A Cooperative Repair Method for a Distributed Scheduling System

Daniel E. Neiman and Victor R. Lesser
Computer Science Department
University of Massachusetts
Amherst, MA 01003
dann@cs.umass.edu

Abstract

For some time, we have been studying the issues involved in job-shop scheduling in an environment of cooperative distributed agents, none of which has a complete view of the resources available, or of the tasks to be scheduled. Schedules produced cooperatively by such distributed agents using constraint satisfaction methods are often not optimal because of the inherent asynchronicity of the distributed scheduling process, the bounded rationality of the scheduling agents, and the difficulty in completely integrating meta-level heuristics into an agent's local scheduling processes. This paper describes a modification to distributed scheduling in which the loosely coupled distributed processing methods are supplemented with a tightly coupled parallel repair process. We explore the implications on the repair process of a distributed environment in which the designer of the repair algorithm must address issues of agent communication and organization. We describe a search algorithm and a set of heuristics for guiding the repair process and present some experimental results in the context of the Distributed ARM, an airline resource scheduling system.

Introduction

For some time, we have been studying the issues involved in job-shop scheduling in an environment of cooperative distributed agents, none of which has a complete view of the resources available, or of the tasks to be scheduled. The focus of this paper is the development of tools for examining the trade-offs between a loosely-coupled distributed scheduling process and a more tightly-coupled parallel search method for constructing multi-step repair plans for situations that would otherwise require costly constraint relaxations.

In a previous work (Neiman *et al.* 1994), we reported on heuristics for coordinating the scheduling processes of multiple cooperative agents. During scheduling, agents regularly transmit abstractions (texture measures) of their current resource capabilities and anticipated resource requirements to remote agents. Agents incorporate this information into their scheduling processes, using it to determine which tasks to schedule next using a most-constrained-variable heuristic, which agents to request resources from, and to select the methods for scheduling a particular resource. Similar heuristics for distributed scheduling systems have been reported by (Yokoo, Ishida, & Kuwabara 1990; Sycara *et al.* 1991; Berry, Choueiry, & Friha 1993; Burke & Prosser 1994). The use of texture measures imposes costs upon the distributed scheduler, both in terms of communication costs and computational overhead for both the transmitter and receiver of the abstractions. Agents performing a scheduling action must determine whether their actions will affect other agents, and which agents would be affected, and transmit information accordingly. Agents must also frequently integrate newly received abstractions into their local planning process. In our micro-opportunistic scheduling architecture (Sadeh 1991), new information requires an agent to continually rerate its scheduling goals in order to focus on the most critical scheduling tasks. This constant rerating of goals causes the agent to spend considerable amount of time on *meta-control* activities, reducing the amount of time available for actually performing scheduling activities. Despite this effort, in most cases, the solution produced via distributed scheduling will be of lower quality than would be achieved by a centralized agent solving the same problem. Orders will be later and more constraints will be violated.[1]

The reasons for the reduction in schedule quality are manifold, but can primarily be attributed to three characteristics of distributed scheduling systems: asynchronicity, bounded rationality, and a bias towards local problem-solving. Asynchronicity exists in distributed systems because the state of the problem-

[1]A natural question at this point is "Why not use a centralized scheduler?". Our answer is that the distributed nature of the scheduling process is forced upon us by the nature of the real world. Resources tend to be physical entities owned or managed by discrete agents that have proprietary knowledge about the capabilities of the resources and that have sole scheduling privileges for those resources.

solving world changes while an agent is deliberating. Other agents are simultaneously processing, transmitting messages, and consuming resources. Agents must therefore continually make decisions based on out-of-date information. The bounded-rationality of agents limits the number of solutions that an agent can feasibly explore. The scheduling process is known to be NP-hard in many cases. For this reason, scheduling agents seek *satisficing* rather than optimal solutions. The amount of backtracking and the methods available to agents for resolving constraint violations are deliberately restricted in order to produce results within a reasonable period of time. In a distributed problem-solver, the area of the solution space that can be explored is even more limited because of the communication overhead incurred. Finally, agents tend to have a bias towards solving problems based on their own local view. Although global information may be available, it is difficult to incorporate it at all levels of the local problem-solving process.

