Distributed Constraint Satisfaction through Constraint Partition and Coordinated Reaction

1

JyiShane Liu and Katia P. Sycara
jsl@cs.cmu.edu katia@cs.cmu.edu

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Phone: (412) 268-8825

Abstract

We present a methodology, called Constraint Partition and Coordinated Reaction (CP&CR), for distributed constraint satisfaction based on partitioning the set of constraints into subsets of different constraint types. Associated with each constraint type is a set of specialized agents, each of which is responsible for enforcing constraints of the specified type for the set of variables under its jurisdiction. Variable instantiation is the joint responsibility of a set of agents, each of which has a different perspective on the instantiation according to a particular constraint type and can revise the instantiation in response to violations of the specific constraint type. The final solution *emerges* through incremental local revisions of an initial, possibly inconsistent, instantiation of all variables. Solution revision is the result of coordinated local reaction of the specialized constraint agents. We have applied the methodology to job shop scheduling, an NP-complete constraint satisfaction problem. Experimental results on a benchmark suite of problems show that CP&CR outperformed three other state-of-the-art scheduling techniques, in both efficiency and number of problems solved. In addition, we experimentally tested the utility and effectiveness of various types of coordination information that the agents exchange.

Themes: Distributed constraint satisfaction, agent societies, emergent system behavior.

1. Introduction

Many problems of theoretical and practical interest (e.g., parametric design, resource allocation, time-dependent scheduling) can be formulated as constraint satisfaction problems. Informally, a constraint satisfaction problem (CSP) is defined by a set of variables, each of which takes its value (is instantiated) from a given domain, and a set of constraints that restrict the admissible variable instantiations. To find a solution to a CSP means to find an assignment of values (an instantiation) for all variables, such that all constraints are satisfied. Recent work in DAI has considered the distributed constraint satisfaction problem (DCSP) [15, 6, 13] in which variables of a CSP are distributed among agents. Each agent has a subset of the variables and tries to instantiate their values. Constraints may exist between variables of different agents and the instantiations of the variables must satisfy these inter-agent constraints. Different models of assigning variables to agents have been investigated. In [15] each agent is

[1]this research is being supported by DARPA under contract number F30602-90-C-0119.

responsible for instantiating a single variable, while in [13], each agent is responsible for a subset of variables, In these approaches, each agent was responsible for checking that all constraints involving the values of variables under its jurisdiction were satisfied, or identifying and resolving any constraint conflicts through asynchronous backtracking. Variables were instantiated in some order, according to a static ([15]) or dynamic ([13]) variable and value ordering, and the final solution was generated by merging partial instantiations that satisfied the problem constraints.

In this paper, we present an approach, called Constraint Partition and Coordinated Reaction (CP&CR), in which the set of agents is partitioned into agent subsets according to the types of constraints present in the DCSP. The fundamental characteristics of CP&CR are: (1) divide-and-conquer with effective coordination (2) avoid sophisticated inter-agent interactions and rely on collective simple local reactions. CP&CR divides a Constraint Satisfaction Problem into several subproblems, each of which concerns the satisfaction of constraints of a particular type. Enforcement of constraints on variables within a subproblem is assigned to a dedicated local problem solving agent which revises variable instantiations so that its own constraint type is satisfied. Since each variable may be restricted by more than one constraint, this means that the instantiation of a variable may be changed by different local problem solving agents. Each agent is iteratively activated and examines local views of a current solution. If it does not find any conflicts in the current iteration, it leaves the current solution unchanged and terminates its own activation. If it does find local constraint violations, it changes the instantiation of one or more variables. A final solution is an instantiation of all variables that all agents agree on, i.e. it does not violate any constraints.

The agents are unaware of each other's presence and constraints. Since constraint enforcement (change in the instantiation of certain variables) by a local problem solver may result in violations of constraints of other agents, the effectiveness of our approach requires coordination between local problem solvers on how they instantiate and revise the instantiation of variables to satisfy their own constraints. The exchange of coordination information helps the agents hedge against the myopia that is implied by the local nature of their problem solving. Experimental results reported in section 5 show the effectiveness and utility of various types of coordination information.

The domain of application of the methodology is job shop scheduling, one of the difficult constraint satisfaction problems. Job shop scheduling deals with allocating a limited set of resources to a number of activities (operations) associated with a set of orders (jobs). Job shop scheduling is a well-known NP-complete problem [7, 5]. Constraint-based approaches have been applied to the scheduling problem with very good results [4, 12, 11]. CP&CR views each activity as a *variable*. A variable's *value* corresponds to a reservation for an activity. A reservation consists of a start time and the set of resources needed by the activity. The dominant constraints in job shop scheduling are *temporal activity precedence* and *resource capacity* constraints. The temporal precedence constraints along with a job's release date, due date and activity durations restrict the set of acceptable start times for each activity. The capacity constraints restrict the number of activities that can use a resource at any particular point in time and create conflicts among activities that are competing for the use of the same resource at overlapping time intervals. The goal of a scheduling system is to produce schedules that respect the problem constraints, i.e. release and due dates, as well as temporal relations and resource capacity constraints.

In contrast to approaches [12, 11] that utilize incremental construction of partial schedules to produce a complete schedule, our approach first builds an initial schedule that possibly contains constraint violations and *incrementally revises* it to produce a conflict-free schedule. The revisions are made by specialized agents, each of which has a local view of a variable and can change the value (the start time) of the variables under its jurisdiction. Agents are of two types: Resource Agents that are responsible for enforcing resource capacity constraints and Order Agents, responsible for enforcing temporal precedence constraints. In this way, each variable is manipulated by a Resource Agent and an Order Agent. Schedule revision is the result of coordinated local reactions of the specialized constraint agents. The approach has been implemented in a system called CORA (COordinated Reactive Agents). Experimental results, presented in section 5 on a set of benchmark problems attest to the effectiveness of the approach as compared with other constraint-based scheduling methods.

2. Related Work

Approaches based on opportunistic heuristic search [4, 12, 11] generate schedules by opportunistically focusing attention to promising parts of the search space (e.g. bottleneck resources) and assigning one value to a variable at a time. Typically, they analyze the current situation, determine which is the next variable they should focus on, and then decide what is the best value to assign taking into consideration all involved constraints. After a variable has been instantiated, constraint propagation is performed to identify constraint violations that get resolved either by backtracking or by constraint relaxation. These approaches usually suffer intensive computational overhead. Constraint-posting approaches [2, 1, 10] generally do not commit to value assignments at the beginning but analyze the current situation, post additional constraints to exclude capacity conflicts, and then deduce variable assignments from the resulting network of constraints. CP&CR differs from the above approaches in that (a) it builds an initial, possibly flawed schedule and incrementally revises it, and (b) it does not perform global constraint analysis. Each constraint is locally enforced by coordinated reactions of the constraint specialists.

