In the Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (1994)

Distributed Problem Solving through Coordination in a Society of Agents

JyiShane Liu jsl@cs.cmu.edu
Katia Sycara katia@cs.cmu.edu
Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a methodology, called Constraint Partition and Coordinated Reaction (CP&CR), where a problem solution emerges from the evolving computational process of a group of diverse, interacting, and well-coordinated reactive agents. Problem characteristics are utilized to achieve problem solving by asynchronous and well coordinated local interactions. The coordination mechanisms guide the search space exploration by the society of interacting agents, facilitating rapid convergence to a solution. Our domain of problem solving is constraint satisfaction. We have applied the methodology to job shop scheduling with non-relaxable time windows, 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 [Decker, 1987] [Durfee, 1988] [Gasser and Hill, Jr., 1990] 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 [Langton, 1989], composed of simple agents that interact asynchronously. With few exceptions (e.g. [Brooks, 1991][Ferber and Jacopin, 1991] [Shoham and Tennenholtz, 1992]), these models have been used to investigate the evolutionary behavior of biological systems [Langton *et al.*, 1991] [Meyer and Wilson, 1991] 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 section 4, 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* $V = \{v_1, v_2, \cdots, v_m\}$, 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. The *variable set* of a constraint (or a set of constraints), denoted by *vs( )*, is the set of non-duplicating variables restricted by the constraint (or the set of constraints). A solution to a CSP is an assignment of values (an instantiation) for all variables, such that all constraints are satisfied. Numerical CSPs

(NCSPs) [Lhomme, 1993] are a subset of CSPs, in which constraints are represented by numerical relations between quantitative variables usually with fairly large domains of possible values. Many CSPs of practical importance, such as scheduling, and parametric design, are NCSPs. Constraint satisfaction algorithms typically suffer from feasibility/efficiency problems for NCSPs due to their enormous search spaces.

In general, CSPs are solved by two complementary approaches, backtracking and network consistency algorithms [Mackworth, 1987][Dechter, 1988][Nadel, 1989]. Recently, heuristic revision [Minton *et al.*, 1992] and decomposition [Dechter *et al.*, 1991][Freuder and Hubbe, 1993] techniques for CSPs have been proposed. On the other hand, recent work in DAI has considered the distributed CSPs [Huhns and Bridgeland, 1991] [Sycara *et al.*, 1991] [Yokoo *et al.*, 1992] in which variables of a CSP are distributed among agents. Each agent has an *exclusive* subset of the variables and has sole responsibility to instantiate their values. Instead, our approach provides a decomposition scheme in which constraint type as well as constraint connectivity are used. 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 a job shop scheduling NCSP is collectively solved by a set of agents with simple local reactions and effective coordination. CP&CR divides an NCSP into a set of subproblems according to constraint type and assigns each subproblem to an agent. Interaction characteristics among agents are identified. Agent coordination defines agent roles, the information they exchange, and the rules of interaction. The problem solution emerges as a result of the evolving process of the group of interacting and coordinating agents. The remainder of the paper is organized as follows. In Section 2, we define the CP&CR model, in which problem decomposition, coordination mechanisms, and asynchronous search procedure are specified. In Sections 3 and 4, we describe an application of CP&CR to job shop scheduling with non-relaxable time windows, and present comparative results on previously studied test problems. Finally, in Section 5, we conclude the paper and outline our current work on CP&CR.

2 CP&CR Model

CP&CR is a problem-solving framework in which a society of specialized and well-coordinated agents collectively solve a problem. Each agent reacts to others' actions and communicates with others by leaving and perceiving particular messages on the objects it acts on. A solution emerges from the evolutionary interaction process of the society of diverse agents. Specifically, CP&CR provides a framework to decompose an NCSP into a set of subproblems based on constraint type and constraint connectivity, identify their interaction characteristics and, accordingly construct effective coordination mechanisms. CP&CR assumes that an NCSP has at least two types of constraints.

2.1 Constraint Partition & Problem Decomposition

Constraints label relations between variables that specify how the values of variables are restricted for compatibility. We formally define constraint characteristics (e.g., constraint type, constraint connectivity) for NCSPs.

Definition 1: Constraint Type - In CP&CR, quantitative constraints are classified by differences in the numerical compatibility between two variables. We identify four types of quantitative

constraints. In Figure 1, a black dot represents a value, v_i, that has been assigned to a variable, x_i. An empty dot represents the only possible value for the other variable, x_j. A shaded region (either open or closed) represents an interval within which an instantiation of the other variable, x_j, will violate the constraint.