These observations have led us to the conclusion that distributed problem solving episodes should have two components: a fully asynchronous phase, characterized by loose coordination between agents in which a satisficing solution is developed, followed by a more tightly coordinated, synchronous phase in which agents cooperate to increase the quality of problematic aspects of the solution. This synchronous phase can either take the form of a post-processing repair episode in which agents attempt to improve the quality of their schedules by identifying and repairing those resource assignments that violate constraints or the form of a special scheduling method during the formation of the original process plan in which agents attempt to proactively avoid constraint relaxations. Because the effects of constraint relaxations tend to propagate throughout a schedule and are difficult to "unwind", we have chosen to implement the latter approach and attempt to minimize the relaxation of hard constraints during scheduling.

Iterative schedule repair algorithms have been studied by many researchers, including (Miyashita & Sycara 1994; Zweben *et al.* 1994). In a centralized system, repairs are usually required when jobs to be scheduled are modified, added, or removed, or when resources break down or become otherwise unavailable. Several systems, however, take an approach similar to ours in that they first develop an almost correct schedule and then incrementally repair constraint violations until a satisfactory schedule is produced (Minton *et al.* 1990; Johnston & Minton 1994).

In the following sections, we discuss how a parallel repair algorithm has been implemented as a separated scheduling method in a distributed scheduling environment. We first describe our experimental system, the Distributed Dynamic Scheduling System (Distributed DSS). We then discuss the issues involved in distributed scheduling and present our heuristics for guiding the repair process. Finally, we describe our experimental setup and present some observations on the performance of the distributed repair system.

Overview: The Distributed Dynamic Scheduling System

In order to test our approach to solving distributed resource-constrained scheduling problems, we have designed a distributed version of a reactive, knowledge-based scheduling system called DSS (the Dynamic Scheduling System) (Hildum 1994). DSS provides a foundation for representing a wide variety of real-world scheduling problems. Its flexible scheduling approach is capable of reactively producing quality schedules within dynamic environments that exhibit unpredictable resource and order behavior. The Distributed DSS was constructed by partitioning the resource and order data structures and assigning separate resource and order partitions to each agent. A communication facility was added to allow agents to transmit resource requests and meta-level control information. The primary form of cooperation in the distributed system is the lending of resources to satisfy a remote agent's order.

The Distributed Airport Resource Management System

The Distributed Airport Resource Management System testbed was constructed using DIS-DSS to study the roles of coordination and negotiation in a distributed problem-solver. DIS-ARM solves distributed Airline Ground Service Scheduling (AGSS) problems where the function of each scheduling agent is to ensure that each flight for which it is responsible receives the ground servicing (gate assignment, baggage handling, catering, fueling, etc.) that it requires in time to meet its arrival and departure deadlines. The supplying of a resource is a multi-step task consisting of setup, travel, and servicing actions. Each resource task is a subtask of the airplane servicing supertask. There is considerable parallelism in the task structure: many tasks can be done simultaneously, for example, planes can be serviced while baggage is being unloaded and loaded. The choice of certain resource assignments will often constrain the start and end times of other tasks. For example, selection of a specific arrival gate for a plane may limit the choice of servicing vehicles due to transit time from their previous servicing locations and may limit refueling options due to the presence or lack of underground fuel tanks at that gate. For this reason, all resources of a specific type can not be considered interchangeable in the AGSS domain. Only the agent that owns the resource can identify all the current constraints on that resource and decide whether or not it can be allocated to meet a specific demand.

Orders and Resources

For the purposes of this discussion, it is necessary to make a clear distinction between two kinds of state that are maintained by each agent; the agent's *orders* and the agent's *resources*.

