Collective Problem Solving through Coordination in a Society of Reactive Agents

JyiShane Liu

CMU-RI-TR-94-23

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

June 1994

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 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

Research in Distributed Artificial Intelligence has been mostly oriented towards multi-agent systems composed of sophisticated individuals [5]. These agents are categorized as cognitive agents, which are given the capabilities to reason about their environment, to predict future events and choose between possible actions, and exhibit goal-driven behavior [13]. In the context of problem solving, this line of research is called *Cooperative Distributed Problem Solving* (CDPS) [11] or *Coordinated Problem Solving* [21], and has been defined by Lesser and Corkill as a problem solving process which involves a "loosely coupled distributed network of semiautonomous problem-solving nodes that are capable of sophisticated problem solving and cooperatively interact with other nodes to solve a single problem" [29]. Decker also refers to CDPS as "coarse-grained, task-level problem decompositions"[12].

Early work in CDPS considers problems, such as distributed interpretation [8, 28], and air traffic control [32, 4], that are geographically dispersed and have communication bandwidth. As pointed out by Durfee *et al.*, typical CDPS applications involve problems that consist of "a set of interdependent subproblems that arise because of spatial, temporal, and functional distribution of data, knowledge, and processing capabilities" [15]. In these applications, a central problem solver is considered as inappropriate because of issues such as limited computation, limited communication, and reliability. CDPS provides techniques to achieve a balance between problem solving and coordination that leads to acceptable overall system performance. CDPS research has been extended to other areas, such as distributed planning and control [40], cooperating expert systems [42], and cognitive models of cooperation [45, 7].

The fundamental strategy behind CDPS is divide-and-conquer. Complex problems are divided into smaller parts that can be reasonably solved and compatible subproblem solutions are then integrated into an overall solution. However, intricate interactions among the set of subproblem solvers can invalidate the advantage of reduced task complexity. Coordination is a complex and little understood phenomenon, and has been regarded as a fundamental component of intelligence. CDPS has primarily been developed for problem applications where centralization is not a viable option and has focused on coordination involving a set of sophisticated cognitive agents.

Recently, there has been growing interest in Artificial Life (ALife), studying computational models of agent societies composed of simple agents that interact asynchronously [27]. Researchers in this area attempt to synthesize and simulate behaviors characteristic of biological systems, such as self-organization, learning, adaptation, and evolution. In these models global behaviors emerge out of the organized, local interactions among individual simple agents [26, 33, 18]. The field of ALife aims to an understanding of spontaneous organization and adaptation and, by eventually building up to, an understanding of mental processes [1]. Enthusiasm has been growing to viewing ALife as *life-as-it-could-be* [27] and as an implication to *bottom-up* intelligence [3].

Apart from pure theoretical research, application oriented researches using ALife strategies have also been successfully demonstrated. Brooks proposed a subsumption architecture for insect-like robots in which coherent behaviors emerge from subcomponents interacting in the world [2]. Steels discussed emergent functionality in which a function is achieved indirectly by the interaction of more primitive components among themselves and with the world [44]. Kitano developed a massively parallel memory-based approach to natural language processing [24]. Despite the growing interest in ALife, the utility of these models in problem solving has not been fully explored.

ALife provides a new perspective for developing effective problem-solving techniques. Instead of using sophisticated agents to perform top-down synthesis, problem-solving can be achieved by the bottom-up interaction of a society of simple agents. In these models, problems are represented by the configuration of a society of reactive agents and their environment. Agents are characterized by their *situation-action* rules. The problem solution emerges as a result of the asynchronous, local interactions among the society of reactive agents. Because of the computational economy of reactive agents, this approach indicates good potential in problem-solving efficiency and scalability. However, to achieve efficient problem solving, agents' actions must be coordinated to promote rapid convergence. These agents perform locally contained actions, while their behaviors are influenced by passive communication from other agents.

We have developed a computational framework for *collective problem solving* by a society of reactive agents. Problem solving is viewed as an emergent functionality from the evolving process of the society of diverse, interacting, and well-coordinated reactive agents. Agents are situated in their environment and act by stimulus and response. Coordinated interactions are based on simple flows of information. The collective actions of the reactive agents potentially provide an effective tool for complex problem solving. Specifically, the development of the collective problem solving framework involves the following subjects:

- *Problem decomposition*: The transformation from a problem to a society of simple agents is defined by a decomposition scheme. Each agent is assigned to a task corresponding to a small part of the problem. Situation-action rules specify how agents would act to achieve their tasks. The problem is solved when all agents achieve their tasks simultaneously.

- *Interaction analysis*: When a problem is mapped into a society of agents, intense interactions among agents ensue. In order for the society to move toward coherence, influences of agents' actions on each other need to be identified. These interactions are viewed as rich information sources that can be exploited to guide agents' behaviors toward group coherence.

- *Coordination mechanism*: Group behavior of agents is characterized by the coordination mechanism in the society. For our problem-solving purpose, we require the group of agents to reach coherence in order to provide a solution. In addition, we seek for rapid convergence to improve problem-solving efficiency. The design of a coordination mechanism includes regulation policies and communication among agents. Effective coordination strategies and useful coordination information are the focus of study.

- *Behavior design*: An agent's behavior corresponds to various actions it performs to achieve its goal. The collective behavior of agents represents problem-solving activities that the group performs. In this framework, it is critical to analyze agent interactions, investigate useful information exchange between agents, and coordinate the highly distributed activities. All of these lead to designing agents' behaviors such that (1) they avoid harmful interactions with other agents, (2) they react appropriately towards rapid group convergence.

- *System*: Investigations in the above subjects follow the construction of a collective problem solving system. A society of agents are created according to specification of the problem. Search is performed in the form of agents' collective actions. The system is expected to perform efficient transformation from initial disorder to group coherence. The problem is solved by the collective interactions of a society of simple, reactive agents.

The problem domains of the collective problem solving framework are *Constraint Satisfaction Problems (CSPs)*. 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* $C = \{c_1, c_2, \cdots, c_n\}$ [31]. 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) [30] 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 space.

In general, CSPs are solved by two complementary approaches, backtracking and network consistency algorithms [31][9][39]. Recently, heuristic revision [34] and decomposition [10][19] techniques for CSPs have been proposed. On the other hand, recent work in DAI has considered the distributed CSPs [23] [46] [49] 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 illustrate other work that are similar to our approach. In Section 3, we define the CP&CR model, in which problem decomposition, coordination mechanisms, and asynchronous search procedure are specified. In Sections 4 and 5, 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 6, we conclude the paper and outline our current work on CP&CR.