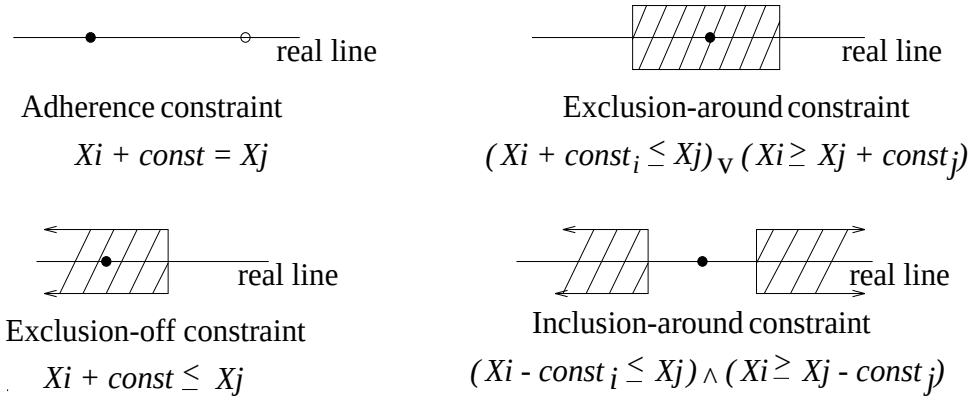


Figure 1: Constraint types classification

1. *adherence type* - A constraint is of adherence type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, to an individual point in the domain. For example, $x_i + const = x_j$.

2. *exclusion-around type* - A constraint is of exclusion-around type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, from a subsection within certain distances from v_i. For example, $x_i + const \neq x_j$, or $(x_i + const_i \leq x_j) \vee (x_i \geq x_j + const_j), const_i, const_j > 0$.

3. *exclusion-off type* - A constraint is of exclusion-off type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, from a connected subsection of the domain with a boundary set by v_i. For example, $x_i + const \leq x_j$.

4. *inclusion-around type* - A constraint is of inclusion-around type if the instantiation of a variable, x_i, to the value v_i restricts the instantiation of another variable, x_j, within a connected subsection of the domain with boundaries set by v_i. For example, $(x_i - const_i \leq x_j) \wedge (x_i \geq x_j - const_j), const_i, const_j > 0$.

We illustrate how our definitions can describe the constraints of some well known CSPs. In the N-Queen problem, both vertical and diagonal attack constraints are of exclusion-around type. In the Zebra problem, association constraints (e.g. the Englishman lives in the red house.) are of adherence type, and single-occupancy constraints (e.g. each attribute, such as pet, color, etc., must be assigned to each house.) are of exclusion-around type.

Definition 2: Constraint Connectivity - Two constraints are said to be *connected* iff the intersection of their variable sets is not empty. This implies that they have constrained variables in common.

c_p and c_q are connected $\equiv vs(c_p) \cap vs(c_q) \neq \emptyset$.

Definition 3: Constraint Partition is a scheme to decompose an NCSP into a set of subproblems by constraint type and constraint connectivity (see Figure 2). Two types of constraint grouping, *constraint bunch*, and *constraint cluster*, are introduced by the decomposition scheme.

A constraint bunch, C_i, is a set of constraints of the same constraint type. Define an operator, *pb( )*, which partitions the constraint set C of an NCSP into a set of constraint bunches, C_i, according to constraint type. Denote the resulting set of constraint bunches by C'. Define an operator, denoted by *type( )*, which identifies the constraint type of a constraint bunch. A constraint bunch has the following properties.

- $pb(C) = \{C_i\} = C'$
- C_i *partition* C
- $type(C_i) \neq type(C_j), i \neq j$

A constraint cluster, $C_{i,m}$, is a set of constraints which are of the same constraint type and are connected to each other. Define an operator, *pc( )*, which partitions a set of constraint bunches into a set of constraint clusters, C'', according to constraint connectivity. A constraint cluster has the following properties.

- $pc(C') = \{C_{i,m}\} = C''$
- *Constraint clusters of the same constraint type have no variables in common*
- *If a constraint cluster contains more than one constraint, each constraint is connected to at least one other constraint in the constraint cluster*