Orders Orders represent the jobs to be scheduled by each agent. Each order triggers the generation of a task network. The task network is a somewhat complex data structure that explicitly represents the temporal constraints between each task in an order. For each resource-requiring task, the agent creates a *service goal*. This goal explicitly represents the legal resources, the earliest start time (EST), and latest finish time (LFT) of the servicing task. A *reservation* represents an assignment of a resource to a service goal for an interval (t_b, t_e) which includes setup, reset, service, and travel times (if the resource is movable). As reservations are made for related tasks within the network, a goal's time window may decrease. The DSS explicitly represents *slack* time for each task within the task-network. This is the amount of time that the task may be shifted to the left (earlier) or right (later) in the schedule without requiring the rescheduling of any other task in the order and without violating any hard constraints. Slack is, therefore, a measure of flexibility in the *task network*.

Figure 1 shows the resource assignments for a single order in a schedule produced within our Distributed Airport Resource Manager testbed using a three agent configuration. The deadline for this order was missed by a significant amount, 16 minutes. The immediate cause of the delay is a late assignment of the Load-Baggage task, however, further diagnosis indicates that this late reservation was forced by the right-shifting of the prequisite task, UnLoad-Baggage. Examination of the scheduling history for this task enables us to plausibly attribute the delay to the asynchronous nature of the scheduling process. During the scheduling process, scheduling agent 0 was requested to provide a baggage truck resource and did so, preempting a previous reservation for the UnLoad-Baggage task for flight 458. This preemption was based on the perception that sufficient global resources remained to allow both the UnLoad-Baggage and Load-Baggage tasks to be scheduled. This judgement did not take into account reservations made and requests generated after the receipt of the most recent texture measures. By the time agent 0 actually scheduled the UnLoad-Baggage task, the only available reservation was too late to allow a reservation for the Load-Baggage task to be made without causing it to be right-shifted, violating the hard constraint represented by the order's due date.

Resources In the Distributed DSS, there may be several resources of the same type, or several resource types that are capable of carrying out the same task. Because the DSS represents resource location and travel time (and, for some applications, resource capacity) resources of the same type are not necessarily interchangeable. For each resource, the scheduling agent maintains a data structure representing reservations and un-used time periods.

Figure 2 shows the reservations for the baggage truck resources for all orders in the schedule of which the order discussed previously is a part. It can be readily seen that there is no simple reassignment for the UB458 task that would allow both it and the LB458 task to meet their deadlines. However, there is considerable unconsolidated unused time, leading us to believe that some plan for reassigning resources might result in an acceptable set of reservations.

Distributed Search Issues In a distributed environment, agents are not able to produce complete plans themselves. Agents cannot generate plans involving resources belonging to another agent and must therefore request information or planning activities from that agent. The presence of loaned and/or borrowed resources also complicates the repair process. A reservation promised to another agent cannot be shifted either left or right without first generating a request to ensure that there is sufficient slack in that agent's tasks. Conversely, an agent with a borrowed resource does not have the information locally to determine whether it can be shifted within the remote agent's resources. Agents must therefore cooperate to generate repair plans. We can structure our algorithms as a parallel search process and investigate the normal issues that arise in any distributed system, namely questions of organization, commitment of effort, information transfer, and reasoning under uncertainty.

The Synchronous Schedule Repair Method

In DSS, the process of securing a resource is achieved through a series of increasingly costly methods: assignment, preemption, and right shifting. These correspond roughly to request satisfaction, backtracking, and constraint relaxation. Preemption is a conservative form of backtracking in which a single existing reservation is preempted in favor of a more constrained task. Right shifting satisfies otherwise intractable requests by shifting the time interval of the reservation downstream (later) until a suitable resource becomes available. Because this method relaxes the latest finish time constraint, it has the potential to seriously decrease the quality of a solution. In the AGSS domain, for example, right shifting a reservation will result in late aircraft departures. Each of these resource securing methods is local. At any point during the process of securing a resource, an agent may choose to generate a remote request to another agent in an attempt to secure the resource remotely. The remote agent can then use any of the above methods in an attempt to secure the resource for the requesting agent (see Figure 3).

