Coordination of multiple agents for production management*

Jyi-Shane Liu[a] and Katia P. Sycara[b]

[a]*Department of Computer Science, National Chengchi University, Taipei, Taiwan*

E-mail: jsliu@cs.nccu.edu.tw

[b]*The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA*

E-mail: katia@cs.cmu.edu

The new challenges of "Agile Manufacturing" and distributed decision making entailed by decentralized organizations led to our interest in the study of computational cooperative problem solving models and coordination techniques for distributed production management. The goal of our research is to address the technical need of distributed production management and develop appropriate computational approaches to support adaptive, cost-effective responsiveness. In particular, we focus on the challenging problem of job shop scheduling, which has been one of the primary foci of production scheduling research. This paper presents a multi-agent problem solving model and an effective coordination technique for job shop scheduling. The model involves a group of agents; each agent is associated with either a job or a resource. A solution to a production scheduling problem is the result of coordinated conflict resolution in the iterative and asynchronous multi-agent decision making process. It is well known in distributed systems research that for tightly interacting, non-decomposable problems, such as job shop scheduling, the need for communicating partial solution results among parts of the system rapidly degrades system performance. On the other hand, limiting communication degrades solution quality. One can limit communication by employing shared memory, but this has the drawback that the shared memory becomes a bottleneck and, in addition, using shared memory limits decentralization. In our approach, we judiciously balance the above concerns. We limit inter-agent communication through a scheme that employs *efficient, small and distributed shared memories, each of which is associated with and shared by a limited number of agents*. We also exploit problem characteristics (e.g. disparity among subproblems) to design an effective coordination technique for the job shop scheduling problem. We have evaluated the utility of our approach through extensive experimentation on a variety of job shop constraint satisfaction and optimization problems with different optimization objectives. Our results show that our approach outperforms or gives comparative results with other state-of-the-art scheduling techniques on benchmark problems.

Keywords: multi-agent problem solving, coordination, production scheduling

* This research has been sponsored in part by ARPA Grants F30602-90-C-0119 and F33615-93-1-1330.

1. Introduction

In the era of global economic markets, overseas competition has impelled the manufacturing industry to move toward the concept of "Agile Manufacturing" [17]. The capabilities to effectively and adaptively operate modern production facilities become increasingly important to the survival and success of business organizations. The need for new logistics and production management techniques to respond rapidly to changing market demands has spawned decentralized organizations and entailed distributed decision making for the common organization goals, e.g., reduce inventory, meet order due dates, reduce resource idleness, etc. Advances of communication services, both for human decision makers and computational devices, further enhance the plausibility of de-centralization. However, success of distributed management hinges on effective coordination, which has not been well studied and well understood in computational approaches. This new challenge leads to our interest in the study of computational cooperative problem solving models and coordination techniques for distributed production management.

Distributed Artificial Intelligence (DAI) is concerned with the coherent problem solving behavior of a group of related individuals (computational agents) [12, 14]. As computer applications begin to permeate into our everyday activities, DAI research receives more and more interest. Recent DAI research has studied important issues, such as negotiation [40], interaction protocol [7], coordination [8, 18], when to communicate with other agents, what to communicate, and how to model other agents, etc. These studies have produced significant results and laid the foundation for real-world cooperative problem solving. Facilitated by the development of networked computer systems and practical concurrent processing architectures, realistic cooperative problem solving has been realized in increasingly complex domains, such as design [1, 2], flexible manufacturing systems [30, 32], office information systems [25, 29, 31], and communication network management [15, 39], etc.

The goal of our research is to address the technical need of distributed production management and develop appropriate computational approaches to support adaptive, cost-effective responsiveness. In particular, our current study focuses on the challenging problem of job shop scheduling, which has been one of the primary foci of production scheduling research. Production scheduling has significant implications in manufacturing management. The effective scheduling of a facility enables faster response to customer demands, reduces its work-in-process, and increases its throughput and, therefore, is instrumental in achieving objectives such as gaining increased market share and increasing return on investment.

This paper presents a multi-agent problem solving model and an effective coordination technique for job shop scheduling. The model involves a group of agents; each agent is associated with either a job or a resource. Lateral coordination among agents defines the process of distributed production management. A solution to a

production scheduling problem is the result of coordinated conflict resolution in the iterative and asynchronous multi-agent decision making process.

It is well known in distributed systems research that for tightly interacting, non-decomposable problems, such as job shop scheduling, the need for communicating partial solution results among parts of the system rapidly degrades system performance. On the other hand, limiting communication degrades solution quality. One can limit communication by employing shared memory, but this has the drawback that the shared memory becomes a bottleneck and, in addition, using shared memory limits decentralization. In our approach, we judiciously balance the above concerns. We limit inter-agent communication through a scheme that employs *efficient, small and distributed shared memories, each of which is associated with and shared by a limited number of agents*.

We argue that design of appropriate coordination strategies should be based on characteristics of the problem structure. We examine one of the recurrent problem structures that involves task disparity among agents, e.g., some of the agents have dominant effects on the process of solution generation. We developed coordination techniques that take advantage of disparity among agents to conduct effective lateral coordinated negotiation. We use negotiation as a metaphor so that constraint conflict resolution can be viewed as a repeated negotiation process. This view has been expressed in other DAI conflict resolution situations [5].

The cooperative problem solving model and the coordination technique are applied to both job shop scheduling constraint satisfaction (with non-relaxable time windows) and constraint optimization (with objectives) problems. We conducted computational experiments on benchmark job shop scheduling problems and compared the results to other centralized scheduling techniques. The goal is to rigorously verify the effectiveness of our cooperative problem solving approach on predictive scheduling, which is an essential component of production management. We intend to build upon the results and further develop our cooperative problem solving approach to address other areas of production management, such as reactive scheduling, inventory control, job routing, resource allocation, etc.

In the remainder of the paper, section 2 defines job shop scheduling (JSS) problems and describes a cooperative problem solving model for the problem. Section 3 presents the design of the coordination techniques for the JSS constraint satisfaction problem, which include the disparity structure of job shops, the coordination procedure, and the set of useful coordination information. Section 4 describes the conflict resolution heuristics of the agents. Section 5 gives an overview of the group negotiation process. In section 6, we extend the coordination techniques to the JSS constraint optimization problem. In sections 7 and 8, we report the experimental results of our approach on benchmark JSS constraint satisfaction and constraint optimization problems, respectively. In section 9, we discuss the evaluation of the experimental results. In section 10, we overview representative related work. Finally, in section 11, we conclude the paper.

2. Job shop scheduling

Production scheduling deals with organizing possibly related production operations over time and across limited resources. The diversity of human and machine activities are modeled in a wide range of scheduling problems of different conditions. We are concerned with a subset of scheduling problems, known as job shop, that has been one of the primary foci of production scheduling research.

2.1. Classic job shop model

To present the classic job shop model, we introduce several basic notions. We adopt the definitions given in [6].

Operation: An operation is an elementary task to be performed.

Processing time: The processing time of an operation is the amount of time required to process the operation; in most cases, setup times are independent of operation relations and are included in the processing times.

Job: A job is a set of operations that are interrelated by precedence constraints derived from technological restrictions.

Machine: A machine is a piece of equipment, a device, or a facility capable of performing an operation.

Release time: The release time (or release date) of a job is the time at which the job is released to the shop floor; it is the earliest time at which the first operation of the job can begin processing.

Due date: The due date of a job is the time by which the last operation of the job should be completed.

Completion: The completion time of a job is the time at which processing of the last operation of the job is completed.

Schedule: A schedule is a specification of the execution of each operation on a particular machine at a specific time interval. A *feasible* schedule is a schedule that observes all problem constraints, e.g., job release time, operation precedence relations, and machine capacity.

