $D_{kij}(\tau)$ is the selected activity's individual demand on each remaining available time interval (equal to the activity's duration), and $D_{R_{kij}}^{aggr}(\tau)$, is the systemwide aggregate demand for that time interval. The reservation with the largest such probability is interpreted as the least constraining one and is selected for making a reservation.

---

[10]The agents only consider time intervals of duration equal to the average duration of the activities requiring the resource, 3 time units in this example. Two time intervals actually qualify as most contended: [7,10] and [8,11]. We just assume that the agents both pick [8,11].

[11]There exist a variety of value ordering heuristics, such as the greedy Value Ordering Strategy (GV) where an agent can select reservations based solely on its local preferences, i.e. irrespective of its own future needs as well as those of other agents. In addition, there exist value ordering strategies that are intermediate between 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 that reservation will result in backtracking (either locally or for another agent). 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. Experiments in centralized scheduling [Sadeh 90, Sadeh 91] indicate that LCV-type heuristics are best at minimizing search, but usually result in poor schedules since reservations are selected irrespective of their contribution to the objective function. Value ordering heuristics of the greedy type usually produce significantly better schedules, but result in extra backtracking (i.e. search takes longer).

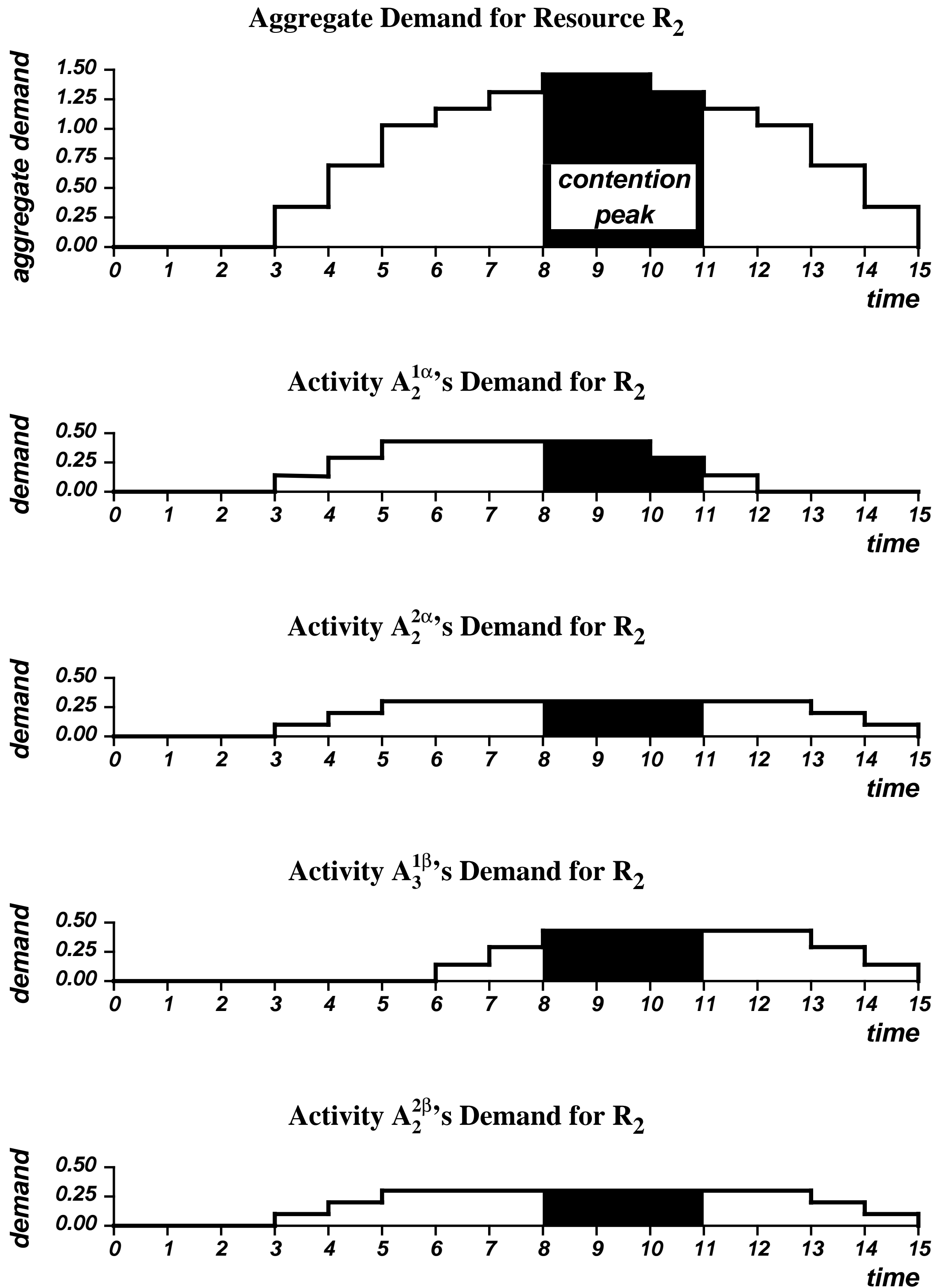

Figure 4-6: Activity ordering by agent α and β.

Let us illustrate the calculation of the LCV heuristic in our two-agent example. As has been mentioned before, using the variable ordering heuristic, agent α has selected activity $A_2^{1\alpha}$ and agent β has selected activity $A_3^{1\beta}$. Now each agent will use the LCV heuristic calculation to determine the start time for its activity on resource R_2. The possible start times for activity $A_2^{1\alpha}$ are 3, 4, 5, 6, 7, 8, 9. The probabilities that a reservation made for each of these start times