Iterative schedule repair [8, 16] is similar to CP&CR in that the solution is generated by iterative revision of an initial rough solution. However, in these approaches, conflicts are reduced by global evaluation of current conflicts and centralized decisions on which conflict to resolve next are made. The work of [9] is closer to CP&CR in that revision heuristics that have been acquired through case-based learning are used for incremental local patching of an initial schedule. Each revision locally enforces all constraints. After each local repair, constraint propagation identifies constraint conflicts caused by the repair. CP&CR differs from this approach in that it reduces conflicts by distributed local conflict resolution where each type of constraint is considered and resolved separately.

Similar to the decentralized character of CP&CR, [13] reported a distributed scheduling system where each agent has many variables under its jurisdiction and is responsible for resolving both capacity and precedence constraints associated with those variables. In [3] a Distributed Asynchronous Scheduler (DAS) is reported which consists of a number of reactive/opportunistic agents in a hierarchical organization. Unlike CP&CR, local problem solvers in these systems are all sophisticated agents and they perform intensive computation for their interactions.

3. Distributed Scheduling by Constraint Partition & Coordinated Reaction

Scheduling constraints are partitioned into two categories: temporal precedence and resource capacity constraints. Within each constraint type, subproblems are formulated. Each subproblem is assigned to a separate agent. In particular, enforcing capacity constraints on a given resource is a subproblem that is under the responsibility of a Resource Agent; enforcing temporal precedence constraints within an order is a subproblem that is assigned to an Order Agent. Therefore, for a given scheduling problem, the number of subproblems (and the number of agents) is equal to the sum of the number of orders plus the number of resources.

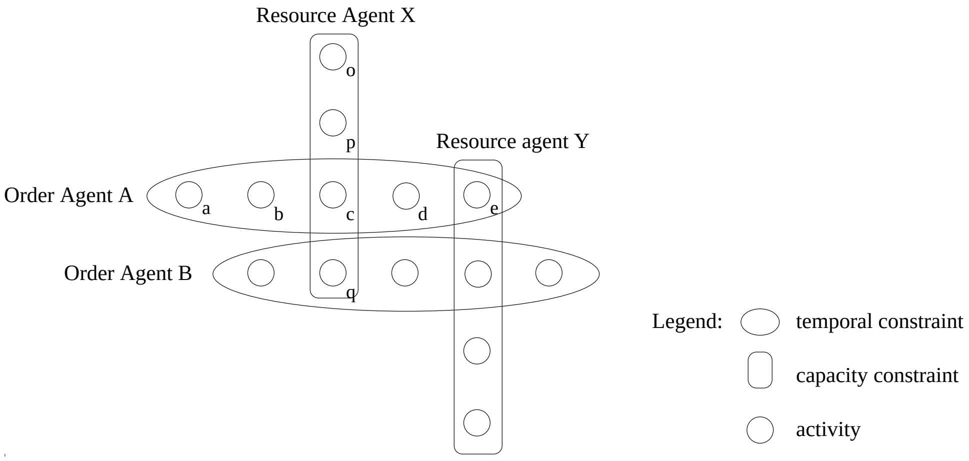

Figure 3-1: Constraint partition of a scheduling problem

Figure 3-1 shows a partial picture of the problem partition. Order Agent A is responsible for the satisfaction of temporal constraints on activities (a, b, c, d, e). Resource Agent X is dedicated to enforcing capacity constraints on activities (o, p, c, q). When these agents are activated, they can change instantiations of activities under their jurisdictions to satisfy their own constraints. Therefore, each activity (e.g. activity c) can be manipulated by both an Order Agent (e.g. Order Agent A) and a Resource Agent (e.g. Resource Agent X). Manipulation of activities by Order Agents may result in constraint violations for Resource Agents and vice-versa. Therefore, coordination between agents is crucial for prompt convergence on a final solution.

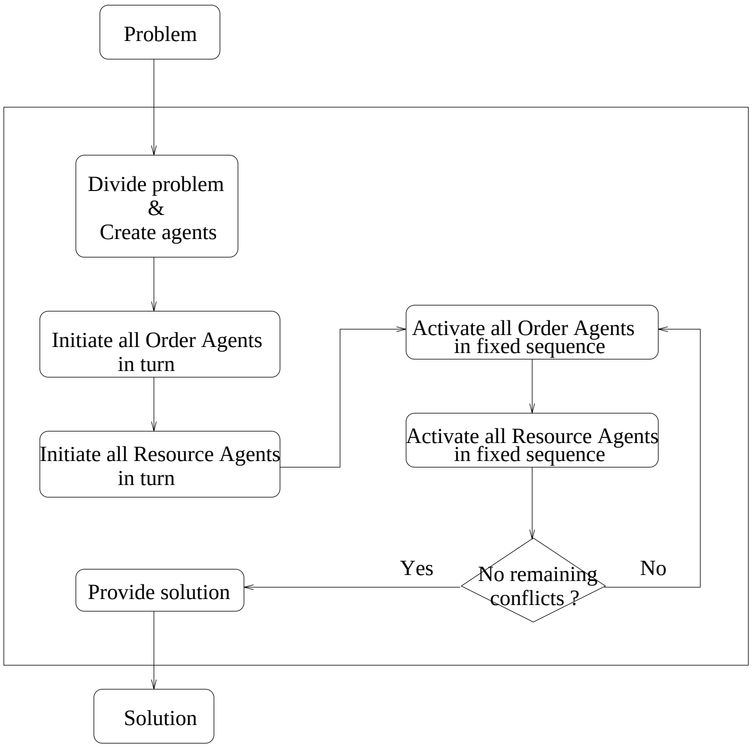

Figure 3-2: Control flow of CORA

In addition to the Order Agents and Resource Agents, the system includes a Manager agent. The Manager is a simple controller who performs the following tasks: (1) decomposition of the input scheduling problem according to resource and order constraints, (2) creation of the corresponding resource and order agents, (3) activation of the agents, and (4) outputting of the final solution when the system reaches quiescence, i.e. when no agent reacts to the current solution any more.