Each of the local methods is implemented as a sin-

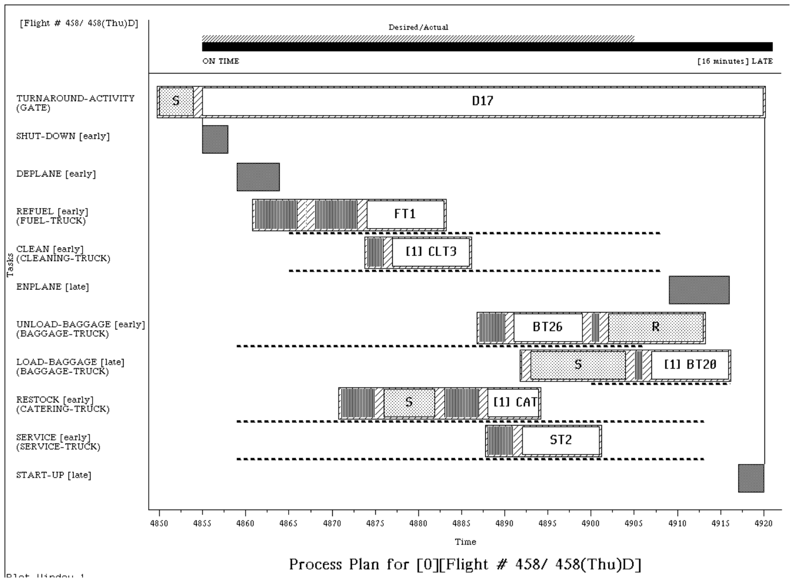

Figure 1: The resource assignments for one order in the Distributed ARM. Setup (S) and reset (R) times are shown in gray, servicing times are shown in white, travel times are shown by vertical striped segments. The dotted lines below each servicing task shows the available slack for that task. A bracketed number indicates the remote agent providing the resource. According to the Desired/Actual legend at the top of the figure, the load baggage task for this order has been right-shifted beyond its LFT (latest finish time), violating a hard deadline constraint.

