Combining Coordination and Repair Strategies in a Distributed Scheduling System

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

December 16, 1997

Abstract

The Distributed DSS is a testbed for exploring issues involved 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. In a previous work, we introduced a tightly coupled parallel repair process into the distributed scheduling process. In this repair process, agents with locally unresolvable schedule conflicts were allowed to initiate a global search process for a set of resource assignments that would allow constraint relaxation to be avoided. Because all agents took part in this global search and performed no other concurrent scheduling tasks, we termed this process *synchronous repair.*

The success of the synchronous repair method led us to speculate that we could reduce the overhead of exchanging meta-level information by omitting to coordinate and by allowing the agents to resolve conflicts by cooperatively repairing schedules. In this paper, we evaluate the performance of a distributed scheduling heuristic incorporating different levels of global knowledge in a distributed scheduling system, with and without the synchronous repair mechanism enabled. We conclude that the repair process is synergistic with an informed scheduling heuristic, that global knowledge leads to better schedules,

and can be incorporated into scheduling heuristics at a relatively low cost.

1 Introduction

The Distributed DSS is a testbed whose purpose is to explore 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. In a previous work [9], we introduced a tightly coupled parallel repair process into the distributed scheduling process. In this repair process, agents with unresolvable schedule conflicts were allowed to initiate a global search process for a set of resource assignments that would allow constraint relaxation to be avoided. Because all agents took part in this global search and performed no other concurrent scheduling tasks, we termed this process *synchronous repair*.

We now explore the implications of the distributed repair process on the need for communication between agents. If we can produce high quality schedules by repairing the results of a loosely coordinated system, can we also produce high quality schedules if agents coordinate seldomly or not at all? Specifically, can we reduce the overhead of exchanging meta-level information by allowing agents to base scheduling decisions solely on local views of constraint tightness, and by allowing the agents to resolve conflicts by cooperatively repairing schedules? In this paper, we examine the performance of distributed scheduling heuristics incorporating either local or global knowledge, with and without the synchronous repair mechanism enabled. We conclude that the repair process is synergistic with an informed scheduling heuristic and that increased global knowledge leads to better schedules.

We present our experimental results in the domain of an airport ground service scheduler. Because of our attempts to capture many of the features of a complex real-world scheduling problem, we feel that these results generalize to distributed resource allocation or planning architectures in which:

- Agents receive tasks in batch, and schedule or plan simultaneously.
- The coordination relationship between agents is cooperative, based on task or resource sharing.

- No single agent has knowledge or responsibility for all instances of a particular resource type.

- Individual instances of resources are not interchangeable due to factors such as location, travel time, or capacity.

- Some version of a "best-first" heuristic has been demonstrated to work for single agent versions of the system.

In the following sections, we first describe our experimental testbed. We then describe a scheduling heuristic in which global knowledge can be incorporated at fairly low cost. Following this, we briefly review the distributed repair algorithm described in [9]. We then present our experimental results and conclusions.

2 Overview: The Distributed DSS

In order to test our approach to solving distributed resource-constrained scheduling problems, we have designed a distributed version of an existing knowledge-based scheduling system called DSS (the Dynamic Scheduling System) [4]. DSS provides a foundation for representing a wide variety of real-world scheduling problems. The DSS is a micro-opportunistic (c.f. Microboss [12]) scheduler based on a blackboard architecture. Orders are represented as tasks and subtasks – each resource-requiring subtask is assigned a *service goal*. These goals are assigned ratings according to the current scheduling heuristic and are rerated whenever constraints on the goal's order or on a resource that could potentially satisfy the goal are modified. 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. Agents will also transmit information regarding their current resource states to other agents.

2.1 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 scheduler. 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, 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. 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.

2.2 Exploiting Meta-level Information

In previous work, we discussed the use of high level abstractions of other agent's resource requirements and capacities in the control decisions of individual scheduling agents in a cooperative distributed scheduling system [8]. We showed how analysis of the abstracted resource requirements of remote agents could guide an agent's choice of local scheduling activities. We were also able to exploit this meta-level information by allowing the scheduling agents to make reasoned decisions about when to attempt to solve impasses locally through backtracking and constraint relaxation and when to request resources from remote agents. We now briefly describe these abstractions and show how to incorporate them into a low-cost multi-perspective variable-ordering heuristic.

