Emergent Constraint Satisfaction through Multi-Agent Coordinated Interaction

JyiShane Liu and Katia Sycara

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
jsl@cs.cmu.edu and katia@cs.cmu.edu

To appear in *Decentralized Artificial Intelligence, IV*, Springer Verlag, 1995

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. Utility of different types of coordination information in CP&CR was investigated. In addition, experimental results on a benchmark suite of problems show that CP&CR performed considerably well as compared to other centralized search scheduling techniques, in both computational cost and number of problems solved.

1 Introduction

Distributed AI (DAI) has primarily focused on Cooperative Distributed Problem Solving [4] [8] by sophisticated agents that work together to solve problems that are beyond their individual capability. Another trend has been the study of computational models of agent societies [9], composed of simple agents that interact asynchronously. With few exceptions (e.g.[1] [5] [18]), these models have been used to investigate the evolutionary behavior of biological systems [10] [12] rather than the utility of these models in problem solving. We have developed a computational framework for problem solving by a society of simple interacting agents and applied it to solve job shop scheduling Constraint Satisfaction Problems (CSPs). Experimental results, presented in Sect. 3, show that the approach performs considerably well as compared to centralized search methods for a set of benchmark job shop scheduling problems. These encouraging results indicate good problem solving potential of approaches based on distributed agent interactions.

Many problems of theoretical and practical interest (e.g., parametric design, resource allocation, scheduling) can be formulated as CSPs. A CSP is defined by a set of *variables* $X = \{x_1, x_2, \cdots, x_m\}$, each having a corresponding domain of *values* $V = \{v_1, v_2, \cdots, v_m\}$, and a set of *constraints* $C = \{c_1, c_2, \cdots, c_n\}$. A constraint c_i is a subset of the Cartesian product $v_l \times \cdots \times v_q$ which specifies which values of the variables are compatible with each other. A solution to a CSP is an assignment of values (an instantiation) for all variables, such that all constraints are satisfied. In general, CSPs are solved by two complementary approaches, backtracking and network consistency algorithms [11][2][16]. Recently, heuristic revision [13] and decomposition [3][6] techniques for CSPs have been proposed.

Recent work in DAI has considered the distributed constraint satisfaction problem (DCSP) [22] [20] 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. 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 ([22]) or dynamic ([20]) variable and value ordering, and the final solution was generated by merging partial instantiations that satisfied the problem constraints. Instead, our approach decomposes a CSP by constraint type. This results in no inter-agent constraints, but each variable may be instantiated by more than one agent. While satisfying its own constraints, each agent instantiates/modifies variable values based on coordination information supplied by others. Coordination among agents facilitates effective problem solving.

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 constraints are satisfied. Since each variable may be restricted by more than one constraint type, 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 remainder of the paper is organized as follows. In Sect. 2, we describe an application of CP&CR to job shop scheduling with non-relaxable time windows. In Sect. 3, we evaluate experimental results on previously studied test problems. Finally, in Sect. 4, we conclude the paper and outline our current work on CP&CR.

2 Distributed Job Shop Scheduling by CP&CR

Job shop scheduling with non-relaxable time windows involves synchronization of the completion of a number of jobs on a limited set of resources (machines). Each job is composed of a sequence of activities (operations), each of which has a specified processing time and requires the exclusive use of a designated resource for the duration of its processing (i.e. resources have only unit processing capacity). Each job must be completed within an interval (a time window) specified by its release and due time. A solution of the problem is a schedule, which assigns start times to each activity, that satisfies all *temporal activity precedence*, *release and due date*, and *resource capacity* constraints. This problem is known to be NP-complete [7], and has been considered as one of the most difficult CSPs. Traditional constraint satisfaction algorithms are shown to be insufficient for this problem [17].

CP&CR views each activity as a *variable* with a *value* corresponding to the start time of the activity. Dominant constraints in job shop scheduling are partitioned into two categories: temporal precedence[1] and resource capacity constraints. Within each constraint type, subproblems are formulated. Each subproblem is assigned to a separate agent. In particular, enforcing temporal precedence constraints within an job is a subproblem that is assigned to an *job agent*; enforcing capacity constraints on a given resource is a subproblem that is under the responsibility of a *resource 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 jobs plus the number of resources. An activity is governed both by a job agent and a resource agent. Manipulation of activities by job 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.