Classic job shop represents a manufacturing production environment where a set of m jobs $J = \{J_1,\ldots,J_m\}$ have to be performed on a set of n machines (or resources) $R = \{R_1,\ldots,R_n\}$. Each job J_i is composed of a set of sequential operations opr_{ij}, $j = 1,\ldots,m_i$, $m_i \leq n$, and can only be processed after its release date RD_i. Each resource R_s has only unit capacity and can process only one operation at a time. Each operation opr_{ij} has a deterministic processing time p_{ij} and has been pre-assigned a unique resource which may process the operation. Since a job consists of a set of operations that has to be performed in sequential order, it is conceptually convenient to envision

a job entering the shop, visiting different machines to have its correspondent operations performed, and then leaving the shop. A job's *routing* is the sequential set of resources that the job visits before its completion. The *arrival time* of a job at a machine is the time at which the job leaves the previous machine in its routing, and is equivalent to the *ready time* of the operation to be performed on the machine.

The model provides only a basic description of actual job shops and does not elaborate on more complex conditions. In order to focus the study on the effects of scheduling and to allow generalization of the experimental results, most research on job shop scheduling share the following assumptions to simplify the model.

(1) A job has a fixed ordering of operations.

(2) A job does not visit the same machine twice.

(3) An operation may have at most one other operation immediately preceding or succeeding it. (No assembly operations.)

(4) An operation of a job can be started only if its immediate preceding operation, if any, is completed. (No overlapped operations.)

(5) An operation can be processed by only one machine. (No alternative resource.)

(6) An operation can not be preempted during its processing.

(7) Operations do not require explicit setup times.

(8) The machines have unit capacities in the entire interval of scheduling (no machine down time).

Job shop scheduling problems [10] involve synchronization of the completion of m jobs J on n resources R (machines). The problem (hard) constraints of job shop scheduling include (1) *operation temporal precedence* constraints, i.e., an operation must be finished before the next operation in the job can be started, (2) *release date* constraints, i.e., the first operation of a job can only begin after the release date of the job, and (3) *resource capacity constraints*, i.e., resources have only unit processing capacity. A solution of the job shop scheduling problem is a feasible schedule, which assigns a start time st_{ij} and an end time et_{ij} to each operation opr_{ij}, that satisfies all problem constraints.

Formally, job shop scheduling problems are defined as:

Given

- J_i, $i = 1,\ldots,m$, a set of m jobs.
- R_k, $k = 1,\ldots,n$, a set of n resources (with unit capacity).
- $J_i = \{opr_{ij}^k\}$, $j = 1,\ldots,m_i$, $m_i \leq n$. Each job J_i consists of a set of sequential operations. An operation opr_{ij}^k is the jth operation in the job J_i and requires the use of resource R_k for a processing time of p_{ij}.
- rt_i, $i = 1,\ldots,m$, a set of release times, where rt_i is the release time of J_i.

Find st_{ij}, $i = 1,\ldots, m$, $j = 1,\ldots, m_i$, *such that*

- $rt_i \le st_{i1}$, and
- $st_{ij} \ge et_{i(j-1)}$ (or equivalently, $st_{ij} \ge st_{i(j-1)} + p_{i(j-1)}$, and
- $(et^k_{pq} \le st^k_{ij}) \vee (et^k_{ij} \le st^k_{pq})$, $p \ne i$.

A schedule can be visualized using *Gantt charts*, where operations are organized along the time line from the perspectives of either jobs or resources. Each box represents an operation with length proportional to its processing time and is marked by a unique identification code. We use a three-digit code, where the first digit is the job number, the second digit is the operation number within the job, the third digit is the resource that processes the operation. For example, opr^B_{21} represents the first operation of J_2 and is processed by R_B.

Figure 1 shows the job Gantt chart of a schedule where only operation precedence constraints and job release date constraints are considered. A vertical arrow in front of each time line represents the release date of each job. The schedule is infeasible because resource capacity constraints are not respected. For example, according to

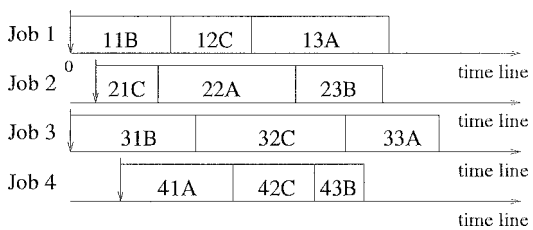

Figure 1. Job Gantt chart of an infeasible schedule.

the schedule, R_B will process part of opr^B_{11} and opr^B_{31} at the same time, which is not feasible. Figure 2 shows the resource Gantt chart of a schedule where only resource capacity constraints are considered. The schedule is also infeasible because job release date constraints and operation precedence constraints are not respected. For example, according to the schedule, R_C will process opr^C_{12} before opr^B_{11} has been finished by R_B, which is not feasible. Also, R_C will not be able to process opr^C_{21} as scheduled because J_2 has not been released at time 0.

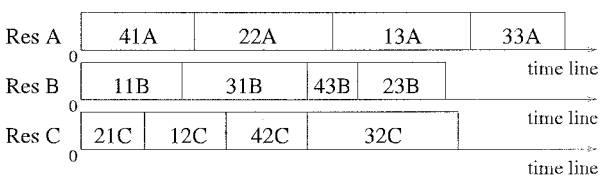

Figure 2. Resource Gantt chart of an infeasible schedule.

We use the above examples to illustrate that when solving job shop scheduling problems, two aspects of constraints that involve job and resource, respectively, must be considered simultaneously. A feasible schedule is shown in figure 3. The job Gantt chart of the schedule displays when the operations of a given job are processed. The resource Gantt chart of the schedule lays out the process of a given resource performing on various operations.

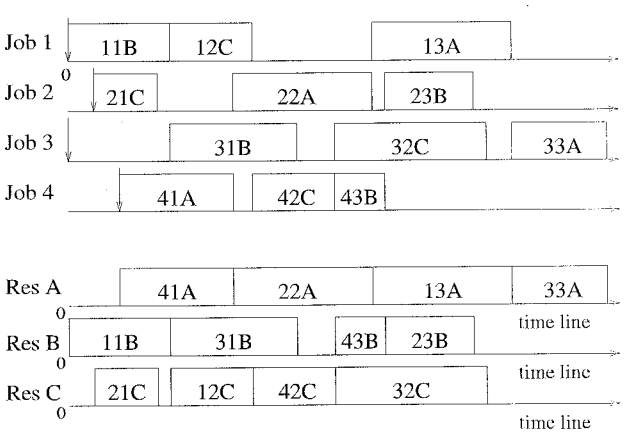

Figure 3. Job and resource Gantt charts of a feasible schedule.

In general, scheduling problems are NP-hard [11]. They require time exponential in the input length. Consequently, the time required to solve problem instances of reasonable size can be on an astronomical scale. For example, consider an exponential relation of time 2^n to problem size in the number of operations n, and a time unit of a microsecond. The *worst case* time required to solve the problem grows rapidly from 0.001 second for size 10 (n = 10), to 18 minutes for size 30, and to 35.7 years for size 50. This exponential explosion can not be alleviated by realistic speed increase of computation. To double the size of a problem that can be solved within 18 minutes would require a 10^9-fold increase in speed. A practical job shop scheduling problem typically involves hundreds of operations. Therefore, almost all realistic scheduling techniques resort to heuristic search that does not guarantee completeness.

Given a job shop scheduling problem, the number of feasible solutions can be enormous. For example, for a problem with m jobs of n operations on n resources, each resource has $m!$ possible processing sequences. The total number of possible schedules is $(m!)^n$ since all precedence constraints between operations can be satisfied by right shifting operations toward the end of the time line. We need to specify conditions that restrict admissible schedules.