2.2.1 Communication of Abstract Resource Profiles

A challenge in exchanging meta-level information is to develop abstractions that are inexpensive to calculate and transmit, yet convey sufficiently accurate profiles of agents' resources and scheduling goals to allow other agents to make accurate inferences about other agents' activities. In this section, we describe the protocol we have developed for the exchange and updating of resource profiles containing summarizations of the agents' committed resources, available resources, and estimated future demand.

Upon startup, each agent in DIS-DSS receives a set of orders to be processed. The agents examine these orders and generate an abstract description of their resource requirements for the scheduling period. For each resource type owned by the agent, this list consists of a list of intervals, with each interval annotated by a triple: resources in use, resources requested, and resources available.

The request field of this triplet represents an abstraction of the agent's true resource requirements. Certain aspects of a reservation such as mobile resource travel times to the objects to be serviced cannot easily be estimated in advance. Each resource request has associated with it an earliest possible start time (EST) and latest finish time (LFT). Because of interactions within an order, these times are not fixed and may change as related tasks are assigned. The true duration of the task is often much less than $LFT(task) - EST(task)$ and can be estimated by the scheduling agent using its domain knowledge regarding the typical time required to perform a task. The actual assignment of the resource may take place at any point within the interval $(EST(task), LFT(task) - duration(task))$. To capture this uncertainty, we define a measure for each task called the *average resource demand* defined as the tuple $ARD(task) = [(EST(task), LFT(task)), duration(task)/(LFT(task) -$

$EST(task)$] where the first element of the tuple indicates the interval of the demand and the second indicates the average demand generated by the task over the entire interval. Although less informed than the probabalistic texture measures used in [14], this measure manages to abstract both the demand and the flexibility associated with a particular task. The demand for a given resource type R at a given agent is therefore $\sum_i^{|T|} ARD(T_i)$ where T is the set of all tasks requiring a resource of type R and T_i is the ith member of that set.

Once resource abstractions have been developed for each resource type required (or possessed) by the agent, it transmits its abstractions to all other agents. Likewise, it receives abstractions from all agents. These abstracted resource profiles are used by each agent to predict when remote agents may request resources and when the local agent may need to borrow resources. This information guides the agent's decision-making process in determining both when to process local goals and when and from whom to request resources. Coordination using resource abstractions requires frequent updating of agents' resource abstractions in order to maintain accuracy, leading to a high overhead in terms of message passing. An update is transmitted whenever an agent allocates, releases, or reschedules a resource such that the number of resources requested or available during an interval changes. In a typical run of the Distributed DSS, we have found that updating of resource abstractions takes approximately 80% of the communication between agents. (Approximately 15% of the communication is dialog related to the requesting of resources and subsequent responses. The remaining messages are devoted to initialization and to the cancellation or refinement of lent resources). For a moderately sized scheduling run, this amounts to on the order of 800-900 communication episodes devoted to updating abstractions. Reducing the frequency of these updates decreases the effectiveness of the coordination (c.f. [8, 14]).

3 Heuristics for Distributed Scheduling

The Distributed DSS is a micro-opportunistic scheduler – each agent is continually rerating service goals and opportunistically selecting the most highly rated task to be scheduled next. The results of the scheduling episode depend critically on the choice of scheduling heuristic. For the experiments described in this paper, we implemented local and texture-based versions of the multiple-perspective heuristic (MPH) described by [4] and demonstratedto be superior to several other scheduling heuristics (such as minimum earliest starting time, minumum late finishing time, first-come-first-served, and minimum job slack [2]) at reducing order tardiness in both the AGSS and the turbine component plant [10] benchmarks. The *local* heuristic bases its rating only on an agent's local state. The *texture-based* version attempts to form an approximation of the global value that would be achieved by a centralized agent by using the abstracted resource profiles transmitted to each agent.[1] We present the distributed texture-based version of this heuristic in some detail, first because it outperforms previous distributed heuristics that we have tried, and secondly, because the methodology we used to derive this heuristic may prove of interest. These rating functions are variations of the 'most-tightly-constrained variable' heuristic. They allow the agents to select the next task to be scheduled based on some measure of its criticality based either on a local or global perspective. Functions such as these have been shown to be effective in single agent scheduling systems [11] and this is our motivation for employing these rating functions in our distributed system.

We note that the choice of a heuristic function is dependent on the nature of the cooperation between agents. In the DIS-ARM scenario, agents are assumed to be completely cooperative. In such a situation, it is appropriate for agents to consider the states of other agents. In a competitive situation, a local heuristic might actually be a better choice.