At system initialization, all order agents are activated first, followed by activation of all resource agents. At each subsequent iteration, the manager activates all order agents and all resource agents in turn. When an agent is activated, it revises the current values of the activities under its jurisdiction according to its local view. Processing stops when no agent has any constraint violations left. Figure 3-2 depicts the control flow of the system. Order agents are activated first because they calculate the *time boundary* for each activity under their jurisdiction (see figure 3-3). Time boundary information is associated with each activity and is used by the corresponding resource agent in instantiating or revising the activity's start time. The time boundary of an activity is defined as the interval between its earliest possible start time and its latest possible finish time. The boundary information for each activity is calculated only once during the initial activation of Order Agents and gets associated with the activity.

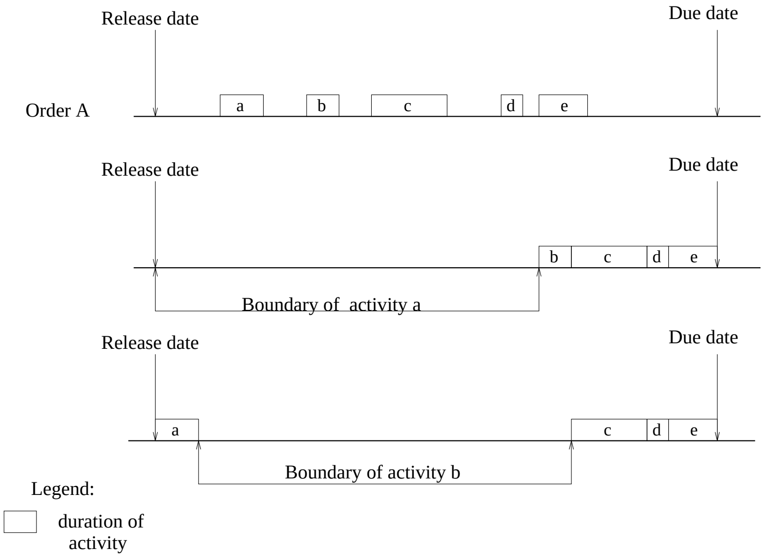

Figure 3-3: Calculation of temporal boundaries by Order Agent

The initial solution, that is subsequently revised, is generated by the Resource Agents. The Resource Agents are activated after boundaries of all activities have been defined by the Order Agents. Each Resource Agent calculates the contention ratio for its resource by summing the durations of activities on the resource and dividing by the interval length between the earliest and latest time boundary among the activities. If this ratio is larger than a certain threshold, a Resource Agent announces itself as a Bottleneck Resource Agent. Activities under the jurisdiction of a Bottleneck Resource Agent are marked as Bottleneck activities.

Each Resource Agent allocates resource intervals to activities under its jurisdiction according to their boundaries. Figure. 3-4 depicts different conditions of allocation of resource intervals based on sequence of activity boundaries. A Resource Agent allocates to each activity the earliest free interval on the resource. For example, activity o in figure 3-4 is allocated first because it has the earliest left boundary, and it gets the earliest interval. The next allocation depends on the next available start time (finish time of activity o) and the boundaries of remaining activities. If there are more than one activity eligible for

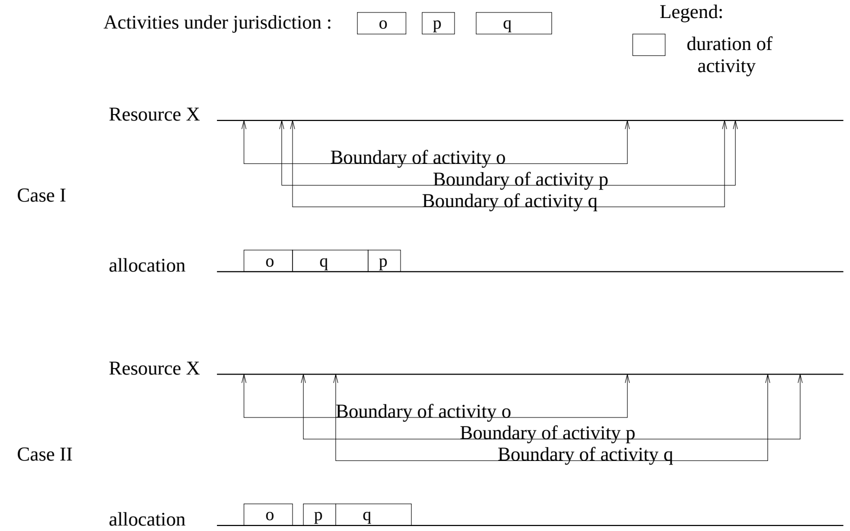

Figure 3-4: Initial allocation of resource intervals by Resource Agent

allocation (left boundary is earlier than the next available start time), the next resource interval is allocated to the activity who has the earliest right boundary, such as activity q in Case I. If there is only one activity eligible, the resource interval is allocated to that activity at the next available start time. If there is no activity eligible, then the activity with the earliest left boundary is allocated to its earliest possible interval, such as activity p in Case II.

When there are multiple Resource Agents identifying themselves as Bottleneck Resource Agents, the Manager intervenes to choose the one with the highest resource contention ratio as the Primary Bottleneck Resource Agent and tells the others that they are Secondary Bottleneck Resource Agents. Secondary Bottleneck Resource Agents are then initiated again and re-allocate resource intervals according to a sequence of activities *with order-correspondence* to the sequence of activities used by the Primary Bottleneck Resource Agent (see Figure 3-5). This represents a coordination between Bottleneck Resource Agents and it plays an important role for solving scheduling problems with multiple bottleneck resources. Note that after this initial information exchange, the agents coordinate strictly according to local views.