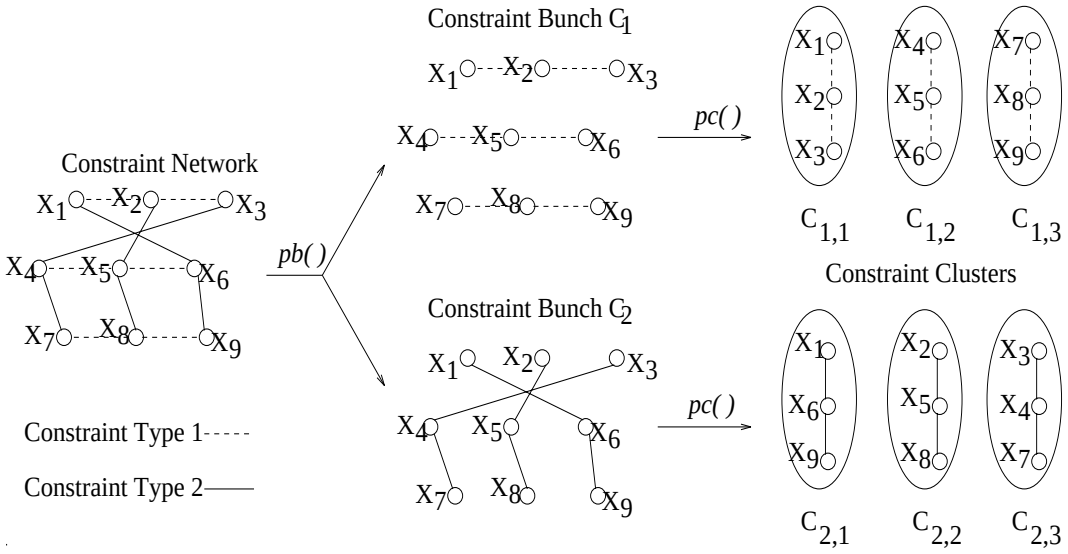


Figure 2: Constraint partition

By constraint partition, an NCSP is decomposed into a set of subproblems, each of which is concerned with the satisfaction of constraints in a constraint cluster, and is assigned to an agent. A solution to a subproblem is an instantiation of the variables in the constraint cluster such that none of the constraints in the subproblem are violated. Each agent has full jurisdiction over variables in the variable set of the assigned constraint cluster. A variable constrained by more than one type of constraint is under the jurisdiction of more than one agent. Agents responsible for the same variable have the same authority on its value, i.e. they can independently change its value. Therefore, a given value of a given variable is part of a solution, if it is *agreed* upon by all its responsible agents in the sense that no agent seeks to further change it. When all subproblems are solved, a solution of the NCSP is found.

2.2 A Society of Reactive Agents

In the framework of CP&CR, problem solving of an NCSP is transformed into collective behaviors of reactive agents. Variables of an NCSP are regarded as objects which constitute agents' environment. An instantiation of the variables characterizes a particular state of the environment. Each agent examines and makes changes to only local environment (variables under its responsibility), and seeks for satisfaction by ensuring that no constraint in its assigned constraint cluster is violated. When an agent detects constraint violations, it reacts by changing the instantiated values of some of the variables under its jurisdiction so that it becomes satisfied.

Agents are equipped with only primitive behavior. When activated, each agent reacts to the current state of the environment by going through an Examine-Resolve-Encode cycle (see Figure 3). It first examines its local view of current environment, 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.

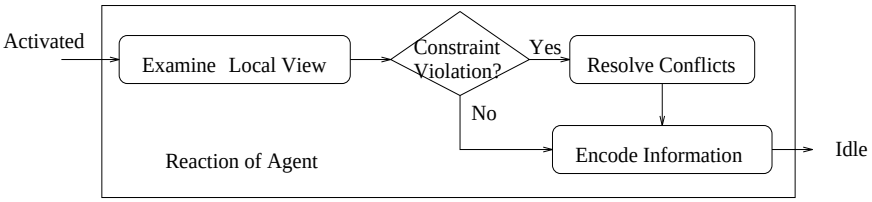


Figure 3: Agent's reactive behavior

Agents coordinate by *passive* communication. They do not communicate with each other directly. Instead, each agent reads and writes coordination information on objects under its jurisdiction. Coordination information on an object represents an agent's partial "view" on the current state of the environment 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.

Agents are divided into subgroups according to perspective (e.g., constraint type). For example, in job shop scheduling problems, one agent subgroup is responsible for resolving capacity constraints, whereas another agent subgroup is responsible for resolving temporal constraints. A variable is under the jurisdiction of agents from different perspectives. Agent subgroups of different perspectives are activated in turn, while agents within a subgroup can be activated simultaneously. An *iteration cycle* is an interval in which all agents are activated once. An initial instantiation of all variables is constructed by a subset of agents. The agents, then, arrive at a solution through collective and successive modifications.

2.3 Coordination Strategy

In a coordinated group of agents, individual behavior is regulated by policies so that the agents' collective actions achieve the common goals. Given the tasks of solving complex, large-scale NCSPs, our coordination mechanisms emphasize convergence efficiency by exploiting characteristics of agent group structure, agent tasks, and communicated information. We have developed coordination strategies that promote rapid convergence by considering the following principles of interaction control.

1. *Least disturbance* - When an agent is resolving conflicts, interactions should be initiated only when necessary and, in such a way as to reduce the chances of causing other concerned agents to subsequently initiate further interaction.

2. *Island of reliability* - Consensus should be reached by a process of evolving coherent group decision-making based on *islands of reliability*, and modifying *islands of reliability* only when group coherence is perceived as infeasible under current assumptions.

3. *Loop prevention* - Looping behaviors, such as oscillatory value changes by a subset of agents, should be prevented.

Least disturbance Least disturbance corresponds to an attempt to minimize ripple effects of agents' actions. To reach group coherence, the number of unsatisfied agents within an operation cycle must be gradually reduced to zero. While an agent always becomes satisfied in an iteration cycle since it instantiates its variables to satisfy only its own constraints, its actions may cause conflicts to instantiations of other agents. Therefore, an agent should resolve conflicts in a way that minimizes the extent of causing disturbances to other agents. Least disturbance is incorporated in agent's heuristics of conflict resolution (see section 3.3). The least disturbance principle is operationalized during conflict resolution in two ways. First, an agent changes the instantiated values of as few variables as possible. Second, for a given selected variable, an agent changes the instantiated value as little as possible.