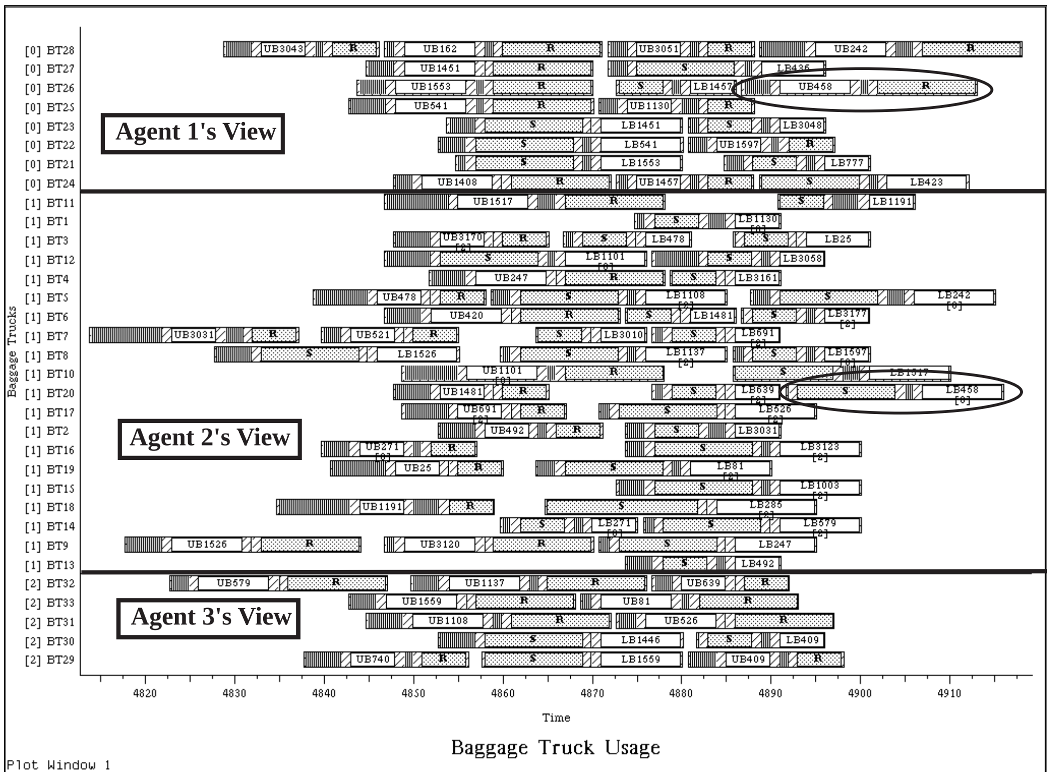

Figure 2: The baggage truck assignments for all orders in a schedule. Blank areas represent unused resource capacity, labeled white areas represent servicing time, gray areas represent setup (S) and reset (R) time. The agent owning the resource is indicated by the number on the left vertical axis, while a bracketed number underneath the task name indicates the agent to which the resource has been lent.

gle *knowledge source* that is executed by a single agent in order to secure a single resource for an order. Because only the agent owning a resource can assign it to an order, agents can construct schedules asynchronously without fear that a resource allocation will interfere with another agent's processing. In contrast, the schedule repair method constructs plans consisting of a sequence of resource cancellations and reassignments. From a distributed processing viewpoint, the replanning process can be considered as a multi-linked negotiation, with each scheduling agent committing to its role in the developing plan. If any other scheduling methods were executed during the course of this replanning, the assignment of resources might cause the repair plan to become infeasible. For this reason, replanning is performed synchronously – agents are allowed to work only on the replanning task under the control of a single initiating agent.

The schedule repair method is invoked whenever a scheduling goal can only otherwise be satisfied by performing a constraint relaxation operation (right-shift method) (see Figure 3). All agents in the system suspend their current problem-solving activities (after completing their current knowledge source invocation and processing all outstanding messages) and collaboratively search for some reassignment of resources that would allow the current scheduling goal to be satisfied without a constraint relaxation. Although the synchronous replanning method may prove to be computationally costly, it is guaranteed not to violate any hard constraints of the existing partial solution – reservations may be swapped, but will not be positioned out of their legal time windows. One possible future direction, however, is to modify the parallel search to look for the least costly possible set of constraint violations when faced with an intractable set of scheduling requirements.

The Distributed Search Process

The actual repair process is implemented as a best-first search. The operators applicable to a given search state are represented as repair goals, specifying the reservation to be moved, the method to be used, and the desired time window, shift amount, and shift direction. States in the search space are represented by repair plans, which specify reservations for the target goals. Each plan has attached data structures describing the changes in resources and task assignments that would be accomplished by executing the steps of the plan. Each plan maintains an OR tree of repair goals representing prerequisites that must be achieved in order for the plan to be executable. As repair goals are generated, they are broadcast to all agents that could potentially satisfy them. Examples of repair plans and goals can be seen in Figure 4.

Our set of basic repair operators is quite small and by no means exhaustive. The complexity of the replanning process stems from the large number of possible plan instantiations possible for each repair goal. In general, new states are created by either shifting reservations or by assigning tasks to new resources:

- **Shift:** Reservations may be shifted left or right (earlier or later) – possibly violating soft constraints and increasing the cost or lowering the quality of a particular task.
- **Move:** A reservation might be moved to another resource provider, assuming that space exists, or that previous steps in the repair plan have created a slot.
- **Shift Adjacent Blocking Goal:** If a task does not have adjacent slack to permit a desired shift, the scheduler may generate a goal to shift an earlier (or later) reservation in the task's network.