2.1 A Society of Reactive Agents

In CP&CR, problem solving of a job shop scheduling CSP is transformed into collective behaviors of reactive agents. Each agent examines and makes changes to only local activities under its responsibility, and seeks for satisfaction by

[1] Release and due dates constraints are considered as temporal precedence constraints between activities and fixed time points.

ensuring that no constraint in its assigned constraint subset is violated. Agents are equipped with only primitive behavior. When activated, each agent goes through an Examine-Resolve-Encode cycle (see Fig. 1). 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 variable instantiations to resolve conflicts according to innate heuristics and coordination information (see Sect. 2.2 and 2.3).

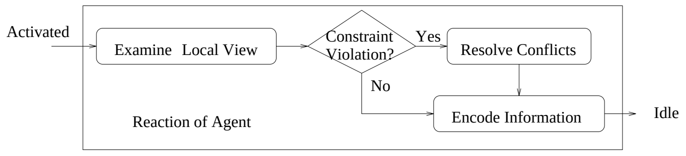


Fig. 1. Agent's reactive behavior

Agents coordinate by *passive* communication. They do not communicate with each other directly. Instead, each agent reads and writes coordination information on variables under its jurisdiction. Coordination information on a variable represents an agent's partial "view" on the current solution and is consulted when other agents are considering changing the current instantiation of the variable to resolve their conflicts. Each time an agent is activated and has ensured its satisfaction, it writes down its view on current instantiations on each variable under its jurisdiction as coordination information.

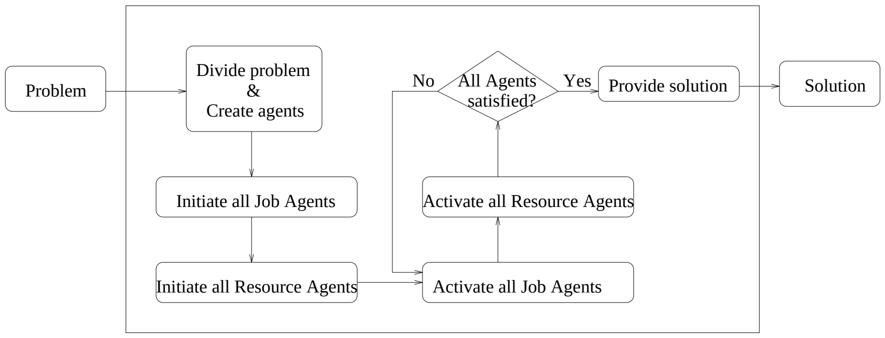


Fig. 2. System Control Flow

System initialization is done as follows: (1) decomposition of the input scheduling problem according to resource and job constraints, (2) creation of the corresponding resource and job agents, (3) activation of the agents (see Fig. 2). Initially each job agent calculates boundary[2] for each activity under its juris-

[2] The boundary of an activity is defined as the interval between its earliest possible

diction considering its release and due date constraints. 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 concludes that it is a bottleneck resource agent.[3] [4]

Activities under the jurisdiction of a bottleneck resource agent are marked as *bottleneck activities* by the agent. Each resource agent heuristically allocates the earliest free resource interval to each activity under its jurisdiction according to each activity's boundary. After the initial activation of resource agents, all activities are instantiated with a start time. This initial instantiation of all variables represents the initial configuration of the solution.[5]

Subsequently, job agents and resource agents engage in an evolving process of reacting to constraint violations and making changes to the current instantiation. In each *iteration cycle*, job and resource agents are activated alternatively, while agents of the same type are activated simultaneously, each working independently. When an agent finds constraint violations under its jurisdiction, it employs local reaction heuristics to resolve the violations. The process stops when none of the agents detect constraint violations during an iteration cycle. The system outputs the current instantiation of variables as a solution to the problem.

2.2 Coordination Information

Coordination information *written* by a job agent on an activity is referenced by a resource agent, and vice-versa.

Job agents provide the following coordination information for resource agents.

1. *Boundary* is the interval between the earliest start time and latest finish time of an activity (see Fig. 3). It represents the overall temporal flexibility of an activity and is calculated only once during initial activation of job agents.
2. *Temporal Slack* is an interval between the current finish time of the previous activity and current start time of the next activity (see Fig. 3). It indicates the temporal range within which an activity may be assigned to without causing temporal constraint violations. (This is not guaranteed since temporal slacks of adjacent activities are overlapping with each other.)

start time and its latest possible finish time.