Island of reliability An *island of reliability* is a subset of variables whose consistent instantiated values are more likely than others to be part of the solution. In particular, islands of reliability should correspond to the most critical constraint clusters, i.e. clusters whose variables have the least flexibility in satisfying their constraints. Islands of reliability provide anchoring for reaching group coherence in terms of propagating more promising partial solutions and are changed less often.[1] For example, in job shop scheduling, a bottleneck resource is an island of reliability. A variable which is a member of an island of reliability is called a *seed variable*. A variable which is not a seed variable is a *regular variable*. Division of seed and regular variables reflects the inherent structure of the problem. The division is static and is complemented by the dynamic interactions among different kinds of agents.

Each agent assumes a role depending on the types of variables it controls. *Dominant agents* are responsible only for seed variables and therefore, are in charge of making decisions within islands of reliability. *Intermediate agents* control variable sets including both seed variables and regular variables. *Submissive agents* are responsible for only regular variables. Intermediate agents interact with submissive agents in a group effort to evolve an instantiation of regular variables compatible with the decisions of dominant agents regarding seed variables. A counter associated with each regular variable records the number of times that a submissive agent has changed the value of the regular variable and, thus, provides an estimate of the search efforts of intermediate and submissive agents to comply with islands of reliability. Intermediate agents monitor the value of the counter associated with the regular variables under their jurisdiction. When the counter of a conflicting regular variable exceeds a threshold, the intermediate agent, instead of changing the conflicting

[1] Blackboard systems (e.g., Hearsay-II speech-understanding system [Erman *et al.*, 1980]) have used the notion of solution *islands* to conduct an incremental and opportunistic problem solving process. Partial solution islands emerge and grow into larger islands, which it is hoped will culminate in a hypothesis spanning the entire solution structure. In CP&CR, islands of reliability refer to partial solutions from some local perspectives and are used as anchors of interaction during the iterative solution repairing process from different local perspectives.

regular variable again, changes the value of its seed variables. In response to value changes in seed variables that result in conflicts, the dominant agent modifies its decisions on islands of reliability. All counters are reset to zero and, therefore, intermediate and submissive agents resume the efforts to evolve a compatible instantiation of regular variables.

Loop prevention Under the principles of least disturbance and islands of reliability, the system exhibits only two types of cyclic behavior. First, a subset of intermediate and submissive agents may be involved in cyclic value changes in order to find a compatible instantiation with dominant agents' decisions. Secondly, a dominant agent may be changing the value of its seed variables in a cyclic way.

The first type of looping behavior is interrupted by intermediate agents when the counter of a conflicting regular variable exceeds a threshold. To prevent the second type of looping behavior, a dominant agent keeps a history of its value changes so that it does not repeat the same configuration of variable instantiations.

2.3.1 Coordinated Group Search

From the point of view of search, the collective problem solving process is a coordinated, localized heuristic search with partially overlapping local search spaces (the values of variables that are the common responsibility of more than one agent). The process starts from an initial instantiation of all variables. The search proceeds as the agents interact with each other while seeking their own goals. Islands of reliability provide the means of anchoring the search, thus providing long term stability of partial solutions. The principle of least disturbance provides short term opportunistic search guidance. The search space is explored based on local feedback. The group of agents essentially performs a search through a series of modifications of islands of reliability. Within each configuration of islands of reliability, intermediate and submissive agents try to evolve a compatible instantiation on regular activities. The search ends when a solution is found or when dominant agents have exhausted all possible instantiation of the seed variables.

CP&CR provides a general framework that is potentially applicable to many NCSPs. We have applied it to solve the Zebra problem (classical test problem for constraint satisfaction algorithms). Experimental results show that CP&CR obtained a favorable performance in terms of the number of variable instantiations required as compared to a number of constraint satisfaction algorithms. In this paper, we focus on the application of CP&CR in job shop scheduling problems.

3 Job Shop Scheduling

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 [Garey and Johnson, 1979], and has been considered as one of the most difficult CSPs. Traditional constraint satisfaction algorithms are shown to be insufficient for this problem [Sadeh, 1991].

3.1 Problem Decomposition and Transformation

Job shop scheduling with non-relaxable time windows is an NCSP, in which each activity is viewed as a quantitative *variable* with a *value* corresponding to the start time of the activity, and all constraints are expressed as numerical relations between variables. CP&CR, by applying the *pb( )* operator, partitions the constraint set into two constraint bunches: a constraint bunch of *exclusion-off* constraints to express temporal precedence constraints on activities within each job[2], and a constraint bunch of *exclusion-around* constraints to express capacity constraints on resources.