[1]A texture (c.f Fox) is an abstraction that allows one to get an overview of some feature of a domain, in this case, resource contention.

3.1 The Multiple-Perspective Heuristic

The multiple-perspective heuristic combines multiple features of the scheduling environment to generate a rating for each scheduling task. Simplified, it has the form

$$MPH(goal) = f_{LST}(goal) * f_{Constraints}(goal)$$

where the first term is a function of the latest starting time of the goal and the second is a measure of the "tightness" of the constraints on the goal as measured by contention with other goals, slack, and available resources. The first term is of more consequence in a dynamic scheduling situation rather than the batch scheduling environment of our AGSS scenario so we will not discuss it in further detail except to say that it gives the heuristic a flavor of dispatch scheduling when contention for resources is low.

The second term of the rating heuristic has the form

$$Constraints(goal) = Contention(goal) * Slack^{-1}(goal) * Resource\_Availability^{-1}(goal)$$

The use of the inverse notation indicates that this term increases in direct proportion to contention for the goal and in inverse proportion to the available slack for the goal and the availability of resources for satisfying the goal. Resource contention is defined as

Let $OG(goal)$ = the set of goals whose time windows overlap the time window of the goal being rated and that can be satisfied by the same resource type(s). Let the ith member of this set be denoted by OG_i.

$$Contention(goal) = \frac{\sum_{i=1}^{|OG|} overlap(goal, OG_i) * slack(OG_i)}{|OG|}$$

Slack is a measure of a goal's temporal flexibility – a goal with considerable slack can be assigned to many different time slots.

$$Slack^{-1}(goal) = ExpectedDuration(goal)/(LFT(goal) - EST(goal))$$

Finally, resource availability is defined as follows:

Let R be the set of resources of the correct type to satisfy the goal, and R_i be the *ith* element of this set.

$$Resource\_Availability^{-1}(goal) = \frac{\sum_{i=1}^{|R|} Used\_time(R_i, EST(goal), LFT(goal))}{(LFT(goal) - EST(goal)) * |R|}$$

The contention component of the MPH, therefore, increases with the number of overlapping goals, decreases when the goal has considerable slack, and increases when there are few available resources available to satisfy the goal within the desired time window.

A centralized and a local implementation of the MPH heuristic differ only in their definitions of the sets OG and R. When constructing these sets, a centralized heuristic would examine the goals and resources of all scheduling agents. The local heuristic uses only those goals and resources known to the agent responsible for scheduling the goal being rated.

The hypothesis underlying the development of the texture-based version of the MPH heuristic is that employing a heuristic that incorporates the resources and goals of all agents would yield superior results to those of the local heuristic. To ease the development of the texture-based heuristic, we first defined an *oracle* – a version of the MPH heuristic that was allowed to "peek" at the actual goals and resources of all agents. Each component of this heuristic was then compared to the corresponding component of the texture-based heuristic during a number of scheduling runs to ensure that they were of a comparable magnitude and the texture-based heuristic reasonably approximated the values of the oracle. We also compared the performance of the oracle heuristic vs. the local heuristic to ensure that our original hypothesis regarding the value of global knowledge was correct.

Constructing a texture-based analog of the MPH heuristic using the resource profiles communicated by agents is made difficult because there is no way to explicitly calculate the set OG, much less the slack of the goals in the set. Instead, we use the average resource demand texture described in section 2.2.1. This measure incorporates an approximation of the slack of overlapping goals, as any goal with considerable slack will contribute less to the demand for a resource at any given time. The contention component of the texture heuristic is computed as:

$$Sum\_Demand(goal) = \sum_{i=1}^{|Agents|} \sum_{j=1}^{|R^i|} Requested(R^i_j, EST(goal), LFT(goal))$$

$$Sum\_Available(goal) = \sum_{i=1}^{|Agents|} \sum_{j=1}^{|R^i|} Available(R^i_j, EST(goal), LFT(goal))$$

$$Contention_{texture}(goal) = Sum\_Demand/Sum\_Available$$

where R^i is the set of appropriate resource types owned by Agent i and R^i_j is the *jth* member of this set.

The other two terms of the contention component are trivially calculated. The goal is all that is necessary for the determination of slack. The resource profiles contain sufficient information to compute global resource availability. If received instantaneously, this value would mirror that of the oracle heuristic, however, delays inherent in the transmission of this information may cause it to be slightly out of date by the time it is incorporated into the goal rating.