A completed plan is shown in Figure 4.

Rating Repair Goals

In order to reduce the complexity of the search process, we sought a method of abstracting schedules in such a way that agents could reliably estimate whether a given repair goal was feasible, and, if so, which agent would be most likely to be able to carry out the goal. This abstraction uses two measures of schedule flexibility, slack and un-used time.

In the Distributed DSS, each agent represents its resources with a complex data structure, describing the activity and location of the resource during each time period as it services each task. This information is too extensive and specialized to transmit to other agents. Instead, we have abstracted the time and state information for each resource into lists of time intervals. Each interval is marked as USED or UNUSED. In addition, each USED interval (which comprises the entire setup, travel, servicing, and reset times of an activity) is also annotated with its *slack* time. This measure delimits a time region within which a task can be shifted without forcing the rescheduling of any other activity within its supertask. These two measures, unused time, and slack, collectively abstract the flexibility within the system, both in terms of resources and of tasks. These measures are used in several ways in the schedule repair process, both to generate heuristics for the search through sets of possible schedules, and in generating meta-goals.

Heuristics for Scheduling Repair Activities

At the beginning of a repair episode, the agents transmit their abstracted unused/slack statistics to all other agents. This allows agents to determine whether a repair goal is heuristically feasible either locally or remotely.

As each agent generates a repair goal, it is rated and assigned to one of several categories.

1. Goals that appear to be immediately satisfiable. These are goals for which a suitable UNUSED time period seems to be available for some resource.

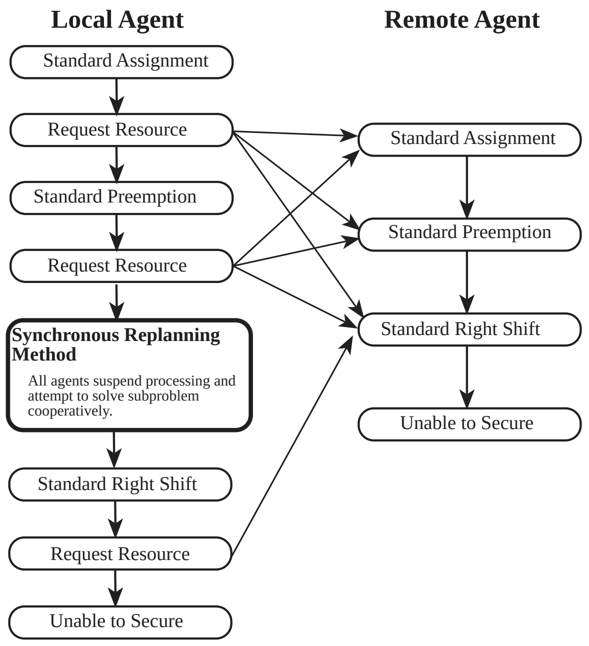

Figure 3: In a distributed scheduling system, determining the method by which a resource will be secured is itself a complex problem. The DSS provides a set of methods of increasing cost for assigning resources. At any time, an agent may decide to generate a request for a resource rather than attempting a local method. Synchronous replanning occurs whenever a (costly) right-shift method is the only remaining possibility for securing a resource.