2 Related Work

There are a lot of work in different areas, such as distributed problem solving, cooperative problem solving, constraint satisfaction, and job shop scheduling, that can be related to CP&CR. We limit the scope of discussion to only the work that utilize the most similar approaches. In particular, we focus on approaches that perform collective problem solving with the following characteristics: (1) no agent has a global view, (2) no agent is capable of solving problem alone, (3) each agent is simple and reactive, (4) each agent is responsible for only a small part of the problem, (5) agents directly/indirectly influence others' actions.

Research utilizing social insects metaphor in collective problem solving is still at its early stage and has been pursued mostly by European researchers. Research efforts are directed toward constructing systems composed of reactive agents that are situated in their environment and act by stimulus and response. Various approaches have been proposed for problems ranging from planning to optimization. The existing reactive multi-agent systems for problem solving that are representative and most related to our research (not necessary with the same motivation and purposes) are briefly described below.

Colorni *et al.* presented a distributed optimization scheme for the Travelling Salesman Problem (TSP) based on the analogue of ant colonies [6]. The mechanism that enables blind ants to establish shortest route paths between their colony and feeding sources were employed. A moving ant lays some substance in varying quantity on the ground. An isolated ant moves essentially at random but may follow a trail when it detects the substance. The probability that an ant moves along a trail increases with the amount of substance on the trail. The scheme required a considerable number of ants and a large number of trials to find the best solution. They obtained respectable results on mid-sized TSPs (up to 75 cities), but with long computational time.

Ghedira constructed a model of simple reactive agents for resource allocation problems without precedence constraints on tasks [22]. The model involves Task and Resource agents in interactions, each of them seeking its maximal satisfaction in terms of getting resource allocation and maximizing resource utilization, respectively. Each unsatisfied Task agent chooses one of the possible Resource agents and required to be allocated. Behavior of a Resource agent is based on a simulated annealing algorithm, where tolerance to local deterioration of satisfaction progressively decreases. Experiments were conducted on randomly generated problems. No comparison to other established approaches was offered.

Ferber developed the *Eco-Problem-Solving* (EPS) model [17], in which problems are decomposed and solved by interactions of simple behavior based *eco-agents*. Eco-agents behave according to their specific programs of simple "*tropism*", i.e. a behavior made of reactions to the environment, and are continuously questing for a satisfaction state. Interactions between agents are characterized by aggression and concession. Problem solving is seen as the production of stable states in a dynamic system, where evolution is due to the behavior of simple agents. Application of EPS to classical planning problems in the blocks world was demonstrated.

Drogoul applied EPS model to the N-puzzle problem [14], which involves rearranging N square tiles from some random initial configuration into a particular goal configuration on a square board containing the tiles and a blank position. Each tile is modelled as an eco-agent and has a goal to reach its destined position. Behaviors of eco-agents are characterized by "the will to be satisfied" and "the obligation to flee". Satisfaction of tiles is prioritized by heuristics (first solving the row or column furthest to the final position of the blank). Additional heuristics (Manhattan distances) are also used in eco-agent's decision-making in direction of movement. Their experimentation showed that the system can solve very large puzzles (up to 899-puzzle) in reasonable time, while the best record of RTA* and LRTA* [25] is 24-puzzle. However, the system did not obtain optimal solutions (the solution lengths of small puzzles are approximately twice the optimal solution lengths computed by A*).

EPS model and CP&CR share the approach of solving complex problems by interactions of simple reactive agents. However, our collective problem solving framework intends to advance the effectiveness of the approach by further exploiting local interactions. First, instead of relying on goal prioritization which entails serial operations, we have developed more effective coordination mechanism which allows asynchronous agent activities. Second, in addition to agents' partial information on their environment, useful coordination information between agents has been investigated and exploited to facilitate rapid group coherence.

Apart from these systems of reactive agents, there has been an organizational model of Asynchronous Team (A-Team) proposed in [47]. A-Team consists of autonomous agents each of which can choose what to do and when to communicate with its team mates, if ever. Agents in A-Team are divided into four categories: *construction* agents, *modification* agents, *deconstruction* agents, and *destruction* agents. These agents are oriented towards solution variation and work on populations of solutions. The populations of data-objects produced by the agents are controlled by "herding" or "consensus-seeking" strategies where the population are kept from explosion by the destruction agents or by mode changes of most agents. A-Team have shown successful applications on solving large travelling salesman problems and configuring task-specific robots.

Unlike A-Team in which agents do not communicate frequently, if ever, CP&CR explicitly utilizes local interactions of reactive agents to guide the search. Simple coordination information is constantly exchanged between agents and directly influences agents' actions. One single solution candidate is iteratively modified by agents. A coordination mechanism regulates agents' actions towards solution consensus. In addition, CP&CR includes a decomposition scheme that defines the organization of agents according to problem specifications.

3 Constraint Partition and Coordinated Reaction

We have developed a collective problem-solving framework, called Constraint Partition and Coordinated Reaction (CP&CR), for a subset of NCSPs. The framework draws on characteristics of a social insects colony, such as functional decomposition, task division, and effective coordination based on simple flows of information. In CP&CR, a society of specialized and well-coordinated reactive agents collectively solve an NCSP. Agents are situated in their environment, react to others' actions, and communicate with others by leaving and perceiving particular messages on the objects they act 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.

3.1 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.

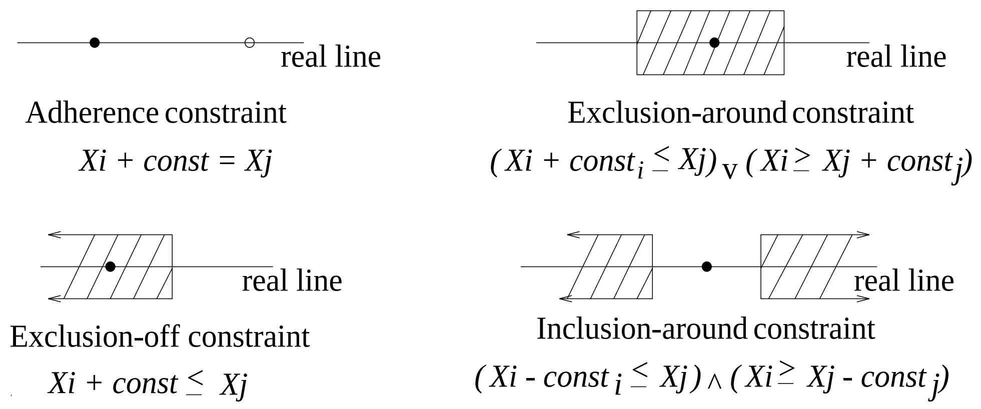

real line
Adherence constraint
$Xi + const = Xj$
real line
Exclusion-around constraint
$(Xi + const_i \leq Xj) \vee (Xi \geq Xj + const_j)$
real line
Exclusion-off constraint
$Xi + const \leq Xj$
real line
Inclusion-around constraint
$(Xi - const_i \leq Xj) \wedge (Xi \geq Xj - const_j)$

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.