4 Schedule Repair in the Distributed DSS

Iterative schedule repair algorithms have been studied by many researchers, including [7, 15, 3, 13]. In a centralized system, repairs are usually required when jobs to be scheduled are modified, added, or removed, or when resources become unavailable. Several systems, however, take the approach of first developing an almost correct schedule and then incrementally repairing constraint violations until a satisfactory schedule is produced [6, 5]. In a distributed system, repair may also be needed because of unanticipated interactions with other agents' schedules or simply to correct for a less than optimal resource assignment. The schedule repair algorithm that we have devised [9] may be used as a post-processing method but we have found it most effective when used during scheduling as a way to avoid constraint relaxations.

The schedule repair method is invoked whenever a scheduling goal can only otherwise be satisfied by performing a constraint relaxation that would adversely affect the order's due date. All agents in the system suspend their current problem-solving activities and collaboratively search for some reassignment of resources that would allow the current scheduling goal to be satisfied without a constraint relaxation. The synchronous replanning method 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.

4.1 The Cost of Distributed Schedule Repair

In this section, we informally discuss the cost of distributed schedule repair in the AGSS domain. Because there is a wide spectrum of possible repair and backtracking algorithms and trade-offs that could be made between speed and completeness these figures are only intended to give the reader an idea of the level of effort expended by our particular repair algorithm. Because of the potentially high cost of distributed search, we made a conscious decision when designing the repair algorithm to limit the scope of the search process. The repair process for a specific delayed goal is carried out only within the resource type satisfying that goal and consists of swapping resources in a search for a better set of allocations. The basic algorithm is based on gap expansion (and may fail in tightly compacted situations, even though suitable reallocations exist).

The heuristic by which agents select potential resources to cascade in the repair process is based on a global perspective of unused time and slack for resources in a time window surrounding the resource to be allocated. Successful repair plans tend to be quite short, with an average cascade length of 3 in the AGSS domain. The number of messages also tends to be small, with generally less than 100 repair goals being generated and transmitted over the course of 5 to 10 cycles (each cycle consists of one goal expansion by each participating agent). Unsuccessful searches for repair plans tend to generate more repair goals and messages. We have set a threshold in order to limit the effort expended, based on our observation that a search generating more than a couple of hundred states will likely never succeed. The number of replanning episodes in a standard scheduling episode varies widely, but averages around 10. Although the synchronous repair method performs considerably more search than, for example, a single resource assignment method, our initial results indicate that this may be made up for by decreased number of knowledge source executions during scheduling.

5 Experimental Results

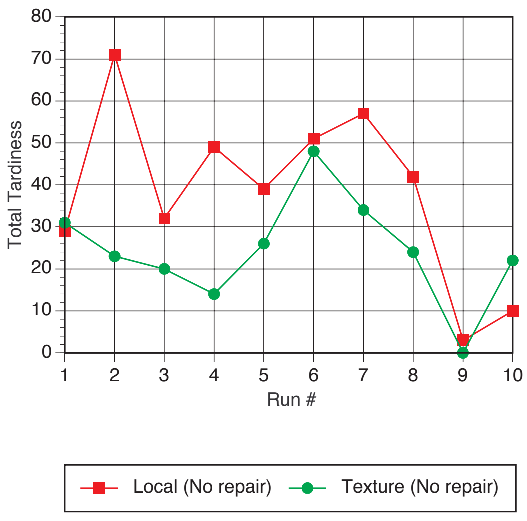
To answer some of the questions we have posed regarding the relative merits of coordination vs. replanning, we have run a number of experiments using the Distributed DSS in the AGSS domain. We chose three order sets developed for a three agent configuration of the DIS-ARM distributed scheduler as reported in [8]. The order sets were intractable, displaying considerable tardiness under distributed scheduling. Each order set has 40+ orders, each of which has six or more subtasks. 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 the **1200** and **0810** 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, however a schedule with zero tardiness is theoretically possible. The **1050** order set is severely over-constrained. Each test case was scheduled using each of the two variable-ordering heuristics, local and texture-based, both with and without synchronous repair. In all episodes, meta-level information was used to guide agent selection and resource allocation methods as described in [8].