[3] If no bottleneck resource is identified, threshold value is lowered until the most contended resource is identified.

[4] In job shop scheduling, the notion of bottleneck corresponds to a particular resource interval demanded by activities that exceeds the resource's capacity. Most state-of-the-art techniques emphasize the capability to identify *dynamic* bottlenecks that arise during the construction of solution. In our approach, the notion of bottleneck is *static* and we exploit the dynamic local interactions of agents.

[5] We have conducted experiments with random initial configurations and confirmed that CP&CR is barely affected by its starting point, i.e. CP&CR has equal overall performance with heuristic and random initial configurations.

3. *Weight* is the weighted sum of relative temporal slack with respect to activity boundary and relative temporal slack with respect to the interval bound by the closest bottleneck activities (see Fig. 4). It is a measure of the likelihood of the activity "bumping" into an adjacent activity, if its start time is changed. Therefore, a high weight represents a job agent's preference for not changing the current start time of the activity. In Fig. 4, activity-p of job B will have a higher weight than that of activity-a of job A. If both activities use the same resource and are involved in a resource capacity conflict, the resource agent will change the start time of activity-a rather than start time of activity-p.

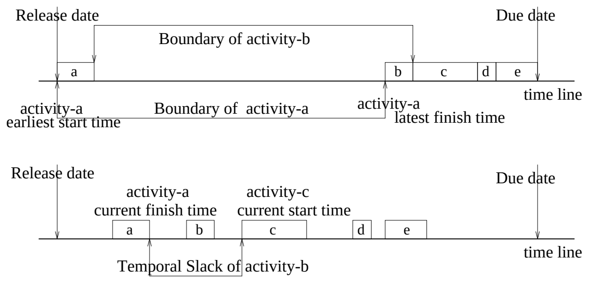


Fig. 3. Coordination information: Boundary and Temporal Slack

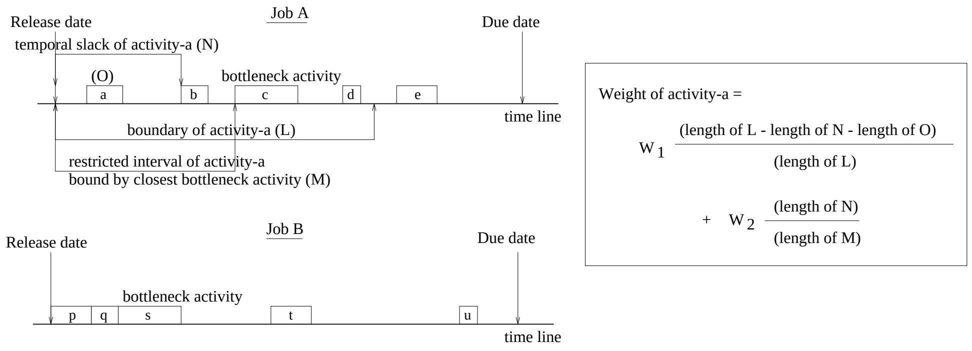

Job A
Release date
Due date
temporal slack of activity-a (N)
(O)
bottleneck activity
a
b
c
d
e
time line
boundary of activity-a (L)
restricted interval of activity-a
bound by closest bottleneck activity (M)
Weight of activity-a =
W1 (length of L - length of N - length of O) / (length of L)
+ W2 (length of N) / (length of M)
Job B
Release date
Due date
bottleneck activity
p
q
s
t
u
time line

Fig. 4. Coordination information: Weight

Resource agents provide the following coordination information for job agents.

1. *Bottleneck Tag* is 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 job agent should treat these activities differently.

2. *Resource Slack* is 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 Fig. 5). It indicates the range of activity start times in which an activity may be changed without causing capacity constraint violations. (There is no guaranteed since resource slacks of adjacent activities are overlapping with each other.)
3. *Change Frequency* is a counter of how frequently the start time of this regular activity set by a job agent is changed by a submissive resource agent. It measures the search effort of job and regular resource agents to evolve an instantiation on regular activities that is compatible to the current instantiation on bottleneck activities.