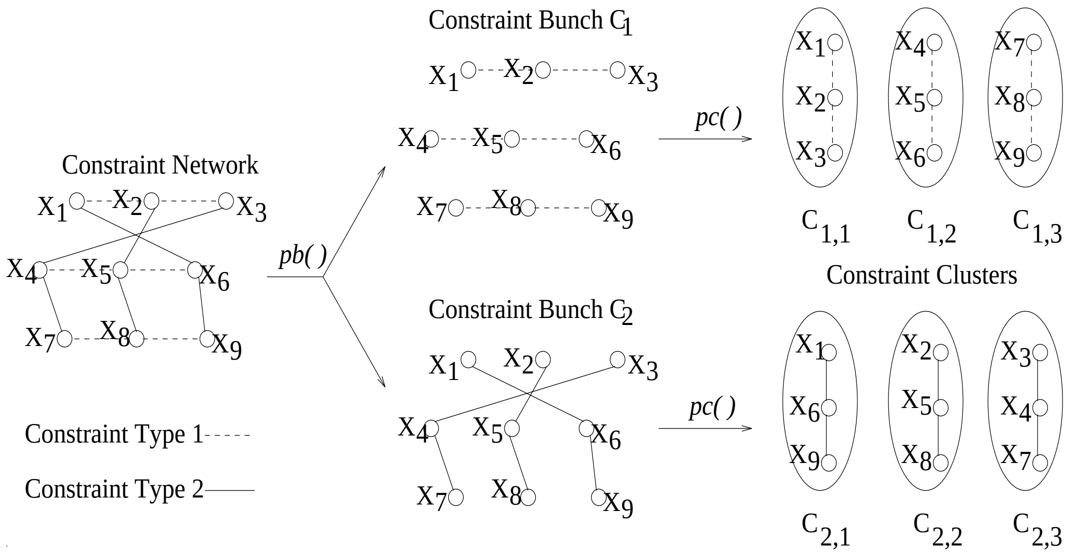


Figure 2: Constraint partition

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*

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. Agent type corresponds to constraint type. Constraint enforcement is *decoupled* within the same type of agents. 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.

3.2 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.

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.

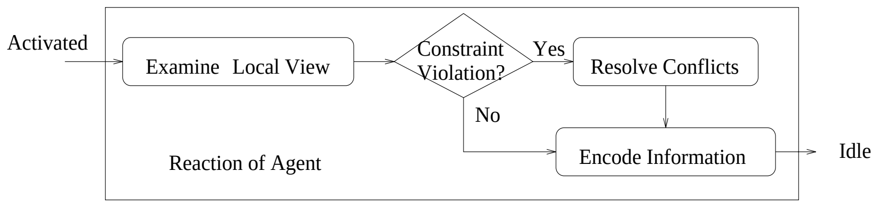


Figure 3: Agent's reactive behavior

Agents are classified according to perspective (e.g., constraint type). For example, in job shop scheduling problems, one agent type is responsible for resolving capacity constraints, whereas another agent type is responsible for resolving temporal precedence constraints. A variable is under the jurisdiction of agents from different perspectives. Agents of different types are activated in turn, while agents of the same type 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.

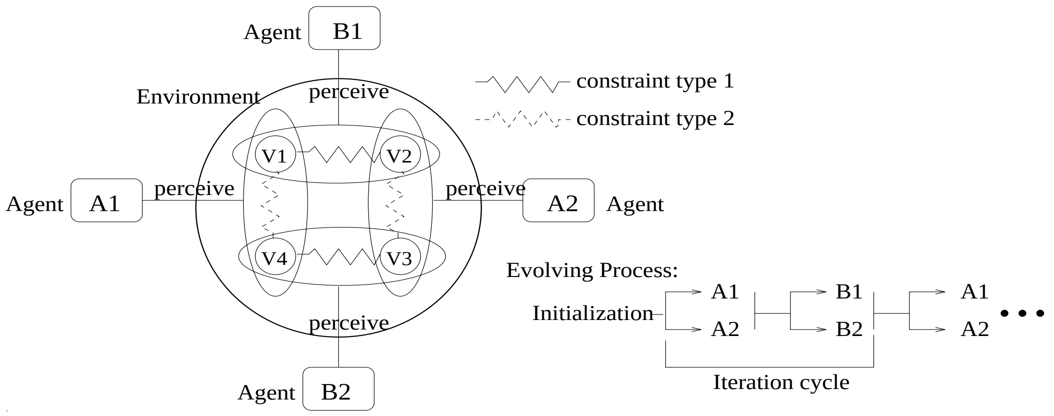


Figure 4: Society of Reactive Agents

Figure 4 shows a society of reactive agents for a simple problem that involves only four variables, v1, v2, v3, v4, whose instantiations are restricted by four constraints of two different types. Variables are assigned to two subgroups of agents, in which subgroup A is responsible for constraint type 2, and subgroup B is responsible for constraint type 1. Within a subgroup, each agent is in charge of a subset of variables that are not connected to the other variables by the same type of constraints. For example, agent A1 is assigned to variables v1 and v4; agent A2 is assigned to variable v2 and v3. Similarly, agent B1 is responsible for variable v1 and v2; agent B2 is responsible for variable v3 and v4. Variables constitute agents' environment. Agents "perceive" only local environment in the sense that they examine and make changes to only the variables under their jurisdiction.

3.3 Interaction Analysis

As the group of agents are searching for an instantiation of variables that is accepted by all, instances of interaction set off one another. In order to facilitate group convergence, sources and results of interaction instances must be analyzed and controlled. For example, Figure 5 shows a particular state of the environment representing a simple job shop scheduling problem which includes only three jobs on three resources.

Job i is a job agent and Res j is a resource agent. Each box is an activity with the first number representing the number of the job and the second number representing its sequence within the job, i.e., A23 is the third activity within job2. A single activity, e.g., A23, is under the jurisdiction of a job agent Job2 and a resource agent Res.Z. In the particular state, Job1 is the only agent who is not satisfied because the precedence constraint between A12 and A13 is violated. In order to complete its task, Job1 agent would need to change the start time of either A12 or A13. Similarly, suppose A23 now starts at time 50, Res.Z agent would need to either change the start time of A33 or A23.