By applying the *pc( )* operator, CP&CR further partitions the constraint bunches into a set of constraint clusters corresponding to jobs or resources. Each job is a constraint cluster of exclusion-off constraints and is assigned to a *job agent*. Each job agent is responsible for enforcing temporal precedence constraints within the job. Similarly, each resource is a constraint cluster of exclusion-around constraints and is assigned to a *resource agent*. Each resource agent is responsible for enforcing capacity constraints on the resource. 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. A *bottleneck resource* is the most contended resource among the resources, and corresponds to the most critical constraint cluster. The set of activities contending for the use of a bottleneck resource constitute an island of reliability and, therefore, are seed variables. A bottleneck resource agent assumes the role of a dominant agent, and a regular resource agent is a submissive agent. With the assumption that each job has at least one activity contending for the bottleneck resources, a job agent is an intermediate agent.

3.2 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 Figure 4). It represents the overall temporal flexibility of an activity and is calculated only once during initial activation of job agents.

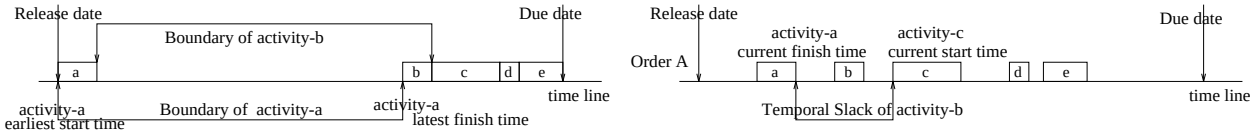
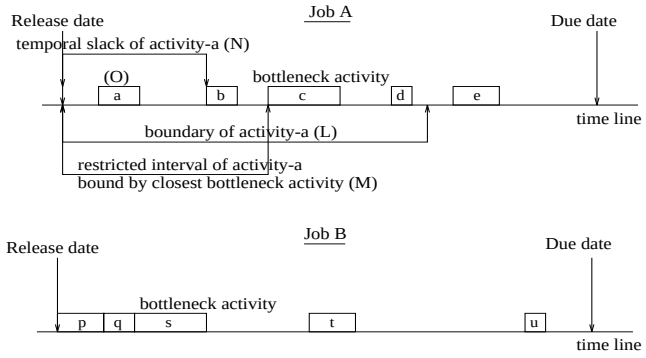


Figure 4: Coordination information: Boundary and Temporal Slack

2. *Temporal Slack* is an interval between the current finish time of the previous activity and current start time of the next activity (see Figure 4). 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.)

[2]Release and due dates constraints are considered as temporal precedence constraints between activities and fixed time points and are included in the exclusion-off constraint bunch.

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 seed activities (see Figure 5). 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 Figure 5, 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.



$$\text{Weight of activity-a} = W_1 \frac{(\text{length of L - length of N - length of O})}{(\text{length of L})} + W_2 \frac{(\text{length of N})}{(\text{length of M})}$$

Figure 5: 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. It indicates the seed variable status of the activity.

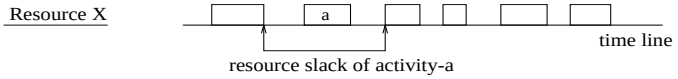


Figure 6: Coordination information: Resource Slack

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 Figure 6). It indicates the range of activity start time 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 between each modification on islands of reliability.

3.3 Reaction Heuristics

Agents' reaction heuristics attempt to minimize the ripple effects of causing conflicts to other agents as a result of fixing the current constraint violations. Conflict minimization is achieved by minimizing the number and extent of activity start time changes. The reaction heuristics utilize perceived coordination information and incorporate coordination strategies of group behaviors.

3.3.1 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 Figure 7). Conflict pairs are resolved one by one. A conflict pair involving a seed 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 seed 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 seed 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.

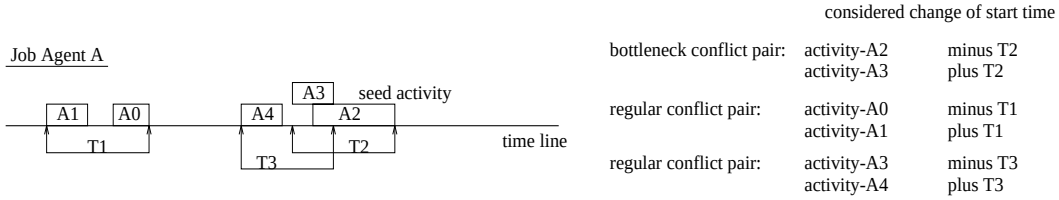


Figure 7: Conflict Resolution of Job Agent

In Figure 7, the conflict pair of activity-A2 and activity-A3 will be resolved first since activity-A2 is a seed variable. 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.