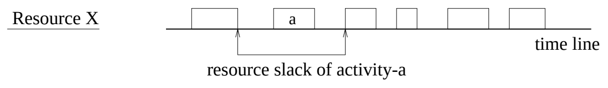


Fig. 5. Coordination information: Resource Slack

2.3 Reaction Heuristics

Agents' reaction heuristics utilize perceived coordination information and incorporate coordination strategies of group behaviors. We have developed coordination strategies that promote rapid convergence by minimizing the ripple effects of causing conflicts to other agents as a result of an agent's fixing the current constraint violations. Conflict minimization is achieved by minimizing the number and extent of activity start time changes. Other coordination strategies include using most constrained agents as an anchor of interaction, and preventing oscillatory value changes.

Reaction Heuristics of Job Agent. Job agents resolve conflicts by considering conflict pairs. A conflict pair involves two adjacent activities whose current start times violate the precedence constraint between them (see Fig. 6). Conflict pairs are resolved one by one. A conflict pair involving a bottleneck activity, i.e., an activity with tighter constraints, is given a higher conflict resolution priority. To resolve a conflict pair, job agents essentially determine which activity's current start time should be changed. If a conflict pair includes a bottleneck and a regular activity, depending on whether the change frequency counter on the regular activity in the conflict pair is still under a threshold, job agents change the start time of either the regular or the bottleneck activity. For conflict pairs of regular activities, job agents take into consideration additional factors, such as value changes feasibility of each activity, change frequency, and resource slack.

In Fig. 6, the conflict pair of activity-A2 and activity-A3 will be resolved first since activity-A2 is a bottleneck activity. If the change frequency of activity-A3 is still below a threshold, start time of activity-A3 will be changed by an addition of T2 (the distance between current start time of activity-A3 and current end time of activity-A2) to its current start time. Otherwise, start time of activity-A2 will be changed by a subtraction of T2 from its current start time. In both cases, start time of activity-A4 will be changed to the end time of activity-A3. To resolve the conflict pair of activity-A0 and activity-A1, either start time of activity-A0 will be changed by a subtraction of T1 from its current start time or start time of activity-A1 will be changed by an addition of T1 to its current start time. If one of the two activities can be changed within its boundary and resource slack, job agent A will change that activity. Otherwise, job agent A will change the activity with less change frequency.

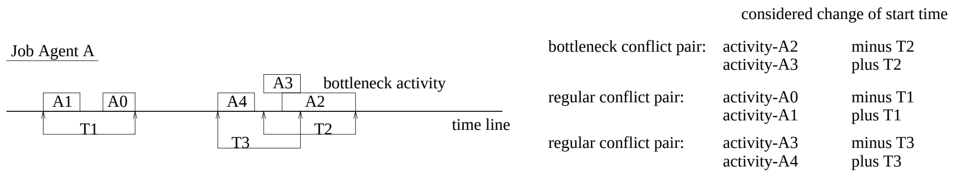


Fig. 6. Conflict Resolution of Job Agent

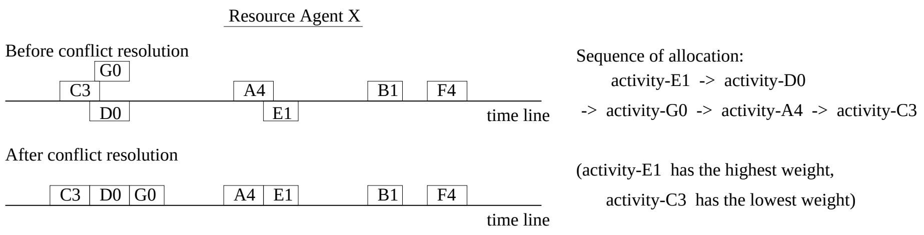

Resource Agent X
Before conflict resolution
G0
C3
A4
B1
F4
D0
E1
time line
After conflict resolution
C3
D0
G0
A4
E1
B1
F4
time line
Sequence of allocation:
activity-E1 -> activity-D0
-> activity-G0 -> activity-A4 -> activity-C3
(activity-E1 has the highest weight,
activity-C3 has the lowest weight)

Fig. 7. Conflict Resolution of Regular Resource Agent

Reaction Heuristics of Regular Resource Agents. To resolve constraint violations, resource agents re-allocate the over-contended resource intervals to the competing activities in such a way as to resolve the conflicts and, at the same time, keep changes to the start times of these activities to a minimum. Activities are allocated according to their weights, boundaries, and temporal slacks. Since an activity's weight is a measure of the desire of the corresponding job agent to keep the activity at its current value, activity start time decisions based on weight reflect group coordination. For example, in Fig. 7, activity-A4 was preempted by activity-E1 which has higher weight. Start time of activity-A4 is changed as little as possible. In addition, when a resource agent perceives a high resource contention during a particular time interval (such as the conflict involving activity-C3, activity-D0, and activity-G0), it allocates the resource intervals and assigns high change frequency to these activities, and thus dynamically changes the priority of these instantiation.