In general, there are two types of scheduling problems, e.g., satisfaction problems and optimization problems. In satisfaction problems, each job is assigned a non-relaxable due date. An admissible solution is a schedule in which all jobs are finished

before their due dates. All admissible solutions are equally good. Usually, we are looking for any one and only one of the admissible solutions. In optimization problems, feasible schedules are evaluated according to objective criteria that reflect the economic goals of organizations. The best solution is a schedule with minimum objective cost. Usually, it is not realistic to find an optimal schedule [1)] because of the enormous numbers of feasible schedules. The practical goal is to find a schedule as good as possible. In this study, we consider both job shop scheduling satisfaction and optimization problems.

2.2. Objective criteria

We introduce two objective functions that are most commonly used in the literature of job shop scheduling. First, we describe two primitive measures that are essential for the objective functions that we consider.

Completion time C_i is the time at which a job J_i completes its processing. It is the end time of the last operation of J_i.

Tardiness T_i is the non-negative amount of time by which the completion time of J_i exceeds its due date dd_i. If J_i is completed earlier than dd_i, the tardiness is zero. ($T_i = max[0, (C_i - dd_i)]$.

We consider two objective functions:

Inventory INV is the in-progress inventory cost of jobs.

$$\text{INV} = \sum_{i=1}^{m} inv_i \times (max[C_i, dd_i] - st_{i1}),$$

where m is the number of jobs, inv_i is a marginal inventory cost introduced to each job by its first operation, st_{i1} is the time at which a job J_i begins its processing, or the start time of the first operation of J_i.

Weighted tardiness T_{wt} is the sum of proportional tardiness multiplied by job importance.

$$T_{wt} = \sum_{i=1}^{m} w_i T_i,$$

where w_i is the weight of a job J_i.

We study job shop scheduling optimization problems based on a cost function that combines both inventory cost and weighted tardiness cost of the schedule. This cost function was proposed in [33] as a more realistic cost model that directly accounts

[1)] Except for small sized problems and special objective functions, such as makespan.

for the tardiness cost, in-process inventory cost, and finished-goods inventory cost introduced by each job. Weighted tardiness cost and inventory cost are usually conflicting objectives.[2)] In general, it is more difficult to optimize a composite objective than just one objective. We adopt this cost function in order to test the capability of our cooperative problem solving approach to optimize conflicting objective.

2.3. A cooperative problem solving model for job shop scheduling

Job shop scheduling problems involve job and resource constraints. Intuitively, it is convenient to assign separate agents that would be in charge of these two aspects of constraints. We propose a model of distributed job shop scheduling in which each job is assigned to a *job agent* and each resource to a *resource agent*. Therefore, the number of agents involved in solving a job shop problem is equal to the sum of the number of jobs and the number of resources in the problem. Job agents are concerned with constraints of a job's aspect, e.g., job release dates, due dates, and operation precedence relations. Resource agents take care of constraints of a resource's aspect, e.g., resource capacity. These agents engage in a coordinated decision making process to solve a scheduling problem. This model seems to be appropriate for the following reasons.

(1) Each agent corresponds to a real entity. This provides a closer management of an entity in its situated environment and, therefore, facilitates potential real-time responsiveness. For example, a resource agent can monitor its local condition (e.g., machine breakdown, processing time variation) and update its processing decisions accordingly. A job agent can respond to due date changes and make necessary revisions on operation processing.

(2) Each agent can make decisions based on its own strategies and its local situation. This provides modularity for constructing the system. For example, we may want a resource agent to manage the resource differently depending on its contention level. Similarly, we may want a job agent to act with different strategies according to the job's importance.

(3) This model provides an appropriate level of granularity to disentangle the problem constraints and analyze how they interact with each other. By examining how a job interacts with a resource and scrutinizing how resources interact with each other through jobs, and vice versa, we hope to obtain information that can facilitate our understanding of the interaction structure of job shop and the design of communication messages and coordination strategies that enable more effective problem solving.

This model has three unique features for cooperative problem solving. First, each agent is concerned with only one type of constraint among a subset of decision vari-

[2)] This composite objective demands just-in-time scheduling, which is one of the irregular objectives [24].

ables (operations). A study based on this model may provide implications to other group decision problems that involve several parties of different interests or concerns. Second, an operation is governed both by a job agent and a resource agent. Changes of operation start times by job agents may result in constraint violations for resource agents and vice versa. Therefore, the start times of operations must be negotiated by job and resource agents. Third, there is no authority among agents. All agents have equal right to assign a value (start time) to a decision variable (operation) under its jurisdiction. Therefore, a solution is a value assignment of all decision variables that all agents agree upon.

We consider a cooperative problem solving approach based on an iterative negotiation process of agents' value assignment to decision variables. Negotiation proceeds by proposal and counter-proposal of value assignment between resource agents and job agents. Resource agents are concerned with resource capacity constraints only. Job agents are concerned with job release dates, due dates, and operation precedence constraints. In optimization problems, job due date constraints are relaxed by job agents. Whenever an agent is not satisfied with the current value assignment, it presents a counter-proposal by actually changing the values of a subset of decision variables to satisfy its own constraints. If an agent is satisfied with the current value assignment, it does not change any value assignment. Within a round of negotiation (or an iteration cycle), resource agents and job agents take turns in being active in the negotiation, e.g., they each have one chance to examine the current value assignment and propose necessary changes. While agents of different types of constraints negotiate alternatively, agents of the same type can be active simultaneously, each working independently. A solution to the satisfaction problems is the result of a successful negotiation process between job agents and resource agents. In optimization problems, one of the resource agents assumes a special role of presenting a proposal of low global objective cost. The approach is called COordinated Negotiation Agents (CONA).

3. The design of coordination techniques

Since the agents are engaged in an iterative negotiation process, their proposals must be coordinated to facilitate convergence to an agreement. We developed a coordination scheme that exploits task disparity in job shop. We distinguish between coordination procedure and coordination information that constitute a coordination scheme. A coordination procedure is a desired negotiation pattern of agents. Coordination information is the set of exchanged information among agents that affects agents' proposals and facilitates desired negotiation results.

3.1. Task disparity

In many application domains, problems often exhibit special structures that can be exploited to facilitate more effective problem solving. One of the most recurrent

structures involves *disparity* among subproblems. We define disparity as the condition where some quantifiable characteristics of subproblems exhibit noticeable deviation from the average. In the context of CONA, disparity of subproblems is exhibited by different levels of agents' flexibility to accept a proposal without constraint violation. For example, if a resource is used only occasionally for a short period of time, chances are the resource agent would not detect a capacity constraint violation when a job agent changes the start time of its operation using the resource. On the other hand, if the resource is a bottleneck resource which is used most of the times for a long interval, a start time change on an operation would most likely create constraint violation for the resource. In other words, agents that are assigned to bottleneck resources are more constrained than agents that are assigned to non-bottleneck resources in the negotiation process. Therefore, there is a disparity between bottleneck resource agents and non-bottleneck resource agents.