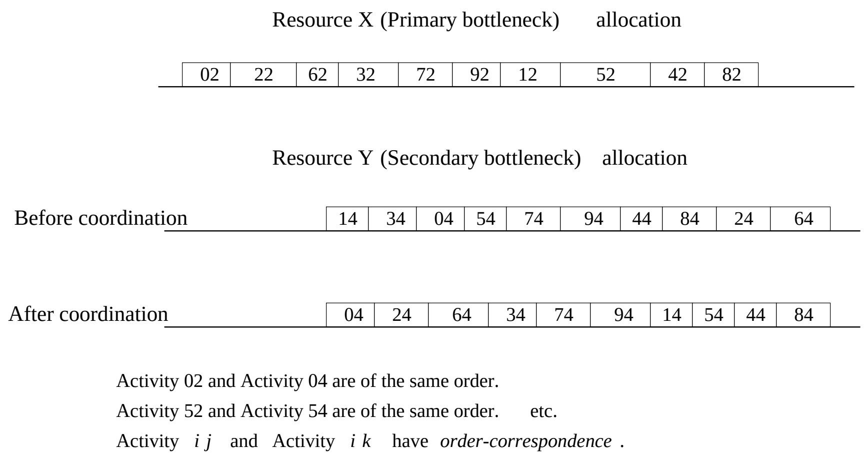

Figure 3-5: Coordination between Bottleneck Resource Agents

After the initial activation of Resource Agents, all activities are instantiated with a start time. This instantiation is the initial schedule that may subsequently be revised. The initial schedule does not contain resource capacity conflicts but it may contain temporal precedence constraint conflicts. Order Agents and Resource Agents coordinate through local reactions to the current solution so that their collective behavior results in a conflict-free final solution.

3.1. Agent Coordination

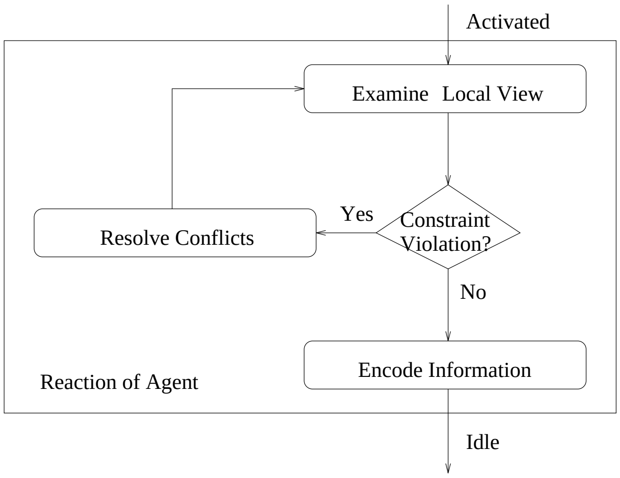

Figure 3-6: Agent reaction to current solution

When activated, each agent reacts to the current instantiation of activities under its responsibility by going through an Examine-Resolve-Encode cycle (Figure 3-6). It first examines its local view of current solution, i.e. the values of the variables under its jurisdiction. If there are constraint violations, it changes activity instantiations to resolve conflicts (section 3.2). Since instantiation of an activity may be changed by an Order Agent and a Resource Agent back and forth, it is very important that they coordinate with each other in making changes to the current solution.

Since agents have no awareness of the existence of others, they do not communicate with each other directly. Instead, they coordinate by reading and writing coordination information on activities. Coordination information represents an agent's "view" on the partial current solution and is consulted when the agent needs to change the current instantiation to resolve its conflicts. After resolving conflicts, an agent *writes down its views* on current instantiations on each activity as coordination information. Coordination information *written* by an Order Agent on an activity is referenced by a Resource Agent, and vice-versa.

3.1.1. Coordination information provided by Order Agents (for Resource Agents)

Boundary: the interval between the earliest start time and latest finish time of an activity (as described in Figure 3-7). It represents the overall temporal flexibility of an activity. If a Resource Agent moves the activity outside this range, it will cause constraint violation for the Order Agent responsible for the activity. Activity boundaries are calculated only once during initial activation of Order Agents.

Temporal Slack: an interval between the current finish time of the previous activity and current start time of the next activity (see Figure. 3-7). It indicates the temporal range within which an activity may be scheduled without causing temporal constraint violations. (This is not guaranteed since temporal slacks of adjacent activities are overlapping with each other.)

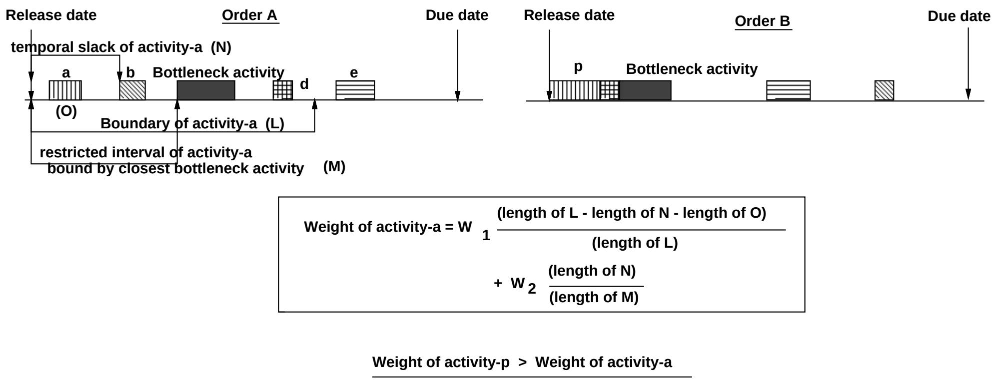

Figure 3-7: Temporal slack and weight determination

Weight: the weighted sum of relative temporal slack with respect to activity boundary and relative temporal slack with respect to the interval bound by the closet Bottleneck activities. It is a measure of the likelihood of the activity "bumping" into an adjacent activity, if it gets rescheduled. The higher the weight, the more likely it is that rescheduling the activity will cause conflicts. Therefore, a high weight represents a preference for not moving the activity (Figure 3-7).

Extra-weight: an additional measure of the importance of not moving the activity. There are three conditions for an activity to have extra-weight: (1) when an activity has bumped into a bottleneck activity and is moved to a new location, (2) when the number of times an activity has moved reaches a certain threshold, and (3) when a bottleneck activity is moved to a new location. At each of the three conditions, the extra-weight of this activity is set to a predetermined number.

3.1.2. Coordinated information provided by Resource Agents (for Order Agents)

Bottleneck Tag: a tag which marks that this activity uses a bottleneck resource. This tag is put by a Bottleneck Resource Agent on all activities under its jurisdiction. It implies that the responsible Order Agent should treat this activity differently.

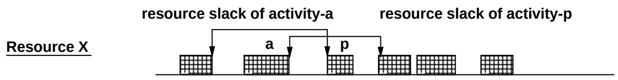

Figure 3-8: Resource slack

Resource Slack: an interval between the current finish time of the previous activity and the current start time of the next activity on the resource timeline (see Figure 3-8). It indicates the range of activity locations to which an activity can be moved without causing capacity constraint violations. (There is no guarantee on this since resource slacks of adjacent activities are overlapping with each other.)