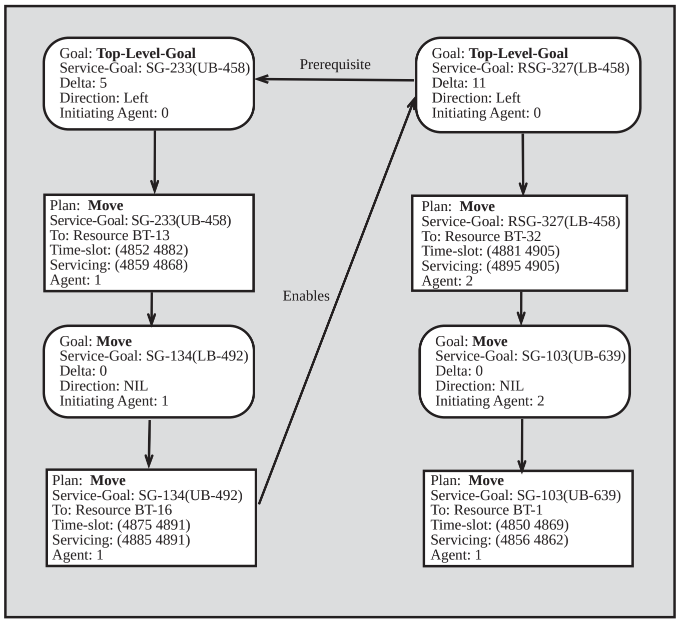

Figure 4: A plan-tree developed by the distributed repair mechanism for the LB458 ARM scenario. The repair process was initiated by Agent 0, who determined that the goal LB458 must be shifted 11 minutes to the left. A prerequisite, developed by the operator, move-blocking-task, is to shift the immediately adjacent upstream task, UB458, to a reservation at least 5 minutes earlier. Agent 1 receives the repair goal and is able to satisfy it by shifting the service goal SG-134 to a new resource, BT-16, and UB-458 to BT-13 at an earlier time. This allows the system to develop a plan for assigning a reservation to the LB458 task within the due date of the Flight 458 order.

2. Goals that appear to be feasible: These are goals for which there exists some resource such that the sum of UNUSED time plus the slack time of adjacent resources is greater or equal to the amount of time required for the reservation. The goal could therefore be potentially satisfied by either a shift or a move operation.
3. Goals that are potentially feasible. These are goals for which the total amount of UNUSED time and slack for all resources are greater than required.
4. Goals that appear not to be feasible. These are goals for which the sum of UNUSED and slack time is less than the amount of time required. In the best case, satisfying this goal would require shifting the left predecessor of some goal in order to increase the amount of slack time in the region. In the worst case, this implies that there is no further flexibility in the schedule and no plan can be developed without relaxing hard constraints.

Within each category, goals are grouped according to heuristics designed to improve the quality of the final solution and the performance of the distributed replanning system. First, goals are ordered to reflect the time-shift preferences of the tasks themselves. For example, tasks with preferences to be scheduled as late as possible are rated highly if they are to be shifted later, and lower if they are to be shifted to an earlier time slot.

The heuristics for rating goals to improve distributed performance vary according to the organizational structure used in problem solving, but essentially, the object is to favor plans that involve the originating agent or plans that can be further developed locally. The first heuristic will minimize the cost of actually implementing the final repair plan, because the majority of the tasks will be able to be carried out by the initiating agent. This also has the benefit of reducing that agent's obligations to other agents. The second heuristic, favoring plans that can be developed locally, reduces the communication overhead.

Experimental Results

The distributed schedule repair method was tested using several order sets developed for a three agent configuration of the DIS-ARM distributed scheduler as reported in (Neiman *et al.* 1994). We chose three order sets displaying considerable tardiness under distributed scheduling despite the use of texture-based scheduling heuristics. For each order set, we generated ten test cases by randomly assigning resources and orders to each of the three agents. The total number of resources provided for these order sets were designed to be minimally sufficient to allow all the jobs to be scheduled – at least one resource type proves to be a bottleneck in each environment. Our results are presented in the table in Figure 1. As measured by total tardiness and number of late orders, the results of the distributed repair process are superior not only to those produced by the distributed scheduler alone, but quite often to the results produced by the centralized scheduler. In the schedules in which repair plans were produced, the number of right shift methods was reduced by 50%. Although the set of scheduling applications explored is not yet large enough to allow us to make sweeping generalizations, we have observed that the UNUSED/slack heuristic is an excellent predictor of whether or not a schedule is repairable. Intervals lacking either tasks with a high degree of slack time or resources with significant unused capacity proved to be intractable for rescheduling purposes. Conversely, the presence of intervals with significant slack or unused times led to the rapid formation of plan trees with very few steps.