3.3.2 Reaction Heuristics of Regular Resource Agents

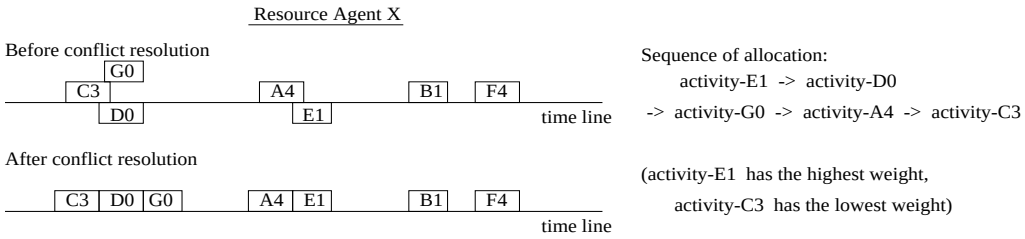


Figure 8: Conflict Resolution of Regular Resource Agent

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 Figure 8, 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.

3.3.3 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 seed 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 (Figure 9 (i)) or re-sequence some activities according to their current start times by swapping the changed activity with an appropriate activity. (Figure 9 (ii)). The intuition behind the heuristics is to keep the changes as minimum as possible.

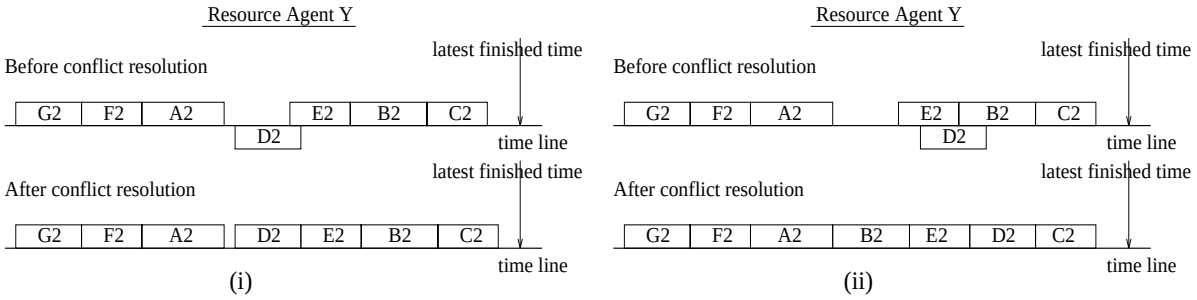


Figure 9: Conflict Resolution of Bottleneck Resource Agent

3.4 System

3.4.1 System Operations

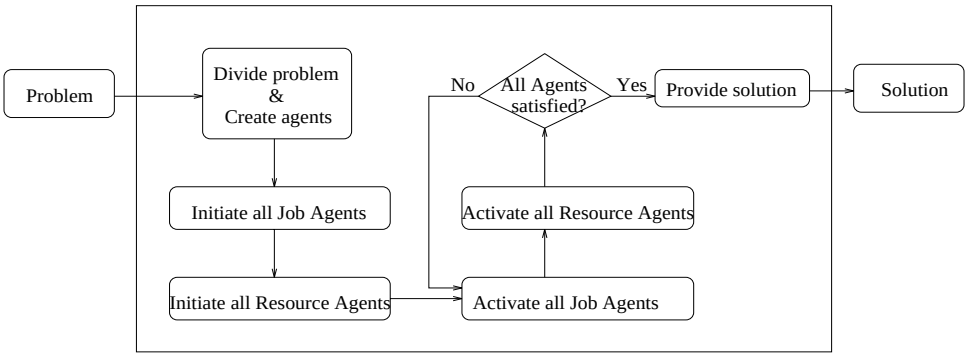


Figure 10: System Control Flow

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 Figure 10). Initially each job agent calculates boundary for each variable under its jurisdiction 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 seed 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 operation 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.

3.4.2 Solution Evolution

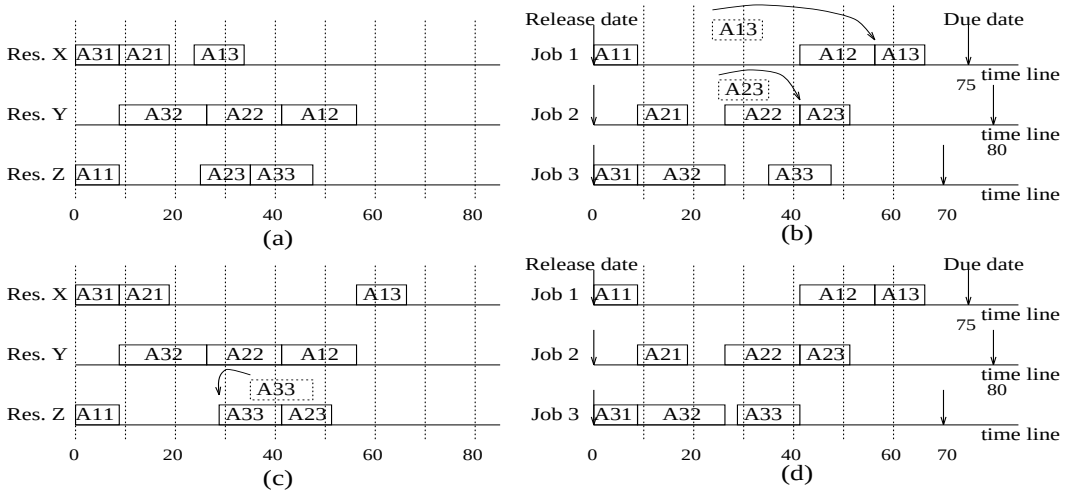


Figure 11: A Simplified Scenario

Figure 11 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), Job1 and Job2 agents are not satisfied with current instantiation and change the start times of A13 and A23, respectively. In (c), Res.Z agent finds a constraint violation and changes the start time of A33. All