results in capacity conflicts for each of these start times are correspondingly 0.44, 0.42, 0.37, 0.33, 0.29, 0.28, and 0.26. We illustrate the calculation for start time 3: The activity demand over the interval {3, 6} (since the duration of the activity is 3 interval units) is given by 1/7+2/7+3/7=0.85. The system aggregate demand over the same interval is given by 0.30+0.60+1.00=1.90 (the values can be read directly from the two upper graphs of figure 4-6). The ratio 0.85/1.90=0.44. Since 0.44 is the largest of the probabilities thusly calculated, 3 is picked by agent α as the best start time for activity $A_2^{1\alpha}$. on resource R_2. Similarly, since the probabilities for the possible start times 6, 7, 8, 9, 10, 11, 12 of activity $A_3^{1\beta}$ of agent g[b] are 0.21, 0.29, 0.33, 0.35, 0.40, 0.44, and 0.46, agent β selects time 12 as the least constraining start time for $A_3^{1\beta}$. The least constraining value ordering heuristic works as expected: the agents end up selecting the non-overlapping intervals {3, 6} and {12, 15}. Notice that all this is done without agent α knowing about the 2 activities of agent β, and without β knowing about those of agent α: the agents only exchanged their total agent demands, not their individual activity demands. In the LCV calculations each agent uses only its own selected activity's demand and the systemwide aggregate activity and no other knowledge.

4.3. Distributed Asynchronous Backjumping

As was discussed earlier, search is directed by variable and value ordering heuristics which select a resource, activity and time interval to schedule at each state. Assuming no other agent has reserved the resource interval, a reservation is made.

Although a resource has been allocated successfully to one activity at each state, it is still possible that a scheduling decision will prevent one or more still unscheduled activities from being scheduled within their start- and due-date constraints, either directly or indirectly. For example, a scheduling decision can directly prevent another activity from being scheduled if it reserves a resource during the only interval over which the unscheduled activity can use the resource. Relatedly, a decision can indirectly prevent another activity from being scheduled because of capacity and precedence constraints among activities. For example, the time interval for which an activity is scheduled can result in narrowing the range of remaining time intervals during which activities following the scheduled activity in a process plan can be scheduled. If the remaining time intervals for these latter activities are ones during which their required resources are unavailable, these latter activities cannot be scheduled.