Figure 1 compares the results of the local, and texture-based heuristics with and without the synchronous repair mechanism for the scenario containing orders beginning at 1200 hours. In all cases, the repair mechanism considerably improved scheduling results. The more informed texture-based heuristics consistently out-performed the local heuristic. Figure 2 compares the performance of the texture-based heuristic (with and without repair) with the performance of the local heuristic with repair enabled. This latter graph allows us to compare the relative importance of closely coupled cooperation between agents (the synchronous repair) and loosely coupled coordination (the texture-based heuristic). Recall that our speculation regarding the synchronous repair mechanism was that it might reduce the need for meta-level communication between agents [9]. In the **1200** order set, synchronous repair greatly improved the quality of schedules produced using a heuristic with a purely local perspective, however, the quality of the schedules improved even more for the texture-based heuristic. Results for the **0810** order set show similar trends. In the highly over-constrained **1050** order set, the texture measures do not perform significantly better than the local heuristics. What's more, resources are so tightly compacted that our repair algorithm, which works by locating and expanding unused space in the schedule, often cannot find opportunities to improve the schedule.

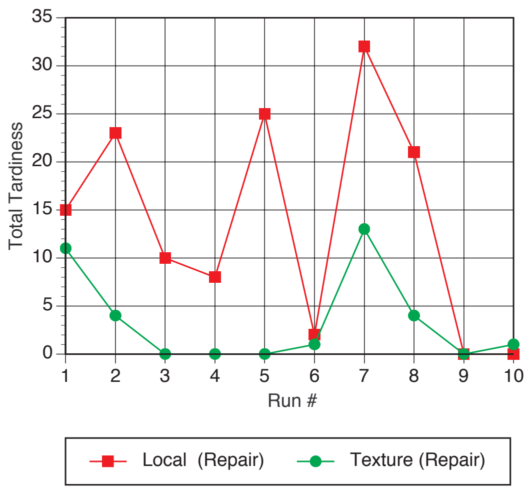
The conclusion we draw from the above is that the performance of a coordinated repair mechanism depends in part on the quality of the solution it begins with. Thus, the texture-based heuristic and the synchronous repair mechanism are synergistic in nature and both mechanisms should ideally be available in a distributed scheduler or planning system.

In Figure 3 and Table 1, we summarize the results for all scheduling episodes.

We can see that scheduling from a local perspective was generally less



(a) 1200-no-repair



(b) 1200-repair

Figure 1: The total tardiness in schedules produced for the 1200 order set both with and without synchronous repair. Each run represents a random assignment of resources and orders to agents.

successful than scheduling with globally-informed heuristics. The texture-based heuristic performed well, indicating that a texture abstraction based on average demand for resources is a good a measure of resource demand, at least for not unreasonably constrained environments. This is an encouraging result because the required information is inexpensive to calculate and easily transmitted to remote agents.

6 Conclusions

We have found that tightly coupled cooperation between agents in the form of a distributed search is a powerful method for improving the quality produced by distributed schedulers. Additionally, we have found that we can devise a high-quality texture-based variable-ordering heuristic using information that

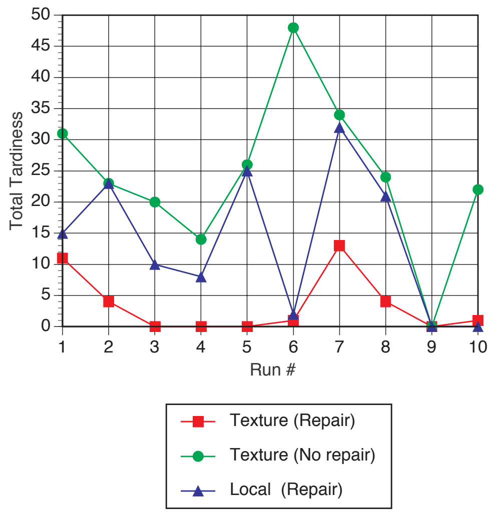


Figure 2: Direct comparison of the texture vs. the local heuristic with repair for the 1200 schedule.

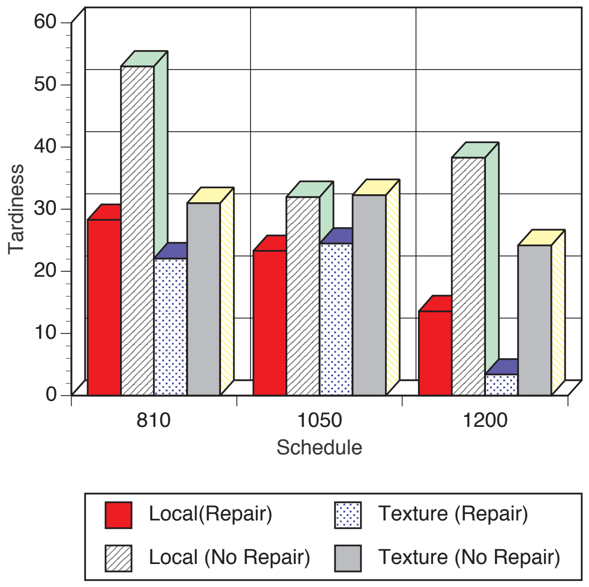


Figure 3: Graphical summary of experimental results.