[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 the search is barely affected by its starting point, i.e. the search procedure has equal overall performance with heuristic and random initial configurations.

agents are satisfied with the current instantiation of variables in (d) which represents a solution to the problem.

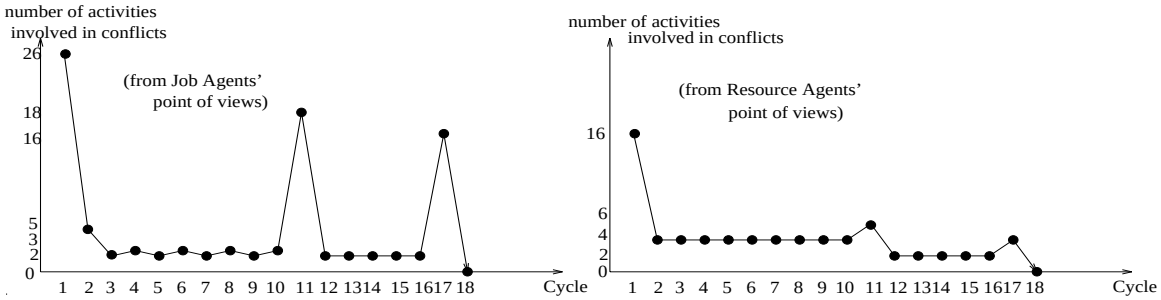


Figure 12: Conflicts Evolution of a more difficult problem

Figure 12 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 islands of reliability set by dominant (bottleneck resource) agents. During cycle 1 to cycle 9, intermediate (job) agents and submissive (regular resource) agents try to evolve a compatible instantiation with islands of reliability, i.e., the instantiation of variables (activities) on the bottleneck resource. In cycle 10, some job agents perceive the effort as having failed and change the values of their seed variables. Bottleneck resource agents respond to constraint violations by modifying instantiation on the islands of reliability. 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 islands of reliability in cycle 16. In cycle 18, the solution is found.

4 Evaluation on Experimental Results

We evaluated the performance of CP&CR on a suite of job shop scheduling CSPs proposed in [Sadeh, 1991]. 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) investigated CORA's scaling up characteristics on problems of larger sizes. The effectiveness of different types of coordination information was reported in [Liu and Sycara, 1993]. We focus on the remaining aspects of evaluation in this paper.

CORA was compared to four other heuristic search scheduling techniques, ORR/FSS, MCIR, CPS, and PCP. ORR/FSS [Sadeh, 1991] 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 [Xiong *et al.*, 1992]. Min-Conflict Iterative Repair (MCIR) [Minton *et al.*, 1992] 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) [Muscettola, 1993] employs a search space analysis methodology based on stochastic simulation which iteratively prunes the search space by posting additional constraints. Precedence Constraint Posting (PCP) [Smith and Cheng, 1993] 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 [Morton and Pentico, 1993], are also included for comparison.

	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

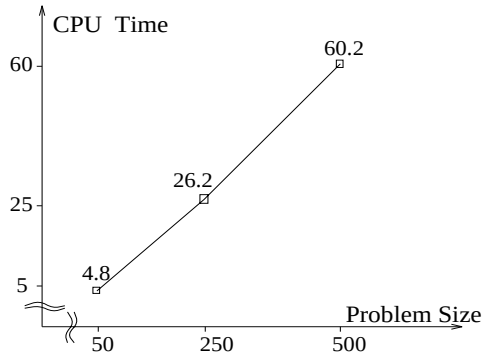


Figure 13: CORA's Scaling Up Property

Table 1 reports the number of problems solved[6] 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 [Muscettola, 1993]. All CPU times were obtained from Allegro Common 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.[7] PCP's CPU times are not listed for comparison because its CPU times in Common Lisp are not available. Its reported CPU times in C are 0.3 second [Smith and Cheng, 1993]. 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. In addition, the same problem generator function producing the benchmark problems was used to produce problem sets of 250 and 500 variables (e.g. 100 factory jobs on 5 machines which is a problem of realistic size). Figure 13 shows CORA's performance on

[6]PCP's performance is sensitive to the parameters that specify search bias [Smith and Cheng, 1993].

[7]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.

these larger sized problems, which exhibits favorable, near-linear scaling-up characteristics.