Although temporal propagation is not able to detect all the indirect effects of a scheduling decision at each state, it is possible to detect t cases in which a scheduling decision makes it impossible for at least one remaining activity to be scheduled. We call this process feasibility checking and it is performed at every state, after a reservation has been made. Feasibility checking consists of: (a) time bound propagation based on activity precedence, and (b) determining whether each activity has at least one interval during which its required resource is available.

If feasibility checking is successful, the scheduling process continues to the next decision state. If it fails (for example assume tha feasibility checking failed for start-time 20 for activity C in figure 4-7), in the standard centralized CSP chronological backtracking would occur[12].

[12]In the figure the possible start-times for an activity, instead of being associated with new states, are denoted by lists appended to a state for better readability.

Chronological backtracking involves the undoing of the last decision made by the system and substituting a different one. In a centralized system, this usually involves considering a different time interval for reserving a resource for the activity scheduled in the last state (activity C in figure 4-7). In the interests of readability, the figure depicts the various possible time intervals for scheduling an activity, just by the start time. When all remaining available time intervals (50, and 80) have been attempted for scheduling C and have failed feasibility checking, the backtracking process will undo the reservation (start-time 10) for the activity and resource scheduled in the **previous** state (activity B in figure 4-7) and attempt an alternative interval (start-time 30) for the previous state (state B). If activity B can be successfully scheduled at another time interval, search will "return" to attempt every possible time interval (including retrying the previously unsuccessful start-time 20) for the now unscheduled activity (activity C). If activity B cannot be successfully scheduled at any other time interval, then the process considers an alternative interval for activity A, then "returns" to consider reservations for the now unscheduled activity B and so on. As a result, this process of making a reservation and feasibility checking is incremental and simultaneously tests the effects of all previous reservations made by an agent on activities remaining to be scheduled.

Figure 4-7: Partial State Space Search

In the multi-agent system, other agents can make reservations throughout an agent's search in an asynchronous manner, making it difficult for the agent to determine which set of previous reservations were responsible for a constraint violation when it is eventually detected. The task facing the agent at this point (when a violation is detected) is to find the last set of its own reservations which, together with those made by other agents, does not violate constraints. In the multi-agent case, suppose a reservation has been made by another agent since the last time our agent has performed feasibility checking. Then, after making a reservation, our agent performs feasibility checking and it fails. Chronological backtracking is no longer efficient because it **assumes that the combination of all previous activity reservations prior to the most recent reservation is still feasible**. In fact, it may no longer be the case that all previous reservations would pass the feasibility test, since other agents' reservations must be considered too.

Therefore, the question is which subset of an agent's previous reservations, combined with the other agents' reservations will still produce a feasible state (one in which it is feasible that the remaining unscheduled activities can be scheduled).

Although chronological backtracking will eventually recognize this subset (if one exists) by trying every possible time interval for each previous activity-state and performing feasibility checking after each, the computational cost is enormous (exponential in the number of activities searched). To reduce the computational cost of backtracking, distributed asynchronous backjumping has been developed.

The backjumping process is as follows:

When infeasibility of a reservation is detected, prior to checking alternative reservations for this activity (and, if necessary for previously scheduled activities), an agent undoes the current infeasible reservation and tests to see whether the reservation of the immediately previously scheduled activity along with other agents' reservations remains feasible.

- If yes, then another reservation for the current activity is checked.
- If not, the process interleaves undoing of previous activity reservations with feasibility checking until a combination of previously made reservations of the agent along with other agents' reservations is feasible.