Location Change: an index of whether the location on the timeline of this activity, set by an Order Agent, has been changed by a Resource Agent.

Change Frequency: a counter of how frequently the location of this activity set by an Order Agent is changed by a Resource Agent. It reveals a brief history of activity moves since agents do not keep track of previous locations of activities. High change frequency indicates that the locations where the Order Agent has moved the activity in the past caused capacity constraint violations which could not be resolved by continuing to move the activity. Therefore, the Order Agent should change locations of other activities (including bottleneck activity) to satisfy its constraints.

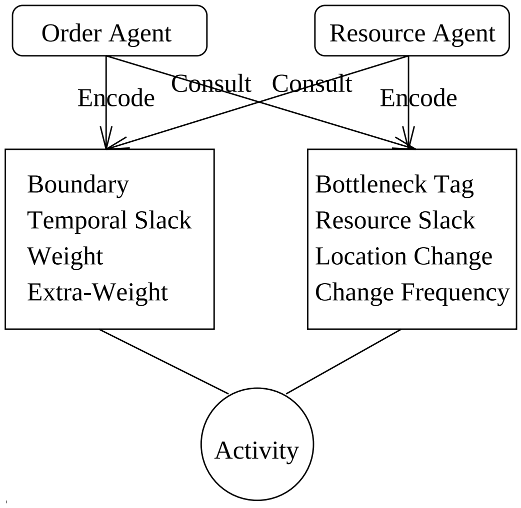

Figure 3-9: Coordination information

Figure 3-9 shows two groups of coordination information encoded on an activity. Coordination information encoded by an Order Agent is consulted by the corresponding Resource Agent, while an Order Agent consults coordination information encoded by the corresponding Resource Agent.

3.2. Reaction Heuristics

When an agent finds a constraint violation in an activity under its jurisdiction, it employs local reaction heuristics to resolve the violation. The reaction heuristics attempt to minimize the ripple effects of causing conflicts to other agents as a result of fixing the current constraint violation. Conflict minimization is achieved by minimizing the number and distance of activity moves.

Order Agent

An Order Agent considers conflict resolution in the context of conflict pairs. A conflict pair involves two adjacent activities whose current locations violate the precedent constraint between them (see Figure 3-10). Associated with each activity is the distance it needs to be moved to resolve the conflict. If either one of the two activities is a bottleneck activity, the conflict pair is categorized as a bottleneck conflict pair, and is given a higher conflict resolution priority. An Order Agent treats a bottleneck activity as somewhat anchored. Since moving bottleneck activities causes severe conflict ripple effects, a bottleneck activity is moved only as a last resort.

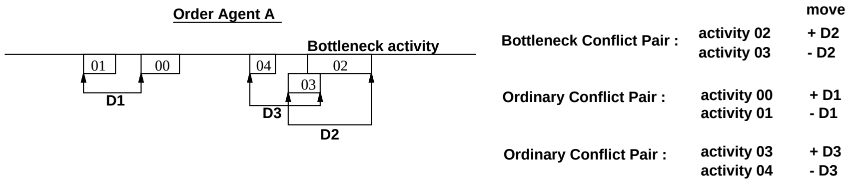

Figure 3-10: Conflict identification by Order Agent

Conflict pairs are resolved one by one. To resolve a conflict pair, an Order Agent essentially determines which activity to move according to the distance associated with each activity. For a bottleneck conflict pair, if the change frequency of the non-bottleneck activity is below a threshold, it is the one to be moved. Otherwise, the bottleneck activity will be moved. To decide which activity to move in an ordinary (non-bottleneck) conflict pair, an Order Agent takes into consideration additional factors, such as moving feasibility of each activity, change frequency, resource slack and location change.

A Resource Agent is concerned with enforcing resource capacity constraints. If a capacity constraint is violated, the Resource Agent completely re-allocates the overallocated resource interval to the competing activities in such a way as to resolve the conflict and, at the same time, keep the location changes to each activity to a minimum.

Ordinary Resource Agent

An Ordinary Resource Agent re-allocates a resource interval to activities based on their weights. Activities with higher weights get allocated first. When allocating a resource interval to an activity, an Ordinary Resource Agent tries the activity's current location first. If it has been preempted by another activity, an Ordinary Resource Agent looks for the two most adjacent intervals available (one left, one right) and chooses one for the activity according to its boundary and temporal slack. Since an activity's weight is a measure of the desire of the corresponding Order Agent to keep the activity at its current location, activity location decisions based on weight reflect group coordination. Figure 3-11 depicts how an Ordinary Resource Agent resolves conflicts. For example, the current location of activity 21 was preempted by activity 81 which has higher weight. Therefore, activity 21 gets an immediately adjacent interval.

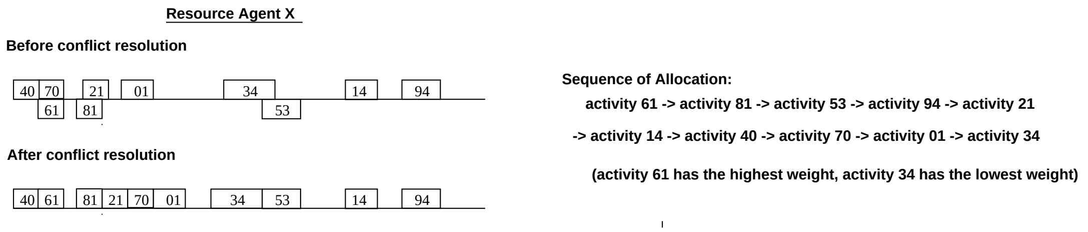

Figure 3-11: Conflict resolution of Ordinary Resource Agent

Bottleneck Resource Agent

A Bottleneck Resource Agent has high resource contention ratio. This means that most of the time a Bottleneck Resource Agent does not allow resource slack between resource intervals (all resource intervals are immediately adjacent with each other). When activity moves occur, capacity constraint violations are very likely to occur. A Bottleneck Resource Agent considers the conflict size of capacity violations. The conflict size is the amount of overlap of activity reservations on the resource. If the conflict size is small and if right-shifting activities on the time line does not cause violations, then right shifting is performed (see Figure 3-12 (i)). Otherwise, it re-sequences activities according to their current locations, and then re-allocate resource intervals according to the new sequence of activities with no slack between each activity (see Figure 3-12 (ii)).