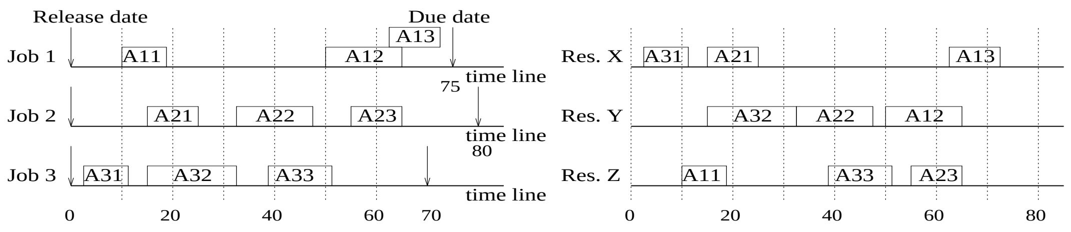


Figure 5: A State of the Environment: An Instantiation of Variables

An analysis on the interactions leads to the following intuitions. First, each agent's action of changing the start time of an activity potentially disturbs others' satisfaction status. To minimize reciprocal harmful interactions, agents' actions should take into account the preference of others regarding variable instantiation. Second, a solution to the most constrained subproblem is more likely to be part of a global solution than that of less constrained subproblems[1] and, therefore, can serve as an anchor of interactions. Third, cyclic interactions should be prevented.

[1]In traditional constraint satisfaction literatures[39], there was a notion of most constrained variable which has the least possible values. The intuition was instantiating the most constrained variable seemed to be more critical than instantiating other less constrained variables and the instantiation would be more likely to be correct than the instantiation from a large domain of possible values. We extend this intuition to CP&CR in which a subset of variables instantiated from a local perspective that are most constrained seems to be more reliable.

3.4 Coordination Mechanism

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.

3.4.1 Coordination Strategy

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 4.3). 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 such that it deviates from the previous value as less 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.[2] For example, in job shop scheduling, a bottleneck

[2]Blackboard systems (e.g., Hearsay-II speech-understanding system [16]) have used the notion of solution *islands* to conduct an incremental and opportunistic problem solving process. Partial solution islands emerge

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 as described below.

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 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.

3.4.2 Coordination Information

Coordination information among agents is associated with each variable by its responsible agents. When an agent is resolving constraint violations on a variable under its responsibility, the coordination information provided by the other agents that govern the same variable is used in decision-making. Two types of information, *disturbance information* and *dominance information*, are exchanged between agents.

Disturbance information reveals how the variable is constrained by an agent regarding

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.

potential value changes in two forms. One form of disturbance information is an interval of values in which the variable may be instantiated free of conflicts. The other form of disturbance information is a value that measures the likelihood of the variable involving in conflicts with another variable in the same cluster, if its instantiated value is changed. Different disturbance information is used by intermediate agents and submissive agents in their innate heuristics of least disturbance. Dominance information indicates special status of seed variables and measures the search effort of intermediate and submissive agents between each modification on islands of reliability. Search efforts are estimated in the form of the number of times the value of each regular variable is changed by a submissive agent. This type of information is used by intermediate agents to recognize seed variables and decide when to contest seed variables. The following section describe how each agent behaves under the influence of coordination information.

3.5 Agents' Behavior

The design of agents' behavior incorporates the coordination strategies and the use of coordination information. For each agent's role, we describe an abstract algorithmic process of how it behaves.

Dominant Agents

1. Examine the instantiations of variables under its jurisdiction. If all constraints are satisfied, stop.
2. Resolve conflicts by changing the instantiated values of some seed variables in a way that the new value of a seed variable being changed by an intermediate agent is in the same direction of modification on the real line by the intermediate agent[3] and that the new configuration of variable instantiations does not repeat an old one in the record.
3. Record the new configuration of variable instantiations.

Intermediate Agents

1. Examine the instantiations of variables under its jurisdiction. If all constraints are satisfied, go to step 3.
2. Resolve conflicts. Use the dominance information on the variables marked by dominant agents to recognize seed variables.
   (a) For conflicts involving seed variables,

[3]For example, in job shop scheduling, when a bottleneck resource agent finds a capacity constraint violation due to a seed activity being changed to a later start time by a job agent, the bottleneck resource agent would re-arrange its resource interval allocation such that the seed activity would not start earlier than the start time set by the job agent.

- If the conflicting regular variable can not be changed to a value within its domain that satisfies its constraint with current instantiation of the seed variable, change the instantiated value of the seed variable.
- Otherwise, consult the dominance information (change counter) on the conflicting regular variable written by a submissive agent. If the counter has exceeded a threshold, change the instantiated value of the seed variable.
- Otherwise, change the instantiated value of the regular variable.

(b) For conflicts involving only regular variables,

- Consult disturbance information (both value interval and change counter) on both regular variables written by submissive agents. Change the instantiated value of the regular variable that has less magnitude of disturbance. If one of the conflicting variables can be changed within its value interval to resolve the conflict, then the agent changes the value of that variable. Otherwise, the agent chooses to change the variable that is changed less frequently (with a change counter of lesser value).

3. Encode disturbance information (likelihood measure) on each variable under its jurisdiction based on their current instantiated values.

Submissive Agents

1. Examine the instantiations of variables under its jurisdiction. If all constraints are satisfied, go to step 4.

2. Resolve conflicts. Consult the disturbance information (likelihood measure) on each conflicting regular variables written by intermediate agents. Change the instantiated values of regular variables that have less magnitude of disturbance.

3. Encode dominance information (change counter) on changed variables. For each changed variable, increase the associated counter by a number that represents the agent's certainty on the new value of the variable. Usually, the counter is increased by one so that it records the number of times the agent has changed the value of the variable, and provides an estimate of the search efforts of intermediate and submissive agents. However, if the agent encounters an intense conflict (a number of variables involved in conflicts within a short interval of values), it can resolve the conflict and set the associated counters of the involved variables to a number larger than the threshold. Therefore, decisions on the dynamically arised critical areas can be propagated.

4. Encode disturbance information (value interval) on each variable under its jurisdiction based on their current instantiated values.

Figure 6 shows the interaction pattern between agents of different roles. Dominant agents are responsible for a consistent instantiation on the islands of reliability. A solid loop represents frequent interactions between intermediate and submissive agents as they modify instantiation on regular variables to satisfy their own constraints and try to evolve a compatible instantiation with islands of reliability. A dotted loop represents less frequent interactions between intermediate and dominant agents that occur when intermediate agents change the instantiation on the islands of reliability and dominant agents respond by making further changes (if necessary) to the instantiation on the islands of reliability to make sure they are consistent. The action flow is switched from solid loop to dotted loop when (1) search efforts on compatible instantiation on regular variables have reached a threshold, (2) some submissive agent identifies dynamic critical areas. Both conditions are perceived by intermediate agents as some change frequency counter exceeds the threshold. Typically, the action flow remains within several solid loops until it is switched by intermediate agents to one dotted loop, and immediately returns to solid loops again.