To compare, consider again backtracking in figure 4-7: notice that before undoing activity C, every remaining time interval is attempted for activity B. Similarly, before activity A's reservation is tested for feasibility, every time interval of B must be tried (in combination with every time interval of C). In contrast, consider the backjumping procedure in figure 4-7. If the current reservation of activity C (start-time 20) is deemed infeasible (assuming other agents have made reservations since the last time feasibility testing was performed), the reservation is undone and, instead of trying alternative time intervals for C, backjumping performs feasibility testing on the remaining (excluding considerations of reservations for C) set of already made reservations (i.e., B and A's current reservations at start-times 10 and 60, along with the reservations made by other agents). If this feasibility check fails, then B's reservation at start-time 10 is undone and the process is repeated (i.e. now only A's reservation at start-time 60 along with the reservations made by other agents are considered). Finally, if A's reservation is feasible, search will resume with a different time interval on B (atart-time 30). In this case, backjumping has identified the first time interval tried for activity B as the one which produced the infeasible state. This was accomplished in just three tests. In contrast, it takes backtracking eight test to identify this situation. If A's reservation is infeasible, then an alternate time interval is considered for A and the process resumes. If all of the agent's reservations were undone and still feasibility checking failed, it means that the reservations made by other agents have forced the agent to an infeasible scheduling situation[13].

---

[13]This has been observed in some of our experiments where a particular decomposition of orders among agents results in a subset of the agents finishing with no backjumping whereas the remaining ones cannot find a solution.

5. The Communication Protocol

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[14]. 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)[15].

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.

---

[14]Using multiple monitoring agents distributes the monitoring responsibility, thus avoiding the inefficiency (in terms of congestion and potential catastrophic failures) that a single bureaucratic monitoring agent would engender.

[15]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.

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 (as specified in a table of monitoring agent) informing it that it will be using shared resources.

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

III. Each agent determines whether its new demand profiles differ significantly from the ones it sent previously for shared resources. If its demand has changed, an agent will send it to the monitoring agent.

IV. The monitoring agent combines all *agent demands* when they are received and communicates the *aggregate demand* to all agents which share the resource[16].

V. Each agent uses the most recent aggregate demand it has received to find 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 communicates this reservation request to the resource's monitoring agent and awaits a response.

VI. The monitoring agent, upon receiving these reservation requests, checks the resource calendar for resource availability. There are two cases:

1. If the resource is available for the requested time interval, 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 the resource had already been reserved for the requested interval, the request is denied. The agent whose request was denied will then attempt to substitute another reservation, if any others are feasible, or otherwise perform backjumping.

VII. Upon receipt of a message indicating its request was granted, an agent will perform consistency checking to determine whether any constraint violations have occurred. If none are detected, the agent proceeds to step II. Otherwise, backjumping occurs with undoing of reservations until a search state is reached which does not cause constraint violations. Any reservations which were undone during this phase are communicated to the monitor for distribution to other agents. After a consistent state is reached, the agent proceeds to step II.

The system terminates when all activities of all agents have been scheduled Backtracking, with this version of the protocol, is based on the following design decisions: 1) Once an agent has been granted a reservation, this reservation is not automatically undone when some other agent who had to backtrack now needs the reservation. This can lead to situations where one agent solves its local scheduling problem but the other agent cannot due to unresolvable constraint violations. 2) If an agent backtracks, 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 greater opportunity to take advantage of the canceled reservations of the backtracking agent, but it results in less computationally intensive performance.

---

[16]With the exception of the first time demands are exchanged, agents do not wait for aggregate demands to be computed and returned prior to continuing their scheduling operations (although they can postpone further scheduling if desired).

6. Experiments with the multi-agent scheduling system

The goals of our experiments were to determine the feasibility of the texture approach to multi-agent scheduling, as well as to test particular mechanisms and parameters that influence system performance. In particular, our experiments considered:

- the effects of agents' incomplete knowledge of each other's plans (i.e. 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 (i.e. the robustness of these measures with respect to delays in the communication of densities),
- the consequences of asynchronous scheduling (e.g., asynchronous use of variable-ordering strategies) without external coordination.

One of the goals of the experiments was to compare performance of multi-agent and centralized schedulers. The experiments summarized here were created from problems found to be difficult in previous research on centralized scheduling [Sadeh 90] and they reflect system performance with respect to search efficiency rather than schedule optimality. The problems were also selected and distributed across the agents in a way that maximized *resource coupling* within orders and across agents.