Besides disparity of ease of accepting proposals in the negotiation process, bottleneck resources also have dominant effects on both the admissibility and the quality of a schedule. For example, figure 4 shows a simple job shop scheduling problem which includes only three jobs, J_i, i = 1, 2, 3, on three resources, R_j, $j = x, y, z$. Each box is an

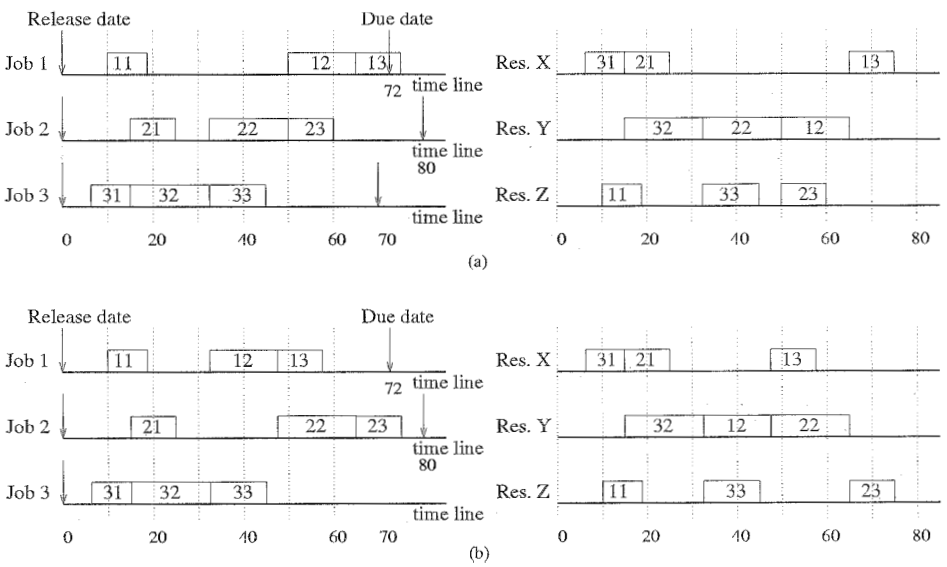

Figure 4. Dominant effects of bottleneck resources.

operation, with the first number representing the number of the job and the second number representing its sequence within the job, i.e., opr_{23} is the third operation of J_2. An operation is under the jurisdiction of a job agent and a resource agent. R_Y is considered as a bottleneck resource because of its high contention. If the problem is a

satisfaction problem, the schedule in (a) is not admissible because J_1 is not finished before its due date. If the problem is a tardiness optimization problem, the schedule in (b) is a better schedule than the schedule in (a) since it does not have any tardiness. The decision of R_Y agent to process opr_{12} before opr_{22} is most critical in generating the schedule in (b).

3.2. Coordination procedure

In view of the iterative proposal, counter-proposal process based on resolving constraint conflicts of individual agents and the task disparity on bottleneck resources, we developed a coordination procedure that promotes rapid convergence of negotiation by considering the following principles of interaction control.

(1) *Least disturbance* – Each agent's proposal of actually changing the start time of an operation potentially disturbs others' constraint satisfaction status. To minimize reciprocal harmful interaction, agents' proposals should take into account the constraints of others regarding current value assignment.

(2) *Islands of reliability* – A proposal by the most constrained agent is more likely to be part of a global solution than that of less constrained agents. In traditional constraint satisfaction literature [27], there was a notion of a most constrained variable which has the fewest possible values. The intuition was that instantiating the most constrained variable seemed to be more critical than instantiating other less constrained variables; in addition, the instantiation would be more likely to be correct than the instantiation of variables with large domains of possible values. We extend this intuition to CONA, in which a proposal from the most constrained agent seems to be more reliable and, therefore, can serve as an anchor of interaction. This intuition also incorporates domain knowledge of job shop scheduling where the processing sequence on the bottleneck resource has the most impact on the admissibility and quality of the schedule. This policy suggests that group agreement should be reached by a process of negotiation based on *islands of reliability*, and modifying *islands of reliability* only when group agreement is heuristically determined as infeasible under the current anchoring proposal.

(3) *Loop prevention* – Looping behaviors, such as oscillatory value changes by a subset of agents, should be prevented. This is a common desired property of all distributed systems.

We now give a conceptual overview of how these principles guide coordination of the negotiation agents.

3.2.1. Least disturbance

Least disturbance corresponds to an attempt to minimize ripple effects of agents' proposals of actual value changes. To reach group agreement, the number of

unsatisfied agents within a round of negotiation should be gradually reduced to zero. While an agent always becomes satisfied in an iteration cycle since it assigns values to its variables to satisfy only its own constraints, its proposal may cause constraint violations to 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 an agent's heuristics of conflict resolution (see section 4). The least disturbance principle is operationalized during conflict resolution in two ways. First, an agent changes the values of as few variables as possible. Second, for a given selected variable, an agent changes the value assignment such that it deviates from the previous value the least possible.

3.2.2. Islands of reliability

An *island of reliability* is a proposal by the most constrained agent and is more likely than others to be part of the solution. Islands of reliability provide anchoring for reaching group agreement in terms of propagating more promising partial solutions and are changed less often.[3)] In CONA, the island of reliability refers to start times assignment on operations using the bottleneck resource by the agent in charge of the resource. An island of reliability is used as an anchor of interaction during the iterative negotiation process between job agents and resource agents.

Recall that each operation is governed by both a job agent and a resource agent. Therefore, the island of reliability is part of every job agent's concerns in satisfying its constraints. To use the island of reliability as an anchor of negotiation, we design job agents in such a way that they would not propose to change value assignment on the island of reliability during their conflict resolution unless special conditions occur. Instead, they would propose to resolve their conflicts by changing the value assignment on operations that are not using the bottleneck resource. We use the negotiation efforts between job agents and non-bottleneck resource agents as conditions of when job agents can propose to change value assignment on the island of reliability. To measure the negotiation efforts between job agents and non-bottleneck resource agents, we associate a counter with each operation using the non-bottleneck resources. The counter associated with an operation records the number of times that a non-bottleneck resource agent has changed the value assignment of the operation. In other words, when a non-bottleneck resource agent proposes to change the value assignment of an operation during its conflict resolution, it increases the counter associated with the operation by one. We assign a heuristic threshold of counter value to contain the negotiation efforts of job agents and non-bottleneck resource agents between each

[3)] Blackboard systems (e.g., Hearsay-II speech-understanding system [9]) have used the notion of solution *islands* to conduct an incremental and opportunistic problem solving process. Partial solution islands emerge and grow into larger islands, which it is hoped will culminate in a hypothesis spanning the entire solution structure.

configuration of the island of reliability. When a job agent detects an operation with a counter value exceeding the threshold in conflict with an operation using the bottleneck resource, the job agent would change the value assignment of the operation using the bottleneck resource. In response to the proposal by the job agent, the bottleneck resource agent, if it finds itself in conflicts, would propose a new configuration of the island of reliability. All counters are reset to zero by the non-bottleneck resource agents. The job agents and the non-bottleneck resource agents resume their negotiation in the hope of an agreement under the new anchoring proposal.

3.2.3. 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 job and non-bottleneck resource agents may be involved in cyclic proposals of value changes in order to find an agreement with bottleneck resource agents' decisions. Secondly, the bottleneck resource agent might propose value changes in a cyclic way.

The first type of looping behavior is interrupted by job agents when the counter of a conflicting non-bottleneck operation exceeds a threshold. To prevent the second type of looping behavior, the bottleneck resource agent keeps a history of its value changes so that it does not repeat the same proposal.

3.2.4. Negotiation pattern

With the three principles of interaction control, the coordination procedure of the negotiation agents is designed to emulate a hierarchical organization with the bottleneck resource agent(s) on the top level and the job agents and the non-bottleneck resource agents on the bottom level. Job agents interact vertically with the bottleneck resource agent(s) and laterally with the non-bottleneck resource agents. There is no direct interaction between bottleneck and non-bottleneck resource agents. By proposing a configuration of the island of reliability, the bottleneck resource agent(s) exerts constraints to job agents, who then negotiate with the non-bottleneck resource agents for an agreement. When an agreement is heuristically determined as infeasible, job agents break the restriction of the bottleneck resource agent(s) by modifying the configuration of the island of reliability. If a bottleneck resource agent finds itself not satisfied with the configuration of the island of reliability modified by job agents, it would propose a new configuration, which again is regarded as new constraints by job agents. The negotiation process continues until all agents are satisfied, which means that an agreement has been found.