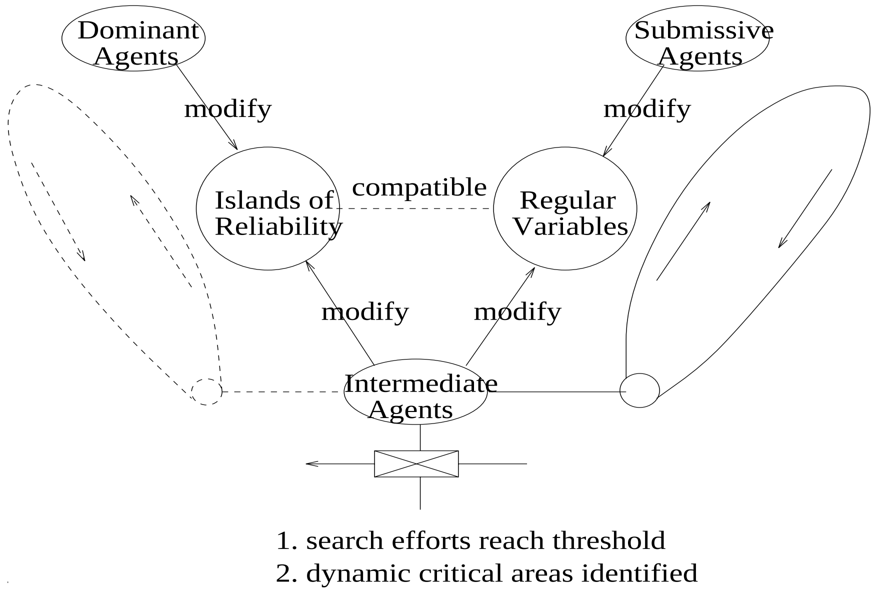


Figure 6: Interaction Pattern

3.6 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[4] (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 the proposed research, we focus on the application of CP&CR in job shop scheduling problems.

4 CP&CR in 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 [20], and has been considered as one of the most difficult CSPs. Traditional constraint satisfaction algorithms are shown to be insufficient for this problem [41].

4.1 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[5], 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

[4]Zebra problem can be considered as an NCSP because (1) all variables (attributes) have domains of numerical values (house numbers), (2) all constraints can be represented by numerical relations between two variables, such as "equal", "not equal", etc.

[5]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.

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.

4.2 Coordination Information

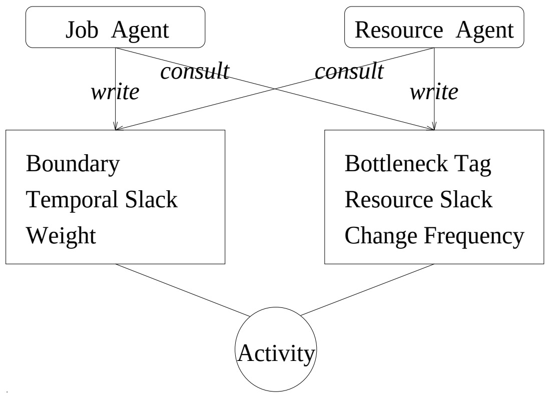


Figure 7: Coordination Information

Coordination information *written* by a job agent on an activity is referenced by a resource agent, and vice-versa, as shown in Figure 7.

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 8). It represents the overall temporal flexibility of an activity and is calculated only once during initial activation of job agents. Boundary is a disturbance information in the form of a value interval.

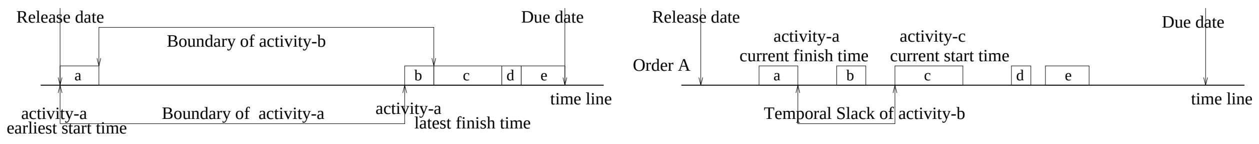
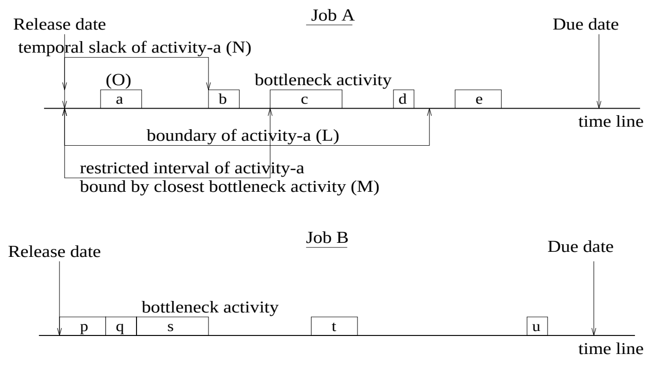


Figure 8: Coordination information: Boundary and Temporal Slack

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 8). 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.) Temporal slack is a disturbance information in the form of a value interval.

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 9). 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. Weight is a disturbance information in the form of a likelihood value. In Figure 9, 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.


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)
Job B
Release date
Due date
bottleneck activity
p
q
s
t
u
time line

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

Figure 9: 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. Bottleneck tag is a dominance information.

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 10). 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.) Resource slack is a disturbance information in the form of a value interval.

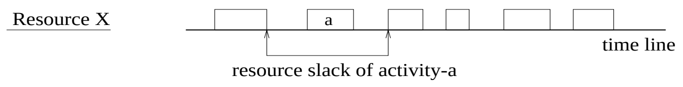


Figure 10: Coordination information: Resource Slack

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

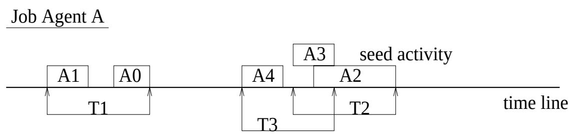
effort of job and regular resource agents between each modification on islands of reliability. In addition, it can be used by submissive resource agents to propagate their decisions on the dynamically arised bottleneck resource intervals by setting the counter to a number larger than the threshold. Change frequency is a dominance information.

4.3 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.