All the experiments were repeated with 1, 2, 3, and 4 agents. All experimental problems were selected so that orders could be distributed approximately evenly between the agents, all resources were shared by the agents (high inter-agent resource coupling), every order used all resources, and problems ranged from 40 - 100 activities. Over 150 experiments were run in order to vary several properties of each problem. In each case, the dependent variable was the efficiency with which the scheduling system found a solution. This was 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 allowed comparing, for example, a 40-activity 1-agent problem to a pair of 20-activity problems solved simultaneously by 2 agents.

Problem versions differed in several ways. First, to establish a baseline, we created a 1-agent system, which was similar to the multi-agent system in every way, except that the 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 (e.g. the aggregate did not include detailed information about the number of activities which contributed to the densities). Furthermore, it was possible 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 multi-agent system.

The timing of agent communication of their changed densities was determined by the following heuristic: in the MINIMUM delay condition, a single reservation on any resource by any agent initiated the exchange of densities for all resources; in the INCREASED delay conditions, densities were exchanged for each resource independently, whenever N reservations were made on it, where N = 1, 3, and 5. This provided a way to observe the effects of wide ranges in communication frequency (and hence the effect of information obsolescence) in the system.

Another version of the 1-agent system was created which used a *semi-random* (see Figure 6-1) version of the variable-ordering heuristic. The goal was to isolate and assess the effects of less accurate variable ordering that might occur in a multi-agent system. Recall that variable ordering is performed asynchronously in parallel in a multi-agent system (each agent selects the best activity to schedule from its subset of all activities which require a critical resource). Agents do not coordinate the selection of activities to schedule to ensure that the globally most critical ones are scheduled first. As a result, variable ordering is probably less effective than in a 1-agent system. The semi-random heuristic still selects activities to schedule from those which require the most critical resource/time-interval (which narrows the selection to a maximum of 20% of the activities in these problems). However, it then randomly selects from this subset, instead of selecting the activity with the greatest demand for the critical resource. Finally, two scheduler versions were created to compare the use of backtracking and backjumping search techniques.

The experimental results are presented in Figures 6-1 and 6-2 for representative groups of 40-activity and 100-activity experiments respectively.

Figure 6-1: Experimental Results of 40-activity experiments

The first important observation is that the use of texture measures was sufficient to allow near perfect performance when the texture information was updated frequently (MINIMUM Delay conditions). Thus, despite the incompleteness of information available in the various versions (different number of agents) of the multi-agent system, texture measures provide satisfactory summarizations. Second, as expected, performance of the multi-agent system does deteriorate as the communication of changing texture information is delayed. Since current texture information is used to perform both variable and value ordering, it is likely that both these processes deteriorate.

The effect of delaying communication/computation of demand densities is greater for all versions of the multi-agent than the 1-agent system. This interaction may reflect the compensatory relation between variable and value ordering. In the multi-agent case, variable ordering may not be as effective because variables are chosen asynchronously. When texture measures are renewed frequently, there is valid information available to compensate for the poorer variable ordering by selecting effective values for variables. However, when density

Figure 6-2: Experimental Results of 100-activity experiments

communication is delayed, value ordering is also weakened and the performance declines. In the 1-agent system, variable-ordering is more effective, so the delay does not hurt performance as much. This view is supported by the results in the 1-agent semi-random variable-ordering condition, where the effects of partially disabling variable ordering accelerates with increasing delay, as it does in the multi-agent case. Note that multi-agent performance is still better than the semi-random condition in all cases, suggesting that variable ordering strategy is robust with respect to the conditions of the multi-agent environment (incomplete, changeable information and asynchronous behavior without external coordination).

We should also note that the semi-random condition is still highly selective relative to completely random variable ordering (in that only activities which use the most critical resource/time interval are considered). In fact, we found that random variable ordering resulted in terrible performance, even in the 1-agent case. Solutions were not found in over 500 states. As expected, the use of a backjumping strategy substantially reduced the search in the multi-agent versions of the system. Performance using a chronological backtracking strategy was highly variable and degraded exponentially with delayed communication (searches exceeded 300 states when communication was delayed to after three reservations).