Reaction Heuristics of Bottleneck Resource Agents. A bottleneck resource agent has high resource contention. This means that most of the time a bottleneck resource agent does not have resource slack between activities. When the start time of a bottleneck activity is changed, capacity constraint violations are very likely to occur. A bottleneck resource agent considers the amount of overlap of activity resource intervals on the resource to decide whether to right-shift some activities (Fig. 8 (i)) or re-sequence some activities according to their current start times by swapping the changed activity with an appropriate activity. (Fig. 8 (ii)). The intuition behind the heuristics is to keep the changes as minimum as possible.

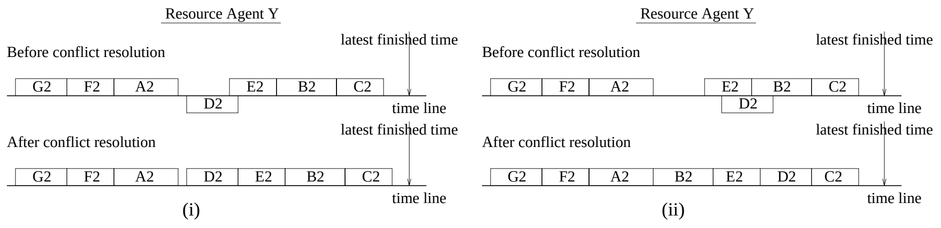


Fig. 8. Conflict Resolution of Bottleneck Resource Agent

2.4 Solution Evolution

Figure 9 shows a solution evolution process of a very simple problem where resource Y is regarded as a bottleneck resource. In (a), resource agents allocate their earliest possible free resource intervals to activities, and thus construct the initial configuration of variable instantiation. In (b), A13, A23 within dotted rectangular boxes represent the start times assigned by resource agents Res.X and Res.Z, respectively. Job1 and Job2 agents are not satisfied with current instantiation because the pairs of (A12 A13) and (A22 A23) are violating their precedence constraints. Job1 (cf. Job2) agent changes the start times of A13 (cf. A23) (shown by solid rectangular box) because A12 (cf. A22) is a seed activity and change frequency of A13 (cf. A23) is zero (have not exceed the threshold). In (c), Res.Z agent finds a capacity constraint violation between A23 and A33 (shown by dotted rectangular box before conflict resolution), and changes the start time of A33 because A23 has a higher weight. All agents are satisfied with the current instantiation of variables in (d) which represents a solution to the problem.

Figure 10 shows a solution evolution process in terms of occurred conflicts for a more difficult problem which involves 10 jobs on 5 resources. In cycle 0, resource agents construct an initial instantiation of variables that includes start times of bottleneck activities by bottleneck resource agents. During cycle 1 to cycle 9, job agents and regular resource agents try to evolve a compatible instantiation of regular activities with the instantiation of bottleneck activities. In cycle 10, some job agents perceive the effort as having failed (by observing the change frequency counter on a regular variable in conflict with a bottleneck activity has exceeded the threshold) and change the values of their bottleneck activities. Bottleneck resource agents respond to constraint violations by modifying instantiation of the bottleneck activities. This results in a sharp increase of conflicting activities for job agents in cycle 11. Again, the search for compatible instantiation resumes until another modification on instantiation of the bottleneck activities in cycle 16. In cycle 18, the solution is found.