4.3.1 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 11). 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. If one of the two activities can be changed within its boundary and resource slack, job agents will change that activity. Otherwise, job agents change the activity with less change frequency. In any conditions, depending on the precedent relation between the two activities, the start time of the selected activity is changed to a value that is either the end time of the other activity or the start time of the other activity minus the duration of the selected activity.



	considered change of start time	
bottleneck conflict pair:	activity-A2	minus T2
	activity-A3	plus T2
regular conflict pair:	activity-A0	minus T1
	activity-A1	plus T1
regular conflict pair:	activity-A3	minus T3
	activity-A4	plus T3

Figure 11: Conflict Resolution of Job Agent

In Figure 11, 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. The decision is based on the boundary, resource slack, and change frequency of both activities.

4.3.2 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. Conflicted activities are allocated in a sequence based on their weights. If their original resource intervals have been preempted by other activities, a most adjacent resource interval within their boundaries are allocated considering the preference of staying within their 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.

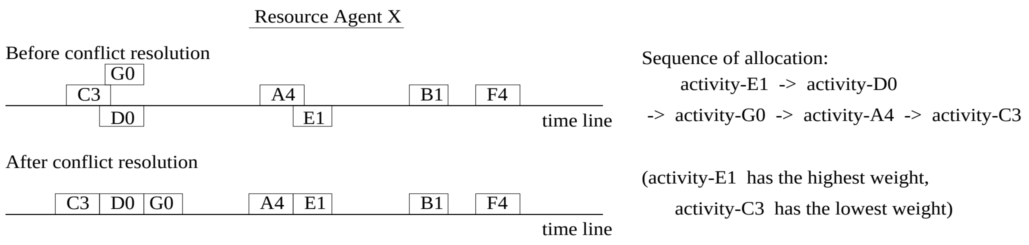

Resource Agent X
Before conflict resolution
G0
C3
A4
B1
F4
D0
E1
time line
After conflict resolution
C3
D0
G0
A4
E1
B1
F4
time line
Sequence of allocation:
activity-E1 -> activity-D0
-> activity-G0 -> activity-A4 -> activity-C3
(activity-E1 has the highest weight,
activity-C3 has the lowest weight)

Figure 12: Conflict Resolution of Regular Resource Agent

For example, in Figure 12, activity-A4 was preempted by activity-E1 which has higher weight. A most adjacent resource interval is allocated to activity-A4. 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.

4.3.3 Reaction Heuristics of Bottleneck Resource Agents

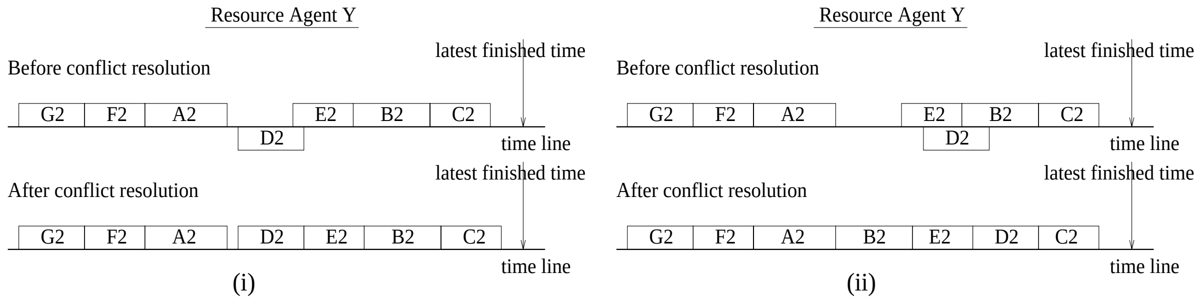


Figure 13: Conflict Resolution of Bottleneck Resource Agent

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 13 (i)) or re-sequence some activities according to their current start times by swapping the changed activity with an appropriate activity. In Figure 13 (ii), the changed activity-D2 is swapped with activity-B2 since the changed start time of activity-D2 is later than the start time of activity-E2. Resource intervals are then allocated to the new sequence - activity-B2, activity-E2, activity-D2. The intuition behind the heuristics is to keep the changes as minimum as possible. Note that the new start time of activity-D2 is in the same direction (on the real line) of modification made by a job agent on activity-D2.

4.4 System

4.4.1 System Operations

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 14). 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.[6]

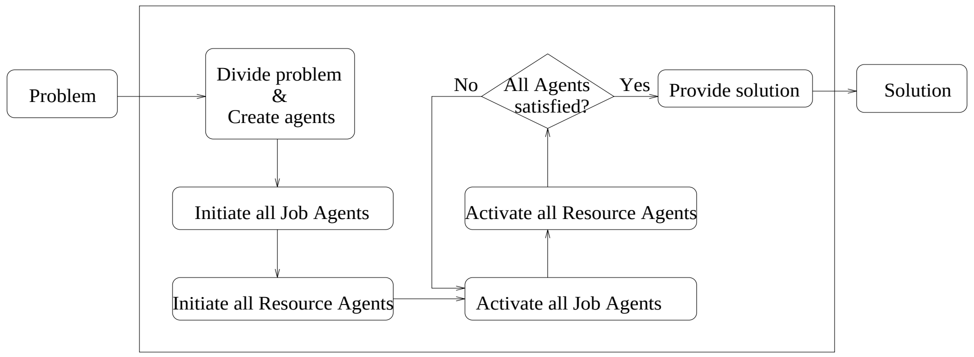


Figure 14: System Control Flow

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* (i.e., fixed resource contention ratio) and corresponds to a resource within the entire interval specified

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

by the problem. We exploit the *dynamic local interactions* of agents that utilize both notions of static and dynamic bottlenecks. For example, static bottleneck is regarded as islands of reliability. Regular resource agents are able to identify high resource contention during particular time intervals (dynamic bottlenecks) and assign high change frequency to the involved activities in order to propagate their decisions.

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.[7]

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.