Another interesting observation from our experimentation was that different distributions of orders to agents produced different results in that one decompositions might result in feasible solutions by all agents whereas another might result in some of the agents finishing the scheduling of their assigned orders whereas the rest did not finish. Although we have not systematically performed experimentation with characteristics of the various decompositions, we hypothesize that agents with "easier" assignments manage to obtain the good reservations for their activities, causing solution infeasibilities for the rest of the agents. Studying the effect of different decompositions and their characteristics is one of the subjects of future research.

7. Concluding Remarks

In this paper we presented the Distributed Constraint Heuristic Search model and presented mechanisms to guide concurrent, asynchronous distributed search. 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. These textures 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 make inferences about intentions of other agents. We have presented two types of textures, their operationalization into variable and value ordering heuristics and their use in distributed problem solving. A communication protocol that enables the agents to coordinate their decisions has been presented. We have developed a variant of dependency-directed backtracking, asynchronous backjumping, that substantially reduces backtracking costs.

Our investigation is conducted in the domain of distributed job-shop scheduling. A model of distributed job shop scheduling as an instance of DCHS has been developed and 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 subsections 4.1 and 4.2. The experiments are designed to give results concerning the role of the heuristics in achieving search efficiency in distributed asynchronous DCHS under conditions of incomplete and rapidly changing information. The testbed is implemented in a decentralized manner in KnowledgeCraft running on top of Common Lisp, and can be run on a set of MICROVAX's 3200.

Future work will concentrate on (a) continued experimentation to identify the class of distributed CSP's that can be solved efficiently by our algorithm, and (b) developing strategies for adaptive use of various value ordering heuristics depending on where each agent is in its search when it has to make a reservation. In addition, we plan to develop negotiation protocols that will enable agents that contend for the same resource/time-interval (i.e. reservation) to negotiate over who should actually be granted the reservation. Criteria for making such a decision will include the relative priorities of the orders that the two contending agents have to schedule, how far each agent is in its scheduling, and whether an agent has access to a substitutable resource, or relies solely on using the contended-for resource.

References

[Baker 74] K.R. Baker.
Introduction to Sequencing and Scheduling.
Wiley, 1974.

[Bitner 75] J.R. Bitner and E.M. Reingold.
Backtrack Programming Techniques.
Communications of the ACM 18(11):651-655, 1975.

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

[Conry 88] Conry, S., Meyer, R., and Leser, V.
Multistage Negotiation in Distributed Planning.
In Bond, A. and Gasser, L (editor), *Readings in Distributed AI*. Morgan Kaufmann, 1988.

[Dechter 89] Rina Dechter.
Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition.
Artificial Intelligence 41:273-312, 1989.

[deKleer 87] de Kleer, J.
Dependency-Directed Backtracking.
In Shapiro, S. (editor), *Encyclopedia of Artificial Intelligence*. John Wiley and Sons, New York, N. Y., 1987.

[Durfee 85] E. Durfee, V. Lesser, and D. Corkill.
Coherent Cooperation Among Communicating Problem Solvers.
Technical Report, Department of Computer Science and Information Science, University of Massachusetts - Amherst, Massachusetts 01003, September, 1985.

[Durfee 87] Durfee, E.H.
A Unified Approach to Dynamic Coordination: Plannign Actions and Interactions in a Distributed Problem Solving Network.
PhD thesis, COINS, University of Massachusetts, 1987.

[Fox 81] Fox, M.S.
An Organizational View of Distributed Systems.
IEEE Transactions on Systems, Man and Cybernetics SMC-11:70-80, 1981.

[Fox 83] Fox, M.S.
Constraint-Directed Search: A Case Study of Job Shop Scheduling.
PhD thesis, Computer Science Department, Carnegie-Mellon University, 1983.

[Fox 89] Mark S. Fox, Norman Sadeh, and Can Baykan.
Constrained Heuristic Search.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 309-315. 1989.

[Garey 79] M.R. Garey and D.S. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman and Co., 1979.

[Gaschnig 79] John Gaschnig.
Performance Measurement and Analysis of Certain Search Algorithms.
Technical Report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1979.

[Golomb 65] Solomon W. Golomb and Leonard D. Baumert.
Backtrack Programming.
Journal of the Association for Computing Machinery 12(4):516-524, 1965.