Table 1: The results of the distributed repair algorithm compared with schedules developed by both centralized and distributed schedulers for three large (40+ jobs) order sets in the ARM testbed. Results are presented in terms of total minutes past the due date for the entire set of orders followed by the number of tardy orders in parentheses.

Scheduler	Environment		
	0810	1050	1200
Centralized	8(3)	32(9)	47(7)
Distributed	55.4 (5.4)	52 (5.6)	76.9 (6.4)
Distributed with Repair	19 (2.8)	41.2 (4.8)	19.2 (1.7)

Conclusions

This work was motivated largely by our realization that we had reached the point of diminishing returns in attempting to improve the performance of distributed schedulers by increasing the quality of the heuristics and the amount of meta-level information available to the agents. The amount of effort agents were spending generating, communicating, and integrating meta-level information into their processing was becoming increasingly burdensome, and the increase in schedule quality due to exploiting this information had become increasingly difficult to obtain. Our results indicate that an alternative approach may be appropriate for the distributed scheduling domain, namely, to allow agents to develop initial approximations of schedules while coordinating only loosely. A more tightly coupled processing phase can then be performed in which the agents coordinate very closely to avoid costly constraint relaxations caused by lack of coordination between the distributed agents.

Considerable work remains to be done. In particular, we would like to determine whether we can significantly reduce the overhead of texture communications by performing infrequent but computation-intensive and communication-intensive repair operations. This will require empirically measuring the relative cost of the synchronous repair mechanism as compared with the overhead of continuous coordination between agents.

In the near term, we intend to continue testing the synchronous repair method on a wider range of schedules and application domains and to investigate the trade-offs between performing repair as a synchronous scheduling method and as a separate post-processing episode.

Acknowledgments

This work was partly supported by DARPA contract N00014-92-J-1698 and NSF grant IRI-9321324. The content of this paper does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. We would like to thank Dr. David Hildum for his work in developing the centralized and distributed versions of the DSS.

References

Berry, P. M.; Choueiry, B. Y.; and Friha, L. 1993. A distributed approach based on temporal abstractions fr planning scheduling and resource allocation. In *Workshop Notes of the IJCAI-93 Workshop on Knowledge-based Production Planning, Scheduling, and Control*, pp 23–31.

Burke, P., and Prosser, P. 1994. The distributed asynchronous scheduler. In Zweben, M., and Fox, M. S., eds., *Intelligent Scheduling*. Morgan Kaufmann. chapter 11.

Hildum, D. W. 1994. *Flexibility in a Knowledge-Based System for Solving Dynamic Resource-Constrained Scheduling Problems*. Ph.D. Dissertation, Computer Science Dept., University of Massachusetts, Amherst, MA 01003.

Johnston, M. D., and Minton, S. 1994. Analyzing a heuristic strategy for constraint-satisfaction and scheduling. In Zweben, M., and Fox, M. S., eds., *Intelligent Scheduling*. Morgan Kaufmann. chapter 9.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1990. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 17–24.

Miyashita, K., and Sycara, K. 1994. Adaptive case-based control of schedule revision. In Zweben, M., and Fox, M. S., eds., *Intelligent Scheduling*. Morgan Kaufmann. chapter 10.

Neiman, D.; Hildum, D.; Lesser, V.; and Sandholm, T. 1994. Exploiting meta-level information in a distributed scheduling system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 394–400.

Sadeh, N. 1991. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling*. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh PA.

Sycara, K.; Roth, S.; Sadeh, N.; and Fox, M. 1991. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6):1446–1461.

Yokoo, M.; Ishida, T.; and Kuwabara, K. 1990. Distributed constraint satisfaction for DAI problems. In *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence*.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and rescheduling with iterative repair. In Zweben, M., and Fox, M. S., eds., *Intelligent Scheduling*. Morgan Kaufmann. chapter 8.