The coordination procedure is intended to quickly find an agreement among agents in problems with a very large combinatorial space, such as job shop scheduling, but cannot guarantee that the agents will find a solution if the solution exists. However, in our experimental study, we show that CONA usually converges very fast in problems with noticeable bottlenecks.

3.3. *Coordination information*

In order to coordinate their proposals of value changes, agents need to exchange information. We developed a set of useful coordination information for negotiation agents in job shop scheduling. In DAI research, two types of communication have been used, e.g., message passing and shared memory. Message passing limits unnecessary communication. However, in situations such as tightly coupled problems, where frequent communication is required, message passing results in great computational overhead. Shared memory reduces computational overhead but may become a communication bottleneck and limit decentralization.

We judiciously balance the above concerns. We limit inter-agent communication through a scheme that employs efficient, small and distributed shared memories, each of which is associated with and shared by a limited number of agents. In particular, coordination information is associated with each operation and is used to express agents' views with regard to current value assignment of the operation. Coordination information *written* by a job agent on an operation is referenced by a resource agent, and vice versa, as shown in figure 5.

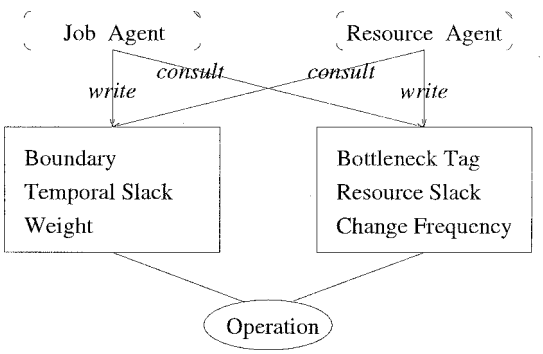

Figure 5. Coordination information.

Job agents provide the following coordination information for resource agents:

(1) *Boundary* of an operation is the interval between the earliest start time and latest finish time of the operation (see figure 6). It represents the overall temporal flexibility of an operation within the time window of a job's release date and due date and is calculated only once during initial activation of job agents.

(2) *Temporal Slack* of an operation is an interval between the current finish time of the previous operation and current start time of the next operation (see figure 7). It indicates the temporal range within which the start time of an operation may be assigned to without causing precedence constraint violations. (This is not guaranteed since temporal slacks of adjacent operations are overlapping with each other.)

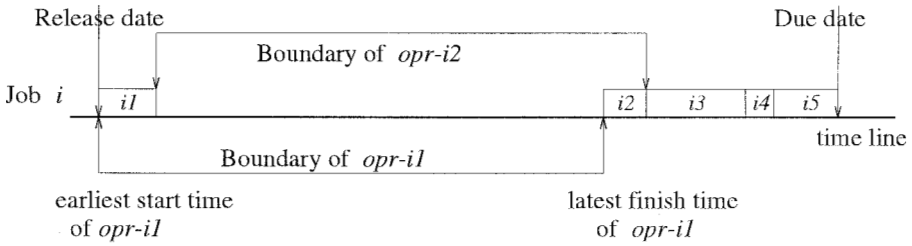

Figure 6. Coordination information: Boundary.

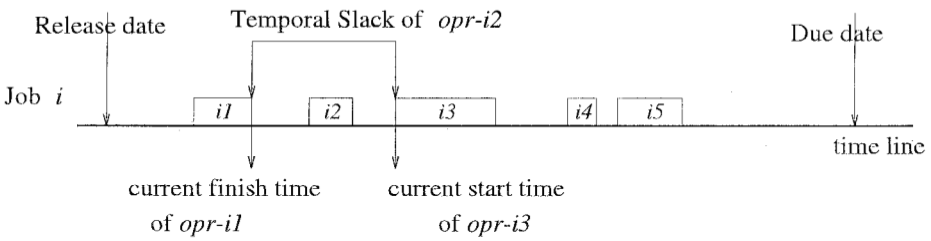

Figure 7. Coordination information: Temporal Slack.

(3) *Weight* of an operation is the weighted sum of relative temporal slack with respect to operation boundary and relative temporal slack with respect to the interval bound by the closest bottleneck operation. Denote the boundary of an operation opr_{ij} by $(st_{ij}^{b}, et_{ij}^{b})$. Denote the temporal slack of opr_{ij} as $(st_{ij}^{ts}, et_{ij}^{ts})$. Denote the interval of opr_{ij} restricted by the closest bottleneck operation as $(st_{ij}^{bb}, et_{ij}^{bb})$. If there is no bottleneck operation before opr_{ij}, then $st_{ij}^{bb} = st_{ij}^{b}$. If there is a bottleneck operation opr_{ik} before opr_{ij}, $k < j$, then $st_{ij}^{bb} = et_{ik} + \sum_{h=k}^{j-1} p_{ih}$, where et_{ik} is the current end time of opr_{ik}. If there is no bottleneck operation after opr_{ij}, then $et_{ij}^{bb} = et_{ij}^{b}$. If there is a bottleneck operation opr_{ik} after opr_{ij}, $k > j$, then $et_{ij}^{bb} = st_{ik} - \sum_{h=j+1}^{k-1} p_{ih}$, where st_{ik} is the current start time of opr_{ik}. The weight of an operation is defined as

$$e_{ij} = k_1 \times \frac{(et_{ij}^{b} - st_{ij}^{b}) - (et_{ij}^{ts} - st_{ij}^{ts}) - p_{ij}}{(et_{ij}^{b} - st_{ij}^{b})} + k_2 \times \frac{(et_{ij}^{ts} - st_{ij}^{ts}}{(et_{ij}^{bb} - st_{ij}^{bb})},$$

where p_{ij} is the processing time of the operation opr_{ij}; k_1 and k_2 are adjusting parameters. We arbitrarily set k_1 to 10 and k_2 to 5.

Weight of an operation is a measure of the likelihood of the operation “bumping” into an adjacent operation and an adjacent bottleneck operation, if its start time is changed. Therefore, a high weight of an operation represents a job agent’s preference for not changing the current start time of the operation. In figure 8,

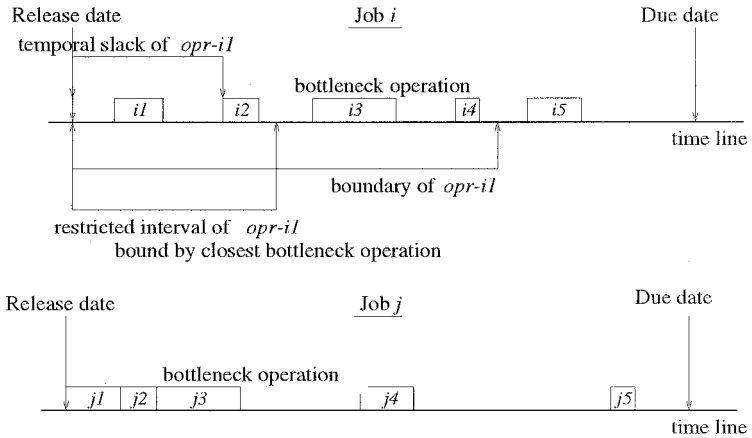

Figure 8. Coordination information: Weight.

both opr_{i3} and opr_{j3} are bottleneck operations. Opr_{j1} of J_j will have a higher weight than that of opr_{i1} of J_i. If both operations use the same resource and are involved in a resource capacity conflict, the resource agent will change the start time of opr_{i1} rather than start time of opr_{j1}.