[Graves 81] Graves, S.C.
A Review of Production Scheduling.
Operations Research 29(4):646-675, July-August, 1981.

[Ishida 90] Ishida, T., Yokoo, M, and Gasser, L.
An Organizational Approach to Adaptive Production Systems.
In *Proceedings of AAAI-90, Boston, Mass..* 1990.

[Keng 89] Naiping Keng and David Y.Y. Yun.
A Planning/Scheduling Methodology for the Constrained Resource Problem.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 998-1003. 1989.

[Kuwabara 89] Kuwabara, K., and Lesser, V.
Extended Protocol for Multi-Stage Negotiation.
In *Proceedings of the 9th International Workshop on DAI*. 1989.

[LePape&Smith 87]
LePape, C. and S.F. Smith.
Management of Temporal Constraints for Factory Scheduling.
In C. Rolland, M. Leonard, and F. Bodart (editors), *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.

[Lesser 90] Lesser, V.
An Overview of DAI: Viewing Distributed AI as Distributed Search.
Journal of the Japanese Society of Artificial Intelligence 5(4), 1990.

[Mackworth 85] Mackworth, A.K., and Freuder, E.C.
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25(1):65-74, 1985.

[Nadel 88] Bernard Nadel.
Tree Search and Arc Consistency in Constraint Satisfaction Algorithms.
Search in Articial Intelligence.
In L. Kanal and V. Kumar,
Springer-Verlag, 1988.

[Parunak 86] Parunak, H.V., P.W. Lozo, R. Judd, B.W. Irish.
A Distributed Heuristic Strategy for Material Transportation.
In *Proceedings 1986 Conference on Intelligent Systems and Machines*. Rochester, Michigan, 1986.

[Rinnooy Kan 76] A.H.G. Rinnooy Kan.
Machine Scheduling Problems: Classification, complexity, and computations.
PhD thesis, University of Amsterdam, 1976.

[Sadeh 90] Norman Sadeh, and Mark S. Fox.
Variable and Value Ordering Heuristics for Activity-based Job-shop Scheduling.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management, Hilton Head Island, S.C.*, pages 134-144. 1990.

[Sadeh 91] Norman Sadeh.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, 1991.

[Smith et. al. 86] Smith, S.F., 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 7(4), Fall, 1986.

[Smith&Hynynen 87]
Smith, S.F. and J.E. Hynynen.
Integrated Decentralization of Production Management: An Approach for Factory Scheduling.
In *Proceedings ASME Annual Winter Conference: Symposium on Integrated and Intelligent Manufacturing*. Boston, MA, December, 1987.

[Yokoo 90] Yokoo, M., Ishida, T., and Kuwabara, K.
Distributed Constraint Satisfaction for DAI Problems.
In *Proceedings of the 10th International Workshop on DAI, Banderra, Texas*. 1990.

Table of Contents

1. Introduction 1
2. Overview of Constrained Heuristic Search 2
3. Distributed CHS 3
4. Distributed CHS Job-Shop Scheduling 7
  4.1. Variable Ordering Scheduling Heuristic 10
  4.2. Value Ordering Scheduling Heuristic 15
  4.3. Distributed Asynchronous Backjumping 17
5. The Communication Protocol 20
6. Experiments with the multi-agent scheduling system 22
7. Concluding Remarks 25
References 25

List of Figures

Figure 4-1: A simple problem with 2 agents, 4 orders, and 4 resources. 9
Figure 4-2: Building agent α's demand for resource R_2. 11
Figure 4-3: Building agent β's demand for resource R_2. 12
Figure 4-4: Agent and aggregate demands for resource R_2. 13
Figure 4-5: Aggregate demands for the three shared resource R_1, R_2, R_3 and the local resource R_4. 14
Figure 4-6: Activity ordering by agent α and β. 16
Figure 4-7: Partial State Space Search 18
Figure 6-1: Experimental Results of 40-activity experiments 23
Figure 6-2: Experimental Results of 100-activity experiments 24