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

CORA exploits local interactions based on the notion of islands of reliability and has showed to perform quite well on problems with clear resource bottlenecks. For problems with no clear bottlenecks and all resources are loosely utilized (say, below 50 percents of utilization), we expect CORA perform with the same efficiency by selecting the most utilized resource as islands of reliability. However, CORA's current mechanism based on *dominant coordination* may not be sufficient for problems in which all resources are at least moderately utilized (say, above 60 percents of utilization) and there is no outstanding bottleneck. We are interested in developing a more sophisticated mechanism based on *competing coordination* and investigate its utility in various scheduling conditions.

5 Conclusions

In this paper, we have presented a collective problem solving framework, where problem solving is viewed as an emergent functionality from the evolving process of a society of diverse, interacting, well-coordinated reactive agents. We show that large-scaled NCSPs can be decomposed and assigned to different problem solving agents according to disjoint functionality (constraint types) and overlapping responsibility (variable subsets). This decomposition results in utilization of interaction characteristics to achieve problem solving by asynchronous and well coordinated local interactions. Application of the methodology to job shop scheduling with non-relaxable time windows results in very good performance. Our experimental results show that the coordination mechanism (1) incorporates search knowledge and guides the search space exploration by the society of interacting

agents, facilitating rapid convergence to a solution, and (2) is independent of initial configuration. In addition, the search complexity grows only linearly with problem size. We are currently applying the CP&CR methodology to Constraint Optimization Problems (COPs). Preliminary experiments show encouraging results compared to both heuristic search and simulated-annealing-based techniques. We are also investigating the utility of CP&CR in other domains with different problem structures.

References

[Brooks, 1991] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the IJCAI-91*, pages 569–595, 1991.

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[Dechter, 1988] Rina Dechter. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

[Decker, 1987] Keith S. Decker. Distributed problem-solving techniques: A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(5):729–739, 1987.

[Durfee, 1988] Edmund H. Durfee. *Coordination of Distributed Problem Solvers*. Kluwer Academic Publishers, 1988.

[Erman *et al.*, 1980] L. D. Erman, F. A. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The hearsay-II speech-understanding system: integrating knowledge to resolve uncertainty. *Computer Survey*, 12(2):213–253, 1980.

[Ferber and Jacopin, 1991] J. Ferber and E. Jacopin. The framework of Eco Problem Solving. In Demazeau and Muller, editors, *Decentralized AI 2*. Elsevier, North-Holland, 1991.

[Freuder and Hubbe, 1993] Eugene C. Freuder and Paul D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of the IJCAI-93*, pages 254–260, 1993.

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

[Gasser and Hill, Jr., 1990] Les Gasser and Randall W. Hill, Jr. Engineering coordinated problem solvers. *Annual Review of Computer Science*, 4:203–253, 1990.

[Huhns and Bridgeland, 1991] M. Huhns and D. Bridgeland. Multiagent truth maintenance. *IEEE Transactions on System, Man, and Cybernetics*, 21(6):1437–1445, 1991.

[Langton *et al.*, 1991] C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors. *Artificial Life II*. Addison-Wesley, 1991.

[Langton, 1989] Christopher G. Langton, editor. *Artificial Life*. Addison-Wesley, 1989.

[Lhomme, 1993] Olivier Lhomme. Consistency techniques for numerical CSPs. In *Proceedings of IJCAI-93*, pages 232–238, 1993.

[Liu and Sycara, 1993] JyiShane Liu and Katia P. Sycara. Distributed constraint satisfaction through constraint partition and coordinated reaction. In *Proceedings of the 12th International Workshop on Distributed AI*, 1993.

[Mackworth, 1987] Alan K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia in Artificial Intelligence*, pages 205–211. Wiley, New York, 1987.

[Meyer and Wilson, 1991] Jean-Arcady Meyer and Stewart W. Wilson, editors. *Proceedings of the First International Conference on Simulation of Adaptive Behavior - From Animals To Animats.* MIT Press, 1991.

[Minton *et al.*, 1992] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Morton and Pentico, 1993] Thomas E. Morton and David W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.* John Wiley & Sons, New York, 1993.

[Muscettola, 1993] Nicola Muscettola. HSTS: Integrated planning and scheduling. In Mark Fox and Monte Zweben, editors, *Knowledge-Based Scheduling.* Morgan Kaufmann, 1993.

[Nadel, 1989] Bernard A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[Sadeh, 1991] 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.

[Shoham and Tennenholtz, 1992] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of AAAI-92*, pages 276–281, 1992.

[Smith and Cheng, 1993] Stephen F. Smith and Cheng-Chung Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of AAAI-93*, pages 139–144, 1993.

[Sycara *et al.*, 1991] Katia Sycara, Steve Roth, Norman Sadeh, and Mark Fox. Distributed constraint heuristic search. *IEEE Transactions on System, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[Xiong *et al.*, 1992] 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.

[Yokoo *et al.*, 1992] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems*, pages 614–621, 1992.