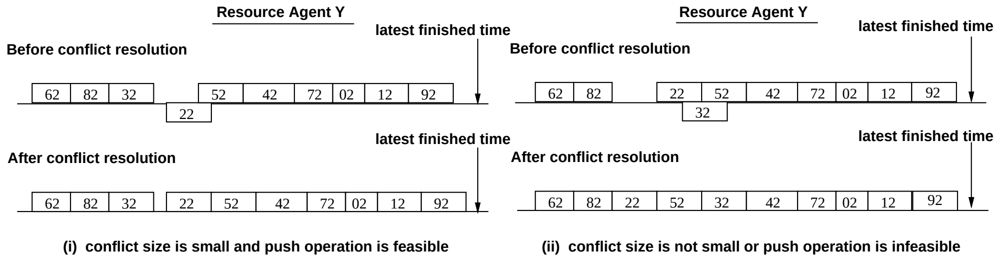

Figure 3-12: conflict resolution of Bottleneck Resource Agent

4. Solution Evolution

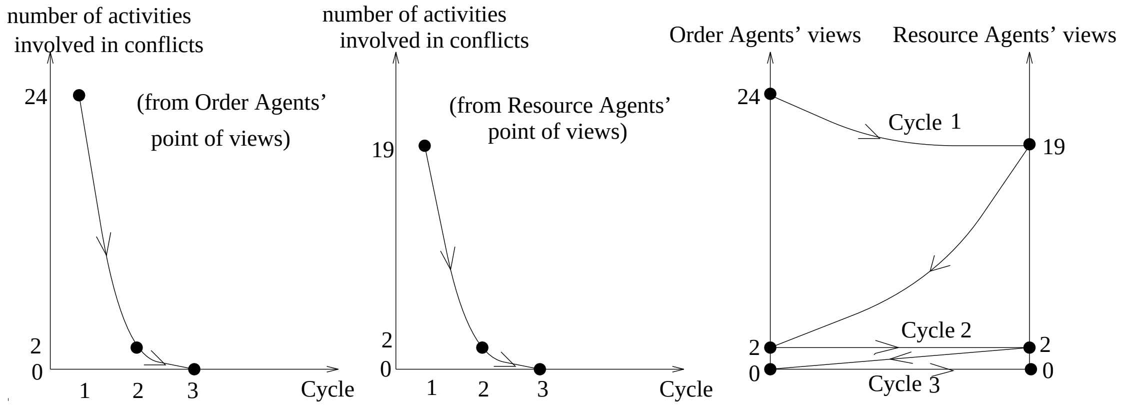

Figure 4-1: Solution evolution for a simple problem

Figure. 4-1 shows a solution evolution process for a relatively simple problem. In cycle 1, when Order Agents are first presented with the initial solution, there are totally 24 activities involved in temporal conflicts. After each Order Agent reacts to the current solution, Resource Agents find a total of 19 capacity constraint conflicts. Each Resource Agent reacts to the current solution. In the beginning of cycle 2, Order Agents only find 2 activities involved in conflicts. After the Order Agents' reactions, the Resource Agents again find 2 activities in conflicts. In cycle 3, both Order Agents and Resource Agents find no activity in conflict. A conflict-free solution has evolved.

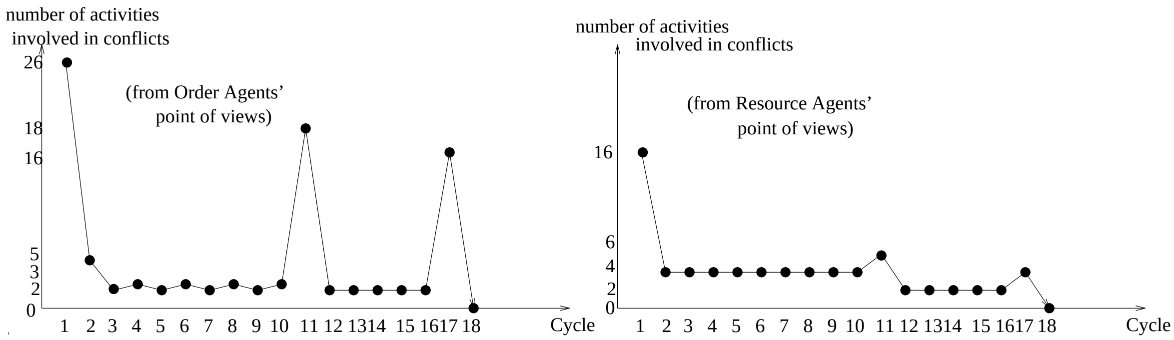

Figure 4-2: Solution evolution for a more difficult problem

Figure. 4-2 shows a solution evolution process for a more difficult problem. In the beginning of cycle 1, Order Agents find 26 activities involved in conflicts. After coordinated reactions of all agents, the number of activities involved in conflicts is reduced to 2 when Order Agents are activated in cycle 3. However, from cycle 3 to cycle 9 the solution evolution process is trapped in an oscillation. The number of activities involved in conflicts oscillates between 2 or 3 for Order Agents and 4 for Resource Agents. This indicates that activity values got switched back and forth by the respective agents because the agents could not find common ground for satisfying their respective constraints. However, since Order Agents and Resource Agents, have only local views, they are not aware of the situation. The crucial piece of coordination information that allows the agents to escape from the oscillation is the change frequency of activity location which serves as a history of activity moves. When the change frequency exceeds a heuristically determined threshold the Order Agent responsible for the variable instantiation moves a bottleneck activity. This initially increases the number of conflicts (as can be seen in the figure) but very soon it allows convergence to a conflict-free solution.

5. Experimental Results

We conducted two sets of experiments with CORA. At first, we compared CORA with three other state-of-the-art scheduling methods on a benchmark suite of scheduling problems. The results show that CORA outperformed the other methods in terms of number of problems solved and computational efficiency (CPU time). We also investigated the effects of coordination information in the system. We compared system performances on a set of coordination configurations ranging from no information to some information to adequate information. The results confirmed that coordination information facilitates successful and fast solution evolution.

The experiments were conducted on the suit of benchmark constraint satisfaction scheduling problems proposed in [11]. The benchmark consists of 6 groups of 10 problems, each of which has 10 jobs of 5 activities and 5 resources. Each job has a linear process routing which visits each one of the five resources. Activity sequences in the process routing are generated randomly, while bottleneck resources are visited after a fixed number of activities to further increase resource contention. Each group of problems differs in two respects: (1) spread of the release and due dates among jobs; (2) number of a-priori bottlenecks. The spread is controlled by varying the amplitute of the intervals within which release and due dates are generated. Three spread levels are introduced: wide (w), narrow (n), and null (0), i.e., both release and due date intervals are collapsed to single points. Aside from different spread levels of release and due dates, the benchmark also considered 1 and 2 a-priori bottlenecks conditions.