Resource agents provide the following coordination information for job agents:

(1) *Bottleneck Tag* is a tag which marks that this operation uses a bottleneck resource. It indicates the status of operation being a bottleneck operation.

(2) *Resource Slack* of an operation is an interval between the current finish time of the previous operation and the current start time of the next operation in the resource's processing sequence (see figure 9). It indicates the range of the time interval in which the start time of the operation may be changed without causing capacity constraint violations. (This is not guaranteed since source slacks of adjacent operations are overlapping with each other.)

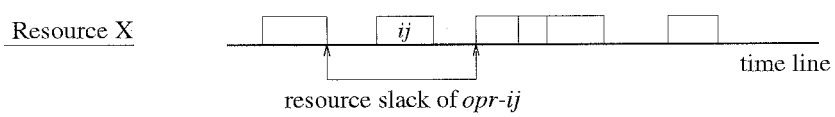

Figure 9. Coordination information: Resource Slack.

(3) *Change Frequency* of an operation is a counter of how many times the start time of this operation set by a job agent has been changed by a non-bottleneck resource agent. Whenever a non-bottleneck resource agent changes the start time of an operation, it increases change frequency of the operation by one. Conceptually, change frequency measures the negotiation effort of job and non-

bottleneck resource agents between each modification on start times of bottleneck operations. In addition, when a non-bottleneck resource agent detects a highly contended resource interval, change frequency can be used by the non-bottleneck resource agent to propagate its decisions on the dynamically arising bottleneck resource intervals[4)] by setting the counter to a number larger than some predetermined threshold. We heuristically set the threshold of change frequency to 3. When a job agent encounters a conflicting operation with change frequency larger than 3, it resolves conflict by changing the start time of the bottleneck operation and, therefore, signals to the bottleneck resource agent that a new configuration of the anchor proposal is required.

Coordination information among agents is associated with each operation by its responsible agents. When an agent is resolving constraint violations on an operation under its responsibility, the coordination information provided by the other agents that govern the same operation is used in proposing a new value assignment.

4. Agents' negotiation heuristics

Agents' negotiation 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 operation start time changes. The design of agents' negotiation heuristics incorporates the coordination procedure and the use of coordination information.

4.1. Negotiation heuristics of job agent

Job agents resolve precedence constraint violations using the following heuristics:

(1) Identify conflict pairs of two adjacent operations whose current start times violate the precedence constraint between them (see figure 10).

(2) Resolve first conflict pairs involving a bottleneck operation.

(3) Change the start time of one operation only to resolve a conflict pair.

[4)] In job shop scheduling, the notion of bottleneck usually corresponds to a particular *resource interval* demanded by operations that exceeds the resource's capacity. Most state-of-the-art scheduling techniques emphasize the capability to identify *dynamic* bottlenecks that arise during the construction of the solution [33, 38]. In our approach with agents' dynamic local interaction, we utilize both notions of *static* (i.e., fixed resource contention ratio) and dynamic bottleneck. Static bottleneck corresponds to a resource within the entire interval specified by the problem. Dynamic bottleneck is a resource interval that is highly contended by operations as a result of job agents' proposals and is identified by non-bottleneck resource agents.

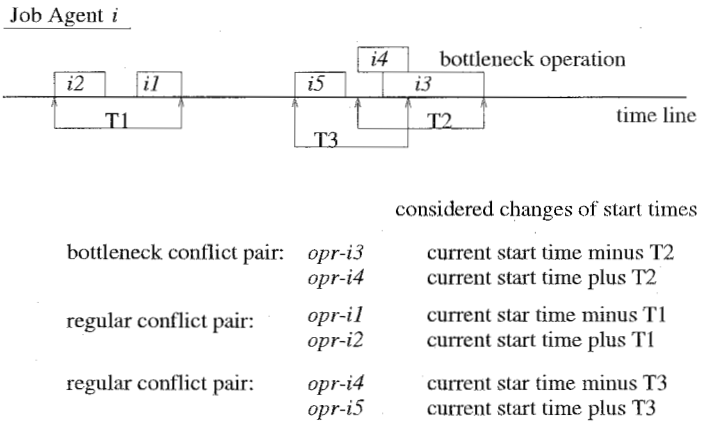

Figure 10. Conflict resolution of job agent.

(4) For a conflict pair including a bottleneck operation and a regular operation, if the change frequency counter on the regular operation is still under the threshold, change the start time of the regular operation; if not, change the start time of the bottleneck operation.

(5) For a conflict pair of regular operations, if one of the two operations can be changed within its boundary and resource slack, change that operation; if not, change the operation with lower change frequency.

(6) Change the start time of an operation such that it deviates from the previous value the least possible.

In figure 10, the conflict pair of opr_{i3} and opr_{i4} will be resolved first since opr_{i3} is a bottleneck operation. If the change frequency of opr_{i4} is still below a threshold, start time of opr_{i4} will be changed by an addition of T2 (the distance between current start time of opr_{i4} and current end time of opr_{i3}) to its current start time. Otherwise, start time of opr_{i3} will be changed by a subtraction of T2 from its current start time. In both cases, start time of opr_{i5} will be changed to the end time of opr_{i4}. To resolve the conflict pair of opr_{i1} and opr_{i2}, either start time of opr_{i1} will be changed by a subtraction of T1 from its current start time or start time of opr_{i2} will be changed by an addition of T1 to its current start time. The decision is based on the boundary, temporal slack, resource slack, and change frequency of both operations.

4.2. Negotiation heuristics of non-bottleneck resource agents

Non-bottleneck resource agents resolve capacity constraint violations using the following heuristics:

(1) Re-allocate the over-contended resource intervals to the competing operations.

(2) Keep changes to the start times of these operations to a minimum.

(3) Allocate operations in a sequence based on the weight information associated with them. If their original resource intervals have been preempted by other operations, a most adjacent resource interval within their boundaries is allocated, considering the preference of staying within their temporal slacks.

(4) If a conflicting resource interval involves more than three operations, assign high change frequency (higher than the threshold) to these operations after conflict resolution.

For example, in figure 11, opr_{a4} was preempted by opr_{e2}, which has higher weight. A most adjacent resource interval is allocated to opr_{a4}. In addition, when a resource agent detects a high resource contention during a particular time interval (such as the conflict involving opr_{c3}, opr_{d1}, and opr_{g1}), it allocates the resource intervals and assigns high change frequency (higher than the threshold) to these operations, and thus dynamically changes the priority of these instantiations.

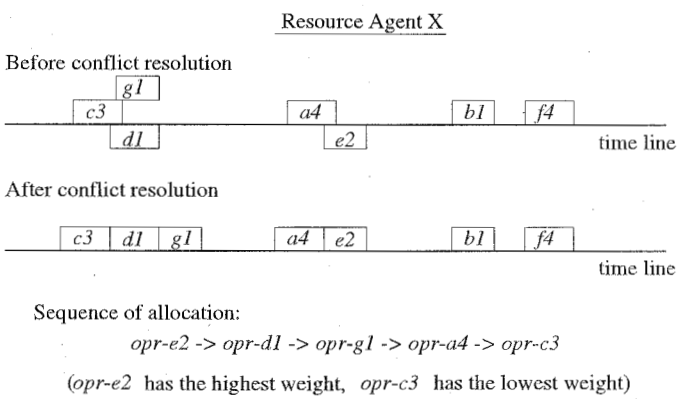

Figure 11. Conflict resolution of non-bottleneck resource agent.