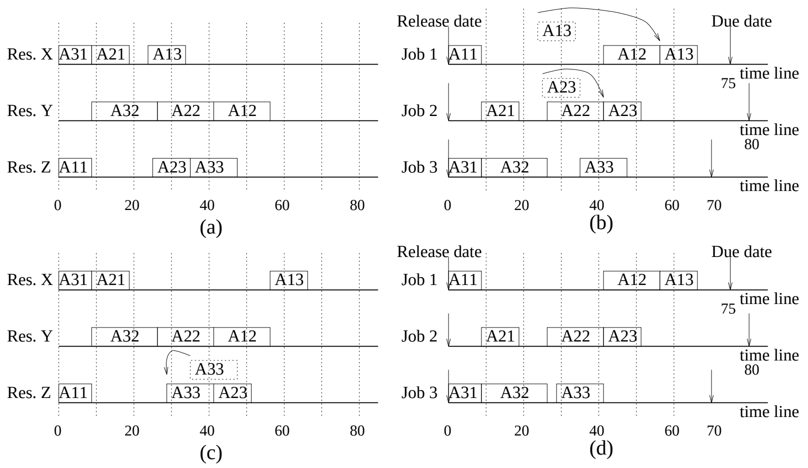


Fig. 9. A Simplified Scenario

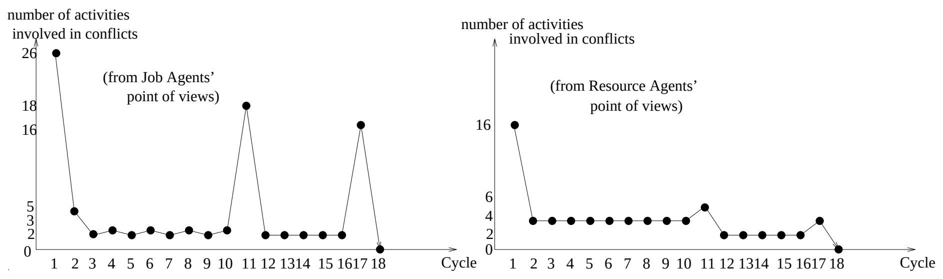


Fig. 10. Conflicts Evolution of a more difficult problem

3 Evaluation on Experimental Results

We evaluated the performance of CP&CR on a suite of job shop scheduling CSPs proposed in [17]. The benchmark consists of 6 groups, representing different scheduling conditions, of 10 problems, each of which has 10 jobs of 5 activities and 5 resources. Each problem has at least one feasible solution. CP&CR has been implemented in a system, called CORA (COordinated Reactive Agents). We experimentally (1) investigated the effects of coordination information in the system, (2) compared CORA's performance to other constraint-based as well as priority dispatch scheduling methods.

3.1 Effects of Coordination Information

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

- C0 represents a configuration in which the system ran with no coordination information at all. Without boundary information, when initially activated, resource agents allocate resource intervals according to random sequences. When job agents are activated, they resolve conflicts by randomly changing the instantiation of one of the two activities in each conflict pair. Similarly, resource agents resolve conflicts based on random priority sequences.
- C1 represents a configuration in which only boundary information is available. 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. Resource agents use the boundary information for heuristic initial allocation of resource intervals. Job agents use the bottleneck tag information to bias resolution of conflict pairs.
- C3 represents a complete configuration in which all coordination information is provided for resource agents and job agents.

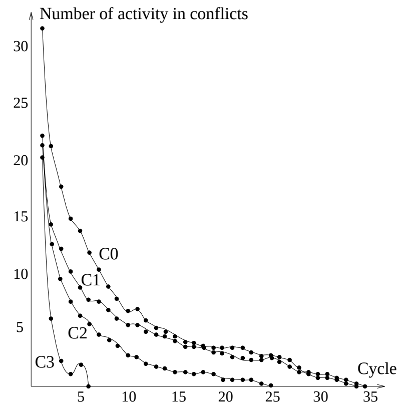
Figure 11 shows the comparative performance of different configurations on the suite of benchmark problems. The additional coordination information for each configuration is underlined in Fig. 11 (i). The number of cycles that the system was allowed was limited to 100. If there were still conflicts at 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 is deterministic and for it each problem was tried only once. We confirm that adding coordination information enables the system to solve more problems within fewer cycles. The results shows the utility of coordination information.

Figure 11 (ii) shows, for different coordination configurations, the successful overall problem solving processes[6] 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. This indicates that increasing rates of convergence are facilitated by more coordination information. The curve for deterministic C3 has a peak at cycle 5. This reveals that when the problem was not solved within the first few cycles, an instantiation modification on the activities using bottleneck resources typically occurred. The curves for C0, C1, and C2 do not exhibit a peak because the system does not have particular pattern of interaction in those coordination configurations.