Table 1: The average tardiness of all schedules compared according to heuristic and repair methods. Standard deviation is given in parentheses.

Scheduler	Environment		
	0810	1050	1200
Local	28.3 (20.6)	23.3 (6.3)	13.6 (11.3)
Local (No Repair)	53.0 (16.6)	32.3 (8.6)	38.3 (20.8)
Texture	22.1 (15.1)	24.5 (11.3)	3.4 (4.8)
Texture (No Repair)	31.0 (15.0)	32.3 (12.4)	24.2 (12.6)

is relatively inexpensive to transmit and to incorporate into a rating scheme. Incorporating these resource profile abstractions of other agent's resource availability and demands into the variable ordering heuristic also improves the quality of schedules. Neither technique alone is a panacea – the best schedules were obtained when loosely coupled coordination techniques were combined with closely coupled cooperation between agents. Certain environmental characteristics, such as a very high degree of over-constraint, may reduce the effectiveness of both techniques.

We note that there is a high degree of variance in results depending on the initial allocation of orders and resources to agents despite the fact that the sets of orders and resources remain the same throughout. Such variance is common in distributed systems in which issues of timing can critically affect results. Identifying the features leading to this variance will lead us towards even better algorithms for performing distributed scheduling. Some of these features, related to interactions between asynchronous activities in a distributed best-first scheduler, are presented in a companion paper [1].

7 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. IRI-9523419 and No. 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

[1] Mike H. Chia, Daniel E. Neiman, and Victor R. Lesser. Coordinating asynchronous agent activities in a distributed scheduling system. Submitted to ICMAS-97.

[2] Edward W. Davis and James H. Patterson. A comparison of heuristic and optimal solutions in resource-constrained problem scheduling. *Management Science*, 21(8):944–955, April 1975.

[3] Herwig Henseler. Distributed real-time rescheduling. In *Proceedings for the International Conference on Improving Manufacturing Performance in a Distributed Enterprise: Advanced Systems and Tools*, pages 173–178, Edinburgh, July 1995.

[4] David W. Hildum. *Flexibility in a Knowledge-Based System for Solving Dynamic Resource-Constrained Scheduling Problems.* PhD thesis, Computer Science Dept., University of Massachusetts, Amherst, MA 01003, August 1994.

[5] Mark D. Johnston and Steven Minton. Analyzing a heuristic strategy for constraint-satisfaction and scheduling. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 9. Morgan Kaufmann, 1994.

[6] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17–24, Boston, 1990.

[7] Kazuo Miyashita and Katia Sycara. Adaptive case-based control of schedule revision. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 10. Morgan Kaufmann, 1994.

[8] Daniel Neiman, David Hildum, Victor Lesser, and Tuomas Sandholm. Exploiting meta-level information in a distributed scheduling system. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 394–400, Seattle, August 1994.

[9] Daniel E. Neiman and Victor R. Lesser. Cooperative repair of schedules in a distributed scheduling system. In *Proceedings of AI and Planning Systems (AIPS-96)*, May 1996.

[10] Peng Si Ow. Experiments in knowledge-based scheduling. Technical report, The Robotics Institute, Carnegie Mellon University, 1986.

[11] Norm Sadeh and Mark S. Fox. Value and variable ordering heuristics for activity-based job-shop scheduling. In *Proceedings, Fourth International Conference on Expert Systems in Production and Operations Management*, pages 134–144, Hilton Head, SC, May 1990.

[12] Norman Sadeh. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling*. PhD thesis, Carnegie Mellon University, Pittsburgh PA, March 1991.

[13] Arvind Sathi and Mark S. Fox. Constraint-directed negotiation resource reallocations. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence Vol. II*, pages 163–193. Morgan Kaufmann, 1989.

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

[15] Monte Zweben, Brian Daun, Eugene Davis, and Michael Deale. Scheduling and rescheduling with iterative repair. In Monte Zweben and Mark S. Fox, editors, *Intelligent Scheduling*, chapter 8. Morgan Kaufmann, 1994.