4.3. Negotiation heuristics of bottleneck resource agents

A bottleneck resource agent has high resource contention. This means that most of the time, a bottleneck resource agent does not have resource slack between operations. In job shop scheduling, two operators, *exchange*(i,j) and *right-shift*(i,t), have been used [21, 22, 24, 28] to modify the processing sequence of operations on a resource. *Exchange*(i,j) switches (or swaps) the start times of a pair of operations (i,j). *Right-shift*(i,t) re-assigns the start time of operation i to a later time t. Bottleneck resource agents resolve capacity constraint violations using the following heuristics:

(1) Examine the amount of overlap of conflicting operation resource intervals.

(2) If the overlap is considered as small, right-shift operations that follow the changed operation.

(3) If the overlap is considered as not small, swap the changed operation with an appropriate operation.

In figure 12(i), opr_{e3}, opr_{b3}, and opr_{c3} are all right-shifted to resolve capacity conflicts because the conflicted interval is small and there are slacks before the latest finish time of opr_{c3}. In figure12(ii), the changed opr_{d3} is swapped with opr_{b3} since the changed start time of opr_{d3} is later than the start time of opr_{e3}. Resource intervals are

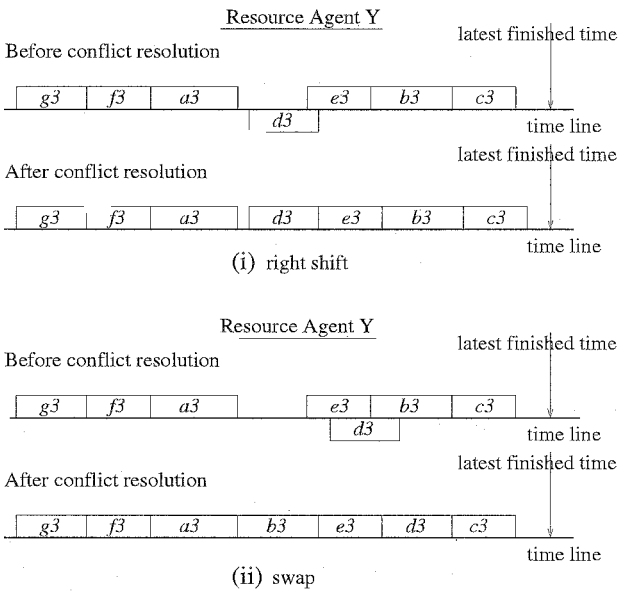

Figure 12. Conflict resolution of bottleneck resource agent.

then allocated to the new sequence – opr_{b3}, opr_{e3}, opr_{d3}. The intuition behind the heuristics is to keep the number of changed variables as minimum as possible. Note that the new start time of opr_{d3} is in the same direction (on the time line) of modification made by a job agent on opr_{d3}. This is to comply with the job agent's constraints of not being able to process the preceding operations of opr_{d3} earlier.

5. Group negotiation process

Given a job shop scheduling problem, agents are created to represent jobs and resources. The group negotiation proceeds as follows:

Step 1. Initialize agents.

(a) Job agents become active. Each job agent calculates a boundary for each operation under its jurisdiction considering its release and due date constraints.

(b) Resource agents become active.

- Each resource agent calculates the contention ratio for its resource by summing up the processing times of operations on the resource and dividing it by the interval length between the earliest and latest boundary time among the operations.

  Denote the set of operations using a resource R_k by $\{opr_{ij}^k\}$. Denote the boundary of an operation by (st_{ij}^b, et_{ij}^b). A resource's contention ratio r is calculated by

$$r = \frac{\sum p_{ij}}{(max(et_{ij}^b) - min(st_{ij}^b))},$$

  where p_{ij} is the processing time of an operation opr_{ij}. If the resource contention ratio r is larger than a certain threshold, a resource agent concludes that it is a bottleneck resource agent [5] and marks the operations under its jurisdiction as bottleneck operations.

- Each resource agent heuristically allocates the earliest free resource interval to each operation under its jurisdiction according to each operation's boundary.

All operations are assigned a start time. This initial value assignment of all variables represents the initial proposal of the tentative schedule.

Step 2. Job agents are active. Each job agent looks for constraint violations. If it does not find any, it announces that it is in a satisfaction status. Otherwise, it proposes changes to the current value assignment.

Step 3. Resource agents are active. Each job agent looks for constraint violations. If it does not find any, it announces that it is in a satisfaction status. Otherwise, it proposes changes to the current value assignment.

Step 4. If all agents are in the satisfaction status, stop the process. The current value assignment of variables represents an agreement of all agents and is a solution to the problem. Otherwise, go to step 2.

5.1. An illustrative example

We illustrate, in more detail, the negotiation process through an example. This example features detection of dynamic bottlenecks and reconfiguration of the island of

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

reliability. The input scheduling problem consists of five jobs on five machines with the same due date. During initialization, job agents calculate boundary information for each operation according to the release/due dates and the processing times of operations. In figure 13, t_1 is the earliest time opr_{00} can start; t_4 is the latest time opr_{00} should finish if job0 is to meet its due date. Therefore, (t_1, t_4) is the boundary of opr_{00}. Denote the boundary start time and boundary end time of opr_{00} by st^b_{00} and et^b_{00}, respectively. We have $st^b_{00} = t_1$ and $et^b_{00} = t_4$.

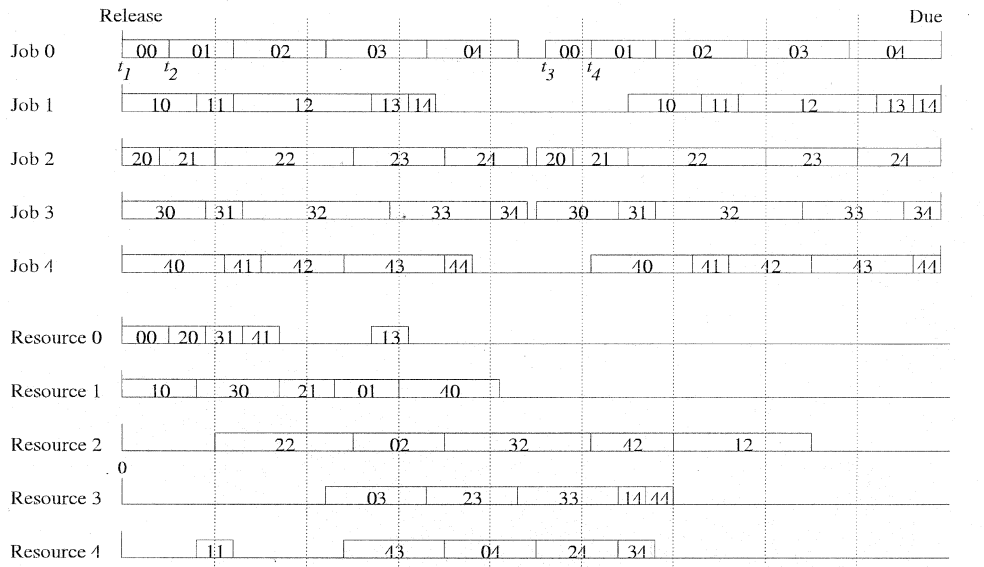

Figure 13. Initialization of job and resource agents.

With this boundary information associated with each operation, resource agents build an initial schedule with a heuristic, e.g., allocate free resource intervals to operations with earliest boundary start time at the beginning and then with earliest boundary end time if there is no idleness. For example, in resource0, both opr_{00} and opr_{20} have the earliest boundary start time. Opr_{00} is arbitrarily chosen for allocation. Then, opr_{20} is allocated next because it has the earliest boundary end time among the remaining operations. The allocation sequence is followed by opr_{31}, opr_{41}, and opr_{13}. Note that opr_{13} is assigned to its earliest start time. In resource1, after the allocation of opr_{10} which has the earliest boundary start time, the allocation sequence of opr_{30}, opr_{21}, opr_{01}, and opr_{40} is established according to earliest boundary end time. Among the five resources, R_2 identifies itself as the bottleneck resource since it has the largest resource contention ratio. All five operations using R_2 are marked as bottleneck operations.