C0	No coordination information
C1	Boundary (heuristic initial allocation)
C2	Boundary + Bottlenck tag
C3	Boundary + Temporal slack + Weight Bottleneck tag + Resource slack + Change frequency

Overall Performance	Coordination configuration			
	C0	C1	C2	C3
No. of Porb. Solved(Avg.)	8.0	15.8	36.3	60
Avg. Cycle	33.3	36.3	24.7	5.2

(i)



(ii)

Fig. 11. Comparative Performance between Coordination Configurations

3.2 Comparison with Other Scheduling Techniques

CORA was compared to four other heuristic search scheduling techniques, ORR /FSS, MCIR, CPS, and PCP. ORR/FSS [17] incrementally constructs a solution by chronological backtracking search guided by specialized variable and value ordering heuristics. ORR/FSS+ is an improved version augmented with an intelligent backtracking technique [21]. Min-Conflict Iterative Repair (MCIR) [13] starts with an initial, inconsistent solution and searches through the space of possible repairs based on a *min-conflicts* heuristic which attempts to minimize the number of constraint violations after each step. Conflict Partition Scheduling (CPS) [15] employs a search space analysis methodology based on stochastic simulation which iteratively prunes the search space by posting additional

[6] For C0, C1, and C2, only successful overall problem solving processes are averaged and shown.

constraints. Precedence Constraint Posting (PCP) [19] conducts the search by establishing sequencing constraints between pairs of activities using the same resource based on *slack-based* heuristics. In addition, three frequently used and appreciated priority dispatch rules from the field of Operations Research: EDD, COVERT, and R&M [14], are also included for comparison.

Table 1 reports the number of problems solved[7] and the average CPU time spent over all the benchmark problems for each technique. Note that the results of ORR/FSS, ORR/FSS+, MCIR, CPS, and PCP were obtained from published reports, of mostly the developers of the techniques. MCIR is the only exception, which is implemented by Muscettola who reported its results based on randomly generated initial solutions [15]. All CPU times were obtained from Lisp implementations on a DEC 5000/200. In particular, CORA was implemented in CLOS (Common Lisp Object System). CPS, MCIR, ORR/FSS, and ORR/FSS+ were implemented using CRL (Carnegie Representation Language) as an underlying frame-based knowledge representation language. CPU times of CPS, MCIR, ORR/FSS, and ORR/FSS+ were divided by six from the published numbers as an estimate of translating to straight Common Lisp implementation.[8] PCP's CPU times are not listed for comparison because its CPU times in Lisp are not available. Its reported CPU times in C are 0.3 second [19]. Although CORA can operate asynchronously, it was sequentially implemented for fair comparison. The results show that CORA works considerably well as compared to the other techniques both on feasibility and efficiency in finding a solution.

	CORA	CPS	MCIR	ORR/ FSS	ORR/ FSS+	PCP	EDD	COVERT	R&M
w/1	10	10	9.8	10	10	10	10	8	10
w/2	10	10	2.2	10	10	10	10	7	10
n/1	10	10	7.4	8	10	10	8	7	9
n/2	10	10	1	9	10	10	8	6	9
0/1	10	10	4.2	7	10	10	3	4	6
0/2	10	10	0	8	10	8 ~ 10	8	8	8
Total	60	60	24.6	52	60	58 ~ 60	47	40	52
AVG. CPU time	4.8 seconds	13.07 * seconds	49.74 * seconds	39.12 * seconds	21.46 * seconds	N/A	0.9 seconds	0.9 seconds	0.9 seconds

Table 1. Performance Comparison

3.3 Evaluation

As a scheduling technique, CORA performs a heuristic *approximate* search in the sense that it does not systematically try all possible configurations. Although

[7] PCP's performance is sensitive to the parameters that specify search bias [19].

[8] ORR/FSS and ORR/FSS+ obtained 30 times speedup in C/C++ implementation. We assumed a factor of five between Common Lisp and C/C++ implementations.

there are other centralized scheduling techniques that employ similar search strategies, CORA distinguishes itself by an interaction driven search mechanism based on well-coordinated asynchronous local reactions. Heuristic approximate search provides a middle ground between the generality of domain-independent search mechanisms and the efficiency of domain-specific heuristic rules. Instead of the rigidity of one-pass attempt in solution construction (either it succeeds or fails, and the decisions are never revised) in approaches using heuristic rules, CORA adapts to constraint violations and performs an effective search for a solution. As opposed to generic search approaches, in which a single search is performed on the whole search space and search knowledge is obtained by analyzing the whole space at each step, CORA exploits local interactions by analyzing problem characteristics and conducts well-coordinated asynchronous local searches.