5.1. Comparison with other scheduling methods

Three other heuristic scheduling methods are compared with CORA on the benchmark: (1) Constraint Partition Scheduling [10], (2) Min-Conflict Iterative Repair [8], and (3) Micro-Opportunistic Search [11].

Constraint Partition Scheduling (CPS) constructs solutions by repeatedly identifying bottleneck conflicts and posting constraints to resolve them [10]. Analysis of resource capacity is based on a stochastic simulation. The final solution is deduced from the resulting constraint network.

Min-Conflict Iterative Repair (MIN-CONF) starts from an initial inconsistent solution and iteratively repairs it until a conflict-free solution is found. The initial solution is generated based on stochastic simulation. At each repair iteration, a variable is selected and a value is assigned to it based on the criterion of minimizing the number of remaining conflicts. If a solution is not found after a fixed number of iterations. a new initial solution is generated and the cycle repeats. Since MIN-CONF can run for a very long time if conflicts still exist, the maximum number of iterations allowed in the experiments was set to 5000.

Micro-Opportunistic Search (MICRO OPP) employs heuristic search with dynamic variable and value orderings. A solution is constructed by incremental extension of consistent partial value assignments. At each cycle, a variable is selected and assigned a value based on the heuristic orderings. Consistency is then enforced through the constraint network. The search backtracks when dead ends occur, e.g. the domain of the possible values of a variable becomes empty. MICRO OPP was run with two benchmark configurations - CHRON BKTRK, chronological backtracking and INTEL BKTRK, intelligent backtracking [14].

The performance results of CPS and MIN-CONF were reported in [10] in which these algorithms were each run 5 times for each problem because of their intrinsic random nature. CPU times for CPS and

MIN-CONF were not available in [10]. However, we show CPU times based on unpublished recent work by Muscettola as their optimistic estimates. The performance results of MICRO OPP have been reported in [14].

	CORA	CPS	MIN-CONF	MICRO OPP	
				CHRON BKTRK	INTEL BKTRK
w/1	10	10 (0.96)	10 (0.36)	10	10
w/2	10	9 (0.89)	3 (0.33)	10	10
n/1	10	10 (0.94)	5 (0.44)	8	10
n/2	10	10 (0.92)	1 (0.40)	9	10
0/1	10	10 (0.82)	4 (0.25)	7	10
0/2	10	9 (0.91)	0	8	10
Total	60	58	23	52	60
AVG. CPU time	2.9 seconds	78.43 seconds	298.42 seconds	234.72 seconds	128.78 seconds

The above Table reports the number of problems solved and the average CPU time needed over all the benchmark problems for each technique. Each row represents different groups of the benchmark. For example, n/2 represents the group of problem with narrow spread and two a-priori bottlenecks. The numbers in the parentheses are the repeatability measures for CPS and MIN-CONF. Since CORA and MICRO OPP are deterministic, they don't have repeatability measures. Among all techniques, only CORA and MICRO OPP with INTEL BKTRK were able to solve all 60 problems.

The last row of the table shows the average CPU time over the entire benchmark for each technique. Note that the CPU times of CPS, MIN-CONF, and MICRO OPP are obtained from implementations in Common Lisp on a DEC 5000/200 workstation, while CORA is implemented in Allegro Common Lisp with CLOS on a SPARC IPX workstation. A DEC 5000/200 workstation runs feaster than a SPARC IPX workstation.

5.2. Comparison on Coordination Configurations

In order to investigate the effects of coordination information on the system's performance, we constructed a set of five coordination configurations.

- C0 represents a configuration in which the system ran with no coordination information at all. Without boundary information, when initially activated, the Resource Agents allocate resource intervals according to random sequences. When the Order Agents are activated, they resolve conflicts by randomly changing the instantiation of one of the two activities in each conflict pair. Similarly, the Resource Agents resolve conflicts based on random priority sequences.
- C1 represents a configuration in which only boundary information is available. The Resource Agents use this information for heuristic initial allocation of resource intervals. After the initial schedule is generated, no other information is available for conflict resolutions.
- C2 represents a configuration in which boundary and bottleneck tag information is available. The Resource Agents use the boundary information for heuristic initial allocation of resource intervals. The Order Agents use the bottleneck tag information to bias resolution of conflict pairs.

- C3 represents a near-complete configuration in which all coordination information is provided for the Resource Agents and Order Agents except coordination between Bottleneck Resource Agents on the initial allocation of resource intervals.
- C4 represents a complete configuration including initial coordination between Bottleneck Resource Agents.

C0 : No coordination information

C1 : Boundary (heuristic initial allocation)

C2 : Boundary + Bottlenck tag

C3 : Boundary + Temporal slack + Weight + Extra weight + Bottleneck tag + Resource slack + Location change + Change frequency

C4 : Boundary + Temporal slack + Weight + Extra weight + Bottleneck tag + Resource slack + Location change + Change frequency + Coordination between Bottleneck Resource Agents

Overall Performance	Coordination configuration				
	C0	C1	C2	C3	C4
No. of Porb. Solved(Avg.)	8.0	15.8	36.3	59	60
Avg. Cycle	33.3	34.8	24.7	6.6	5.8