4.4.2 Solution Evolution

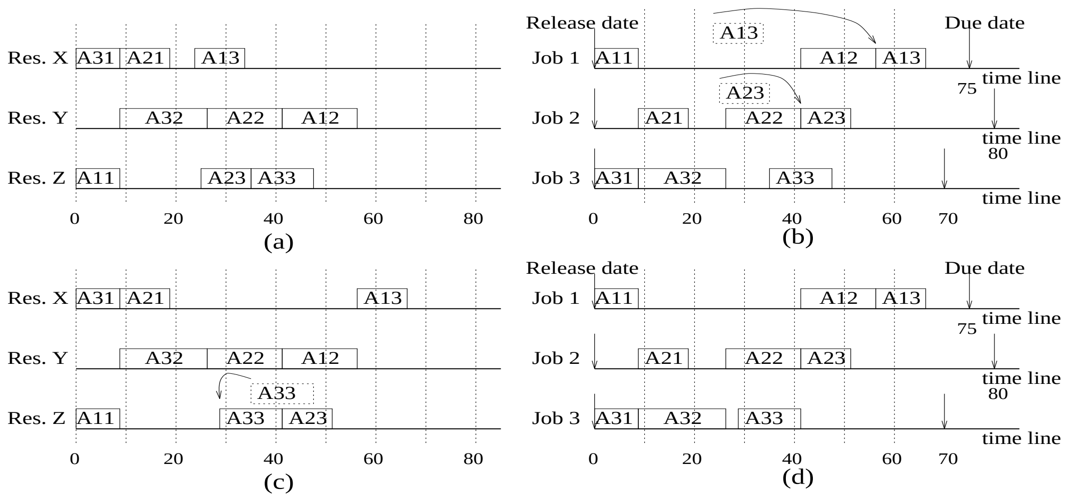


Figure 15: A Simplified Scenario

Figure 15 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 which is a conflict-free schedule from the point of view of resource agents. In (b), A13, A23 within dotted rectangular boxes represent the start times assigned by resource

[7]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 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.

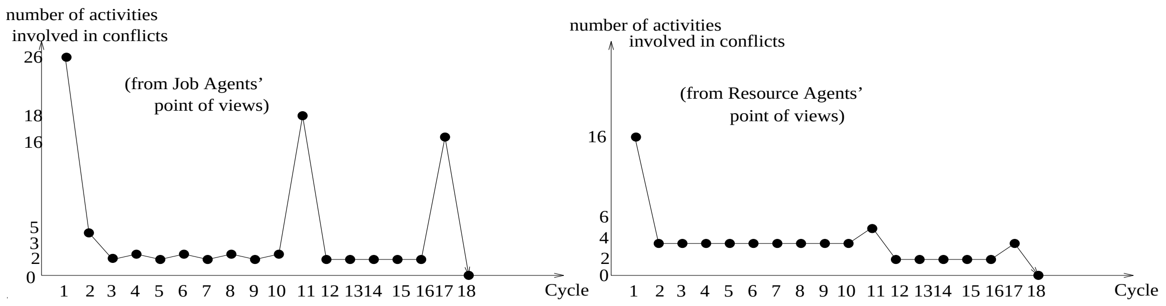


Figure 16: Conflicts Evolution of a more difficult problem

Figure 16 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.

5 Evaluation on Experimental Results

We evaluated the performance of CP&CR on a suite of job shop scheduling CSPs proposed in [41]. 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 group of problems differs in two respects: (1) spread of the release and due dates among jobs; (2) number of a-priori bottlenecks. The spread is controlled by varying the amplitude of the intervals within which release and due dates are generated. Three spread levels are introduced: wide (w), narrow (n), and null (0), i.e., both release and due date intervals are collapsed to single points. Aside from different spread levels of release and due dates, the benchmark also considered one and two a-priori bottlenecks conditions. Each problem in the benchmark 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 the effects of initial solution configuration in the system, (4) investigated CORA's scaling up characteristics on problems of larger sizes.

5.1 Effects of Coordination Information

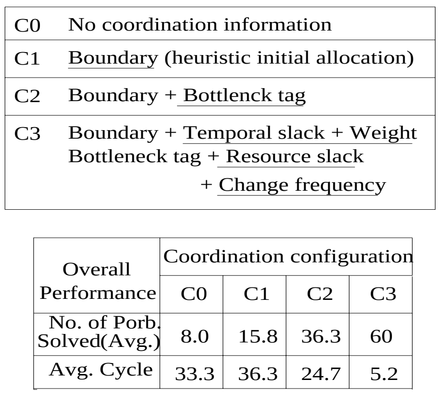
In order to investigate the effects of coordination information on the system's performance, we constructed a set of four coordination configurations.

- 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.

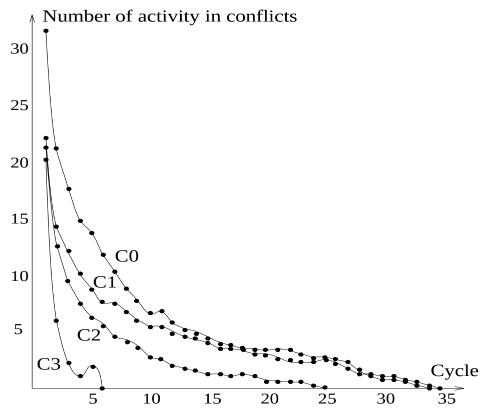
- C3 represents a complete configuration in which all coordination information is provided for resource agents and job agents.



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)

Figure 17: Comparative Performance between Coordination Configurations

Figure 17 shows the comparative performance of different configurations on the suite of benchmark problems. The additional coordination information for each configuration is underlined in Figure 17 (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 17 (ii) shows, for different coordination configurations, the successful overall problem solving processes[8] 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.

5.2 Comparison with Other Scheduling Techniques

CORA was compared to four other heuristic search scheduling techniques, ORR/FSS, MCIR, CPS, and PCP. ORR/FSS [41] 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 [48]. Min-Conflict Iterative Repair (MCIR) [34] 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) [38] employs a search space analysis methodology based on stochastic simulation which iteratively prunes the search space by posting additional constraints. Precedence Constraint Posting (PCP) [43] 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 [37], are also included for comparison.

Table 1 reports the number of problems solved[9] 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[38]. All CPU times were obtained from Lisp implementations on a DEC 5000/200. In particular, CORA was implemented

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

[9]PCP's performance is sensitive to the parameters that specify search bias [43].

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.[10] 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 [43]. 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

5.3 Effects of Initial Solution Configuration

CORA essentially employs reactive agents to iteratively repair current solution until a valid solution evolved. Agents' local interactions direct the solution repairing process, which is different from iterative improvement (hill-climbing) methods [50] [35] that make local changes to reduce a cost function. Previous study [36] indicates that the goodness of a rough initial solution has great effects on the performance of iterative improvement methods. The performance of MCIR based on randomly generated initial solutions provides evidence to this observation.

In order to investigate the effects of initial solution configurations on the system's performance, we conducted experiments with both heuristic and random initial configurations. In heuristic initial configuration, resource agents allocate free resource intervals to an activity with the earliest due date. In random initial configuration, the selection of activity for free resource intervals was random among the eligible activities (i.e., those activities with release dates before the current start time of free resource interval). CORA was ran on each problem for 10 randomly generated initial solution configurations.