The experimental results obtained by various approaches concur with the above observations. Approaches using generic search techniques augmented by domain-specific search-focus heuristics (ORR/FSS, ORR/FSS+, MCIR, CPS) required substantial amount of computational effort. Some of them could not solve all problems in the sense that they failed to find a solution for a problem within the time limit set by their investigators. Approaches using dispatch rules (EDD, COVERT, R&M) were computationally efficient, but did not succeed in all problems. PCP relies on heuristic rules to conduct one-pass search and its performance is sensitive to parameters that specify search bias. CORA struck a good balance in terms of solving all problems with considerable efficiency. Furthermore, with a mechanism based on collective operations, CORA can be readily implemented in parallel processing such that only two kinds of agents are activated sequentially in each iteration cycle, instead of 10 job agents and 5 resource agents under current implementation. This would result in an approximate time-reducing factor of 7 (i.e., 15/2) and would enable CORA to outperform all other scheduling techniques in comparison.

4 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. The final solution *emerges* through incremental local revisions of an initial, possibly inconsistent, instantiation of all variables. We demonstrated the effectiveness of the approach in the domain of job shop scheduling. The power of our approach stems from the coordinated local interactions of the reactive agents. We are currently formalizing the CP&CR methodology and extending it to Constraint Optimization Problems (COPs). We are also investigating the utility of CP&CR in other domains with different problem structures.

References

1. Brooks, R.: Intelligence Without Reason. In Proceedings of the IJCAI-91. (1991) 569–595
2. Dechter, R.: Network-based heuristics for constraint satisfaction problems. Artificial Intelligence. **34** (1988) 1–38
3. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence. **49** (1991) 61–95
4. Decker, K.: Distributed problem-solving techniques: A survey. IEEE Transactions on Systems, Man, and Cybernetics. **17** (1987) 729–739
5. Ferber, J., Jacopin, E.: The framework of Eco problem solving. In Demazeau and Muller, editors, Decentralized AI II. (1991) Elsevier, North-Holland.
6. Freuder, E., Hubbe, P.: Using inferred disjunctive constraints to decompose constraint satisfaction problems. In Proceedings of the IJCAI-93. (1993) 254–260
7. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. (1979) Freeman and Co.
8. Gasser, L., Hill, R., Jr.: Engineering coordinated problem solvers. Annual Review of Computer Science. **4** (1990) 203–253
9. Langton, C., editor: Artificial Life. (1989) Addison-Wesley.
10. Langton, C., Taylor, C., Farmer, J., Rasmussen, S., editors: Artificial Life II. (1991) Addison-Wesley.
11. Mackworth, A.: Constraint satisfaction. In Shapiro, S., editor, Encyclopedia in Artificial Intelligence. (1987) 205–211. Wiley, New York.
12. Meyer, J.-A., Wilson, S., editors: Proceedings of the First International Conference on Simulation of Adaptive Behavior - From Animals To Animats. (1991) MIT Press.
13. Minton, S., Johnston, M., Philips, A., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence. **58** (1992) 161–205
14. Morton, T., Pentico, D.: Heuristic Scheduling Systems: With Applications to Production Systems and Project Management. (1993) John Wiley & Sons, New York.
15. Muscettola, N.: HSTS: Integrated planning and scheduling. In Fox, M., Zweben, M., editors. Knowledge-Based Scheduling. (1993) Morgan Kaufmann.
16. Nadel, B.: Constraint satisfaction algorithms. Computational Intelligence. **5** (1989) 188–224
17. Sadeh, N.: Look-ahead techniques for micro-opportunistic job shop scheduling. Technical report CMU-CS-91-102, School of Computer Science, Carnegie-Mellon University. (1991)
18. Shoham, Y., Tennenholtz, M.: On the synthesis of useful social laws for artificial agent societies. In Proceedings of AAAI-92. (1992) 276–281
19. Smith, S., Cheng, C.: Slack-based heuristics for constraint satisfaction scheduling. In Proceedings of AAAI-93. (1993) 139–144
20. Sycara, K., Roth, S., Sadeh, N., Fox, M.: Distributed constraint heuristic search. IEEE Transactions on System, Man, and Cybernetics. **21** (1991) 1446–1461
21. Xiong, Y., Sadeh, N., Sycara, K.: Intelligent backtracking techniques for job shop scheduling. In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning. (1992) 14–23
22. Yokoo, M., Ishida, T., Kuwabara, K.: Distributed constraint satisfaction for DAI problems. In Proceedings of the 10th International Workshop on Distributed AI. (1990)