Table 5-2
Comparative performance between coordination configurations

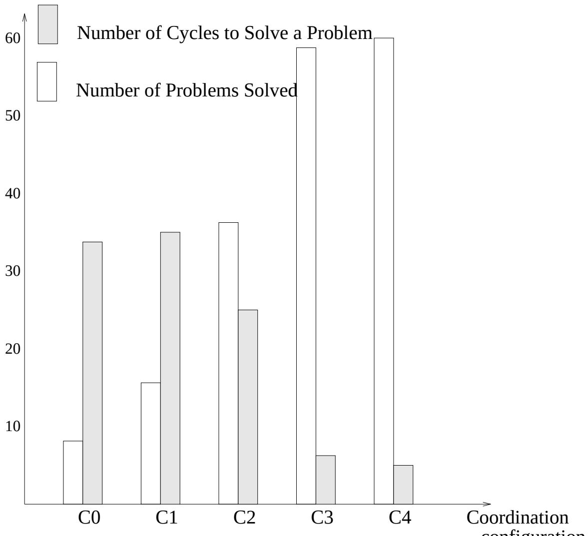

Figure 5-1.
Comparative performance graph between coordination configurations

The above Table and Figure show the comparative performance of different configurations on the suite of benchmark problems. In the table, the additional coordination information for each configuration is underlined. The number of cycles that the system was allowed was limited to 100. If there were still conflicts at cycle 100, the system gave up solving the problem. Since system operations in C0, C1, and C2 have random nature, they were ran on each problem 10 times. The numbers reported are the average number, e.g. 15.8 out of 60 problems were solved means that there were 158 successful runs among 600 (10 runs for each problem). C3 and C4 are deterministic and for these each problem was tried only once. The results show that more coordination information enables the system to solve more problems within fewer cycles. C0 was only able to solve 8 problems with an average of 33.3 cycles for solution evolution (4.7 CPU seconds), while C4 was able to solve all 60 problems with an average of 5.8 cycles (2.9 CPU seconds). A slight increase in the average cycles in C1 (compared to C0) stems from the fact that while C1 was able to solve twice the number of problems than C0 due to boundary information during initial allocation, it had no other advantages over C0 to resolve subsequent conflicts. With additional information, C2 was able to solve double the number of problems solved in C1 in fewer cycles. C3 solved almost all 60 problems in considerably fewer (6.6) cycles than C2. Only one problem, which is a 2-bottleneck problem, was unsolved by C3 within 100 cycles. By adding initial coordination between Bottleneck Resource Agents, C4 solved all 60 problems in 5.8 cycles. The results show the utility of coordination information.

As shown in Figures 4-1 and 4-2, the number of activities involved in conflicts in each cycle typically drops very fast within the first few cycles. After the drop, the problem solving process either solves the problem immediately or encounters a number of oscillations before finally solving the problem. Figure 5-1 shows, for different coordination configurations, the overall problem solving processes in terms of the number of activities involved in conflicts at each cycle. As the coordination information increases, the shape of the curve indicates a steeper drop in the number of conflicts in fewer cycles.

Curves for deterministic C3 and C4 have peaks at cycle 5. This reveals that when the problem was not solved within the first few cycles, an escape from oscillation typically occurred.

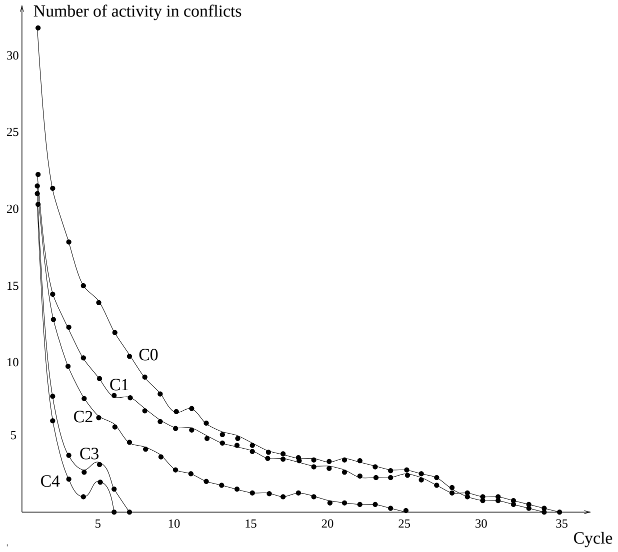

Figure 5-1: Comparison of successful solution evolution among different coordination configurations

6. Conclusions

We have presented an approach to distributed constraint satisfaction based on partitioning the problem constraints into constraint types. Responsibility for enforcing constraints of a particular type is given to specialist agents. The agents coordinate to iteratively change the instantiation of variables under their jurisdiction according to their specialized perspective. We demonstrated the effectiveness of the approach in the domain of job shop scheduling. Experimental results on a suite of benchmark problems showed that the approach outperformed other methods. The power of our approach stems from the types of coordination information that the agents utilize.

References

[1] J. Adams, E. Balas, and D. Zawack.
The Shifting Bottleneck Procedure for Job Shop Scheduling.
Management Science 34, 1988.

[2] D. Applegate and W. Cook.
A Computational Study of Job-Shop Scheduling.
Technical Report CMU-CS-90-145, School of Comoputer Science, Carnegie-Mellon University, 1990.

[3] Peter Burke and Patrick Prosser.
A Distributed Asynchronous System for Predictive and Reactive Scheduling.
Technical Report AISL-42, Department of Computer Science, University of Strathclyde, October, 1989.

[4] M.S. Fox and S.F. Smith.
ISIS: A Knowledge-Based System for Factory Scheduling.
Expert Systems 1(1):25-49, 1984.

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

[6] Huhns, M., Bridgeland, D.
Distributed Truth Maintenance.
In *Proceedings of the 10th International Workshop on DAI*. Bandera, Texas, 1990.

[7] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan.
Recent Developments in Deterministic Sequencing and Scheduling: A Survey.
In M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan (editor), *Deterministc and Stochastic Scheduling*. Reidel, 1982.

[8] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird.
Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.
In *Proceedings of the Eigth National Conference on Artificial Intelligence*. 1990.

[9] Kazuo Miyashita and Katia Sycara.
Case-Based Incremental Schedule Revision.
In M. Fox and M. Zweben (editor), *Knowledge-Based Scheduling*. Morgan Kaufmann, 1993.

[10] Nicola Muscettloa.
Scheduling by Iterative Partition of Bottleneck Conflicts.
Technical Report CMU-RI-TR-92-05, Robotics Institute, Carnegie-Mellon University, 1992.

[11] Norman Sadeh.
Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.
Technical Report CMU-CS-91-102, School of Computer Science, Carnegie-Mellon University, 1991.

[12] Stephen F. Smith, Peng Si Ow, Claude Lepape, Bruce Mclaren, Nicola Muscettola.
Integrating Multiple Scheduling Perspectives to Generate Detailed Production Plans.
In *Proceedings 1986 SME Conference on AI in Manufacturing*, pages 123-137. 1986.

[13] Sycara, K., Roth, S., Sadeh, N., and Fox, M.
Distributed Constrained Heuristic Search.
IEEE Transactions on System, Man and Cybernetics 21(6):1446-1461, 1991.

[14] Yalin Xiong, Norman Sadeh, and Katia Sycara.
Intelligent Backtracking Techniques for Job Shop Scheduling.
In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 14-23. 1992.

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

[16] M. Zweben, M. Deale and R. Gargan.
Anytime Rescheduling.
In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*. 1990.

Table of Contents

List of Figures