Table 2 shows CORA's performance on both heuristic and random initial solution configurations in terms of the number of cycles required to find a solution for a problem. CORA

[10]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.

was able to solve all 60 problems for both heuristic and random initial solution configurations. Within each problem category, the number of cycles required to find a solution for a problem was averaged. The results show that CORA has equal overall performance with heuristic and random initial solution configurations. We experimentally confirmed that CORA is barely effected by its starting point of search.

	w/1	w/2	n/1	n/2	0/1	0/2	Average
Heuristic	5.6	6.3	4.0	4.9	4.4	6.2	5.2
Random	5.4	6.5	4.9	4.7	4.3	6.2	5.3

Table 2: Comparative Performance between Initial Solution Configurations

5.4 Scaling Up Characteristics

Because of its asynchronous, interactive nature, CORA does not render itself to algorithmic complexity analysis. In order to experimentally investigate CORA's scaling up characteristics, we used the same problem generator function producing the benchmark problems to produce two sets of 60 problems that involve 250 and 500 variables (e.g. 100 factory jobs on 5 machines which is a problem of realistic size). These problems exhibit similar scheduling conditions to the benchmark problems. Figure 18 shows CORA's performance on these larger sized problems, which exhibits favorable, near-linear scaling-up characteristics. The results seem to indicate that CORA's search mechanism is polynomial to the size of the search space.

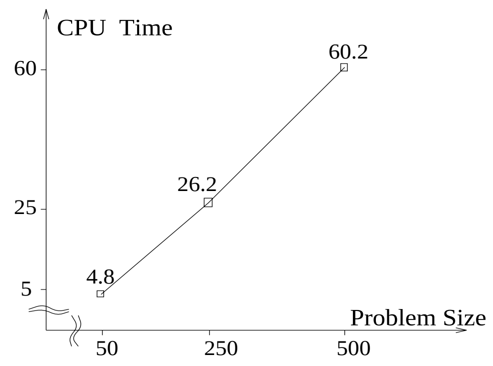


Figure 18: CORA's Scaling Up Property

5.5 Evaluation

The primary purpose of the experiments is to evaluate the utility of CP&CR as a tool to divide and conquer tightly coupled problems, such as job shop scheduling. CP&CR provides a framework in which an NCSP can be decomposed and assigned to different problem solving agents according to disjoint functionality (constraint types) and overlapping responsibility (variable subsets). This decomposition leads to utilization of interaction characteristics that is further exploited to achieve problem solving by asynchronous and well coordinated local interactions. Experimental results show that CORA, facilitated by coordination information, efficiently performed divide-and-conquer on the set of job shop scheduling problems. In addition, CORA exhibits two favorable properties: (1) the performance is relatively insensitive to the initial solution configuration, (2) computational cost seems to be near-linear to the problem size. 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.

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.

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.

6 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

[1] Mark A. Bedau. Philosophical aspects of artificial life. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 494–503, Paris, France, 1992.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, April 1986.

[3] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[4] S. Cammarata, D. McArthur, and R. Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the IJCAI-83*, pages 767–770, 1983.

[5] B. Chaib-Draa, B. Moulin, R. Mandiau, and P. Millot. Trends in distribtued artificial intelligence. *Artificial Intelligence Review*, 6:35–66, 1992.

[6] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. Distributed optimization by ant colonies. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 134–142, Paris, France, 1992.

[7] S. E. Conry, R. A. Meyer, and V. R. Lesser. Multistage negotiation in distributed planning. In *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufmann, San Mateo, CA, 1988.

[8] D. D. Corkill. *A framework for organizational self-design in distributed problem solving networks.* PhD thesis, University of Massachusetts, Amherst, 1982.

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

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

[11] K. S. Decker, E. H. Durfee, and V. R. Lesser. Evaluating research in cooperative distributed problem solving. In *Distributed Artificial Intelligence, Vol. 2*, pages 485–519. Pitman, San Francisco, CA, 1989.

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

[13] Y. Demazeau and J.-P. Muller. From reactive to intentional agents. In Demazeau and Muller, editors, *Decentralized AI 2*. Elsevier, North-Holland, 1991.

[14] Alexis Drogoul and Christophe Dubreuil. A distributed approach to n-puzzle solving. In *Proceedings of the 12th International Workshop on Distributed AI*, 1993.

[15] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989.

[16] 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.

[17] Jacques Ferber and Alexis Drogoul. Using reactive multi-agent systems in simulation and problem solving. In *Distributed Artificial Intelligence: Theory and Praxis*, pages 53–80. Kluwer Academic, Boston, 1992.

[18] Stephanie Forrest, editor. *Emergent Computation – Self-Organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks.* MIT Press, 1991.

[19] 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.

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

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

[22] Khaled Ghedira and Gerard Verfaillie. A multi-agent model for the resource allocation problem: a reactive approach. In *Proceedings of ECAI-92*, pages 252–254, 1992.

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

[24] Hiroaki Kitano. Chanllenges of massive parallelism. In *Proceedings of IJCAI-93*, pages 813–834, 1993.

[25] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

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

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

[28] V. R. Lesser and D. D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Mag.*, 4(3):15–33, 1983.

[29] V. R. Lesser and D. D. Corkill. Distributed problem solving. In S. C. Shapiro, editor, *Encyclopedia in Artificial Intelligence*, pages 245–251. Wiley, New York, 1987.

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

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

[32] D. McArthur, R. Steeb, and S. Cammarata. A framework for distributed problem solving. In *Proceedings of the AAAI-82*, pages 181–184, 1982.

[33] 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.

[34] 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.

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

[36] Paul Morris. On the density of solutions in equilibrium points for the queens problem. In *Proceedings of AAAI-92*, pages 428–433, 1992.

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

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

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

[40] H. Van Dyke Parunak. Manufacturing experience with the contract net. In *Distributed Artificial Intelligence*, pages 285–310. Morgan Kaufmann, Los Altos, 1987.

[41] 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.

[42] D. Smith and M. Broadwell. Plan coordination in support of expert systems. In *Proceedings of DARPA Knowledge-Based Planning Workshop*, Austin, TX, 1987.

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

[44] Luc Steels. Towards a theory of emergent functionality. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior - From Animals To Animats*, pages 451–461. MIT Press, 1991.

[45] Katia Sycara. Resolving goal conflicts via negotiation. In *Proceedings of AAAI-88*, pages 245–250, 1988.

[46] 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.

[47] Sarosh Talukdar. Asynchronous teams. Technical Report EDRC 18-39-92, Engineering Design Research Center, 1992.

[48] 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.

[49] 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.

[50] M. Zweben. A framework for iterative improvement search algorithms suited for constraint satisfaction problems. In *Proceedings of the AAAI-90 Workshop on Constraint-Directed Reasoning*, 1990.