Figure 14 shows the first cycle of the negotiation process. In job agents, operations represented by two boxes are those involved in constraint violations and whose

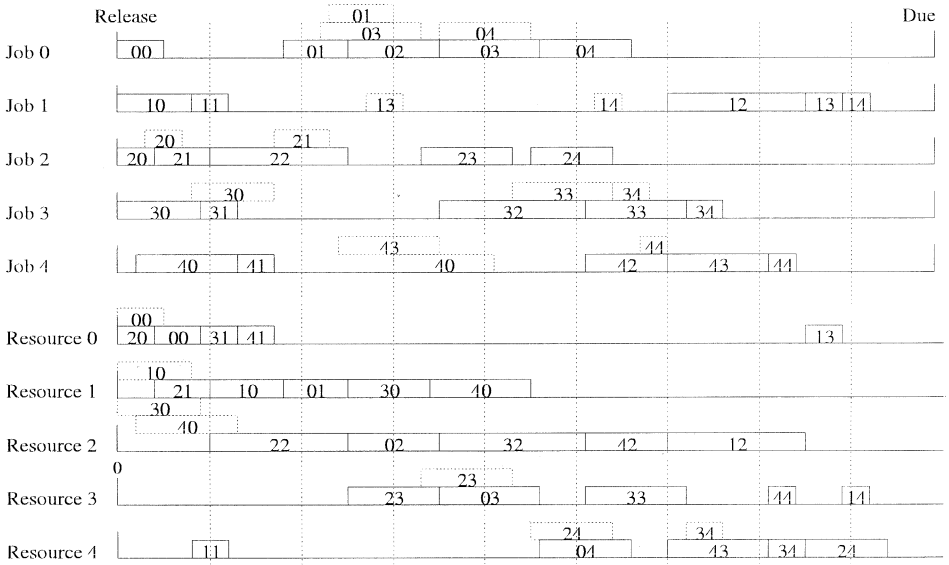

Figure 14. Cycle 1 of an example negotiation process.

start times are changed. The dotted boxes represent their previous locations assigned by resource agents. The solid boxes represent their new locations assigned by job agents after conflict resolution. J_0 detects two conflict pairs – (opr_{01}, opr_{02}) and (opr_{02}, opr_{03}). Since opr_{02} is a bottleneck operation and change frequencies of opr_{01} and opr_{03} are zero, both opr_{01} and opr_{03} are assigned to new locations (heuristics 3 and 4). Opr_{04} is also assigned to a new location in order to comply with the conflict resolution between opr_{02} and opr_{03}. In J_3, two conflict pairs (opr_{30}, opr_{31}) and (opr_{32}, opr_{33}) are detected. The conflict between opr_{32} and opr_{33} is resolved as in J_0. The conflict between opr_{30} and opr_{31} can not be resolved within their resource slacks (heuristic 5). Since change frequencies of both operations are zero, J_3 heuristically chooses opr_{30} to change its start time.

Resource agents become active after all job agents have made necessary modifications on the tentative schedule to ensure no violation of their constraints. In resource agents, the dotted boxes represent operations' previous locations assigned by job agents. The solid boxes represent operations' new locations assigned by resource agents after conflict resolution. In resolving conflicts, resource agents allocate operations to a location as close to the original location as possible (heuristics 1 and 2). In R_0, a capacity constraint violation between opr_{00} and opr_{20} is detected. Since opr_{20} has a higher weight than opr_{00}, opr_{00} is changed to a new location (heuristic 3). R_1 detects a high resource contention between opr_{10}, opr_{21}, opr_{30}, and opr_{40}. Opr_{21} and opr_{01} are allocated to their original locations because they have higher weight information than other operations (heuristic 3). Next, opr_{30} is allocated to a location just after opr_{01} because its processing time is longer than the free resource interval between opr_{21}

and opr_{01} (heuristic 2). The allocation is followed by opr_{40} and opr_{10}. Opr_{10} is allocated to the free resource interval between opr_{21} and opr_{01} since its processing time is equal or smaller than the interval (heuristic 2). After conflict resolution, the R_1 agent assigns a change frequency higher than the threshold to opr_{10}, opr_{30}, and opr_{40} in order to propagate its decisions on the dynamic bottleneck interval (heuristic 4). In R_4, the agent allocates opr_{34} and opr_{24} to a later start time, instead of an earlier start time, by considering their temporal slack information (heuristic 3).

Figure 15 shows the second cycle of the negotiation process. J_1 changes the location of opr_{11} because opr_{11} has a high change frequency (heuristic 4). J_3 changes the locations of opr_{31}, opr_{32}, and opr_{33} because opr_{30} has a change frequency higher than the threshold (heuristic 4). J_4 also assigns a new start time to opr_{41} because

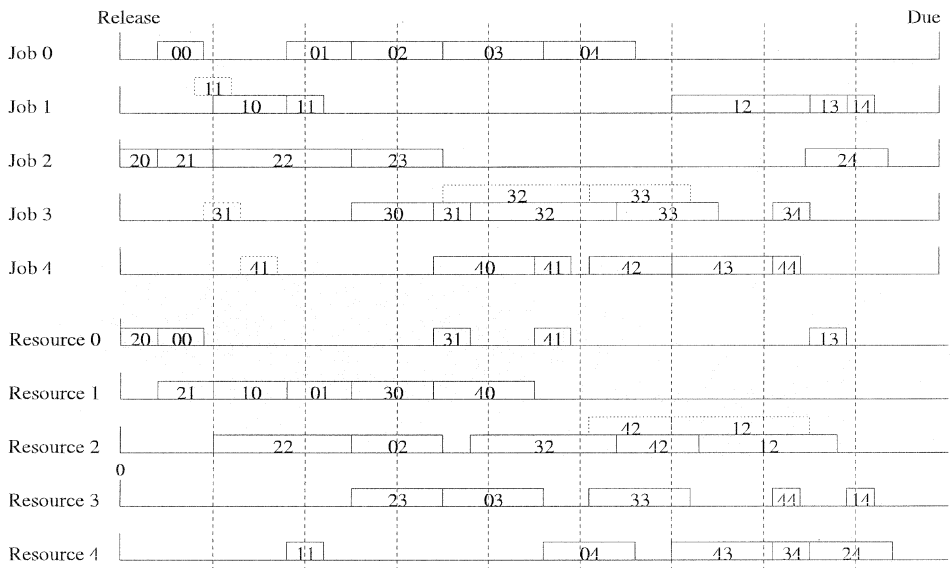

Figure 15. Cycle 2 of an example negotiation process.

opr_{40} has a high change frequency. J_0 and J_2 are satisfied with the current solution and make no modification. During resource agents' negotiation activities, only the R_2 (the bottleneck resource) agent finds constraint violation. Since the interval of capacity constraint violation is small, R_2 right-shifts opr_{42} and opr_{12} (heuristic 2). This represents a new configuration of the islands of reliability. All operations' change frequency counters are reset to zero.

Figure 16 shows the third cycle of the negotiation process. In both J_1 and J_4, opr_{13}, opr_{14} and opr_{43}, opr_{44} are assigned to a later start time because opr_{12} and opr_{42} are bottleneck operations and all these non-bottleneck operations have a change frequency of zero (heuristic 3). R_4 is the only resource agent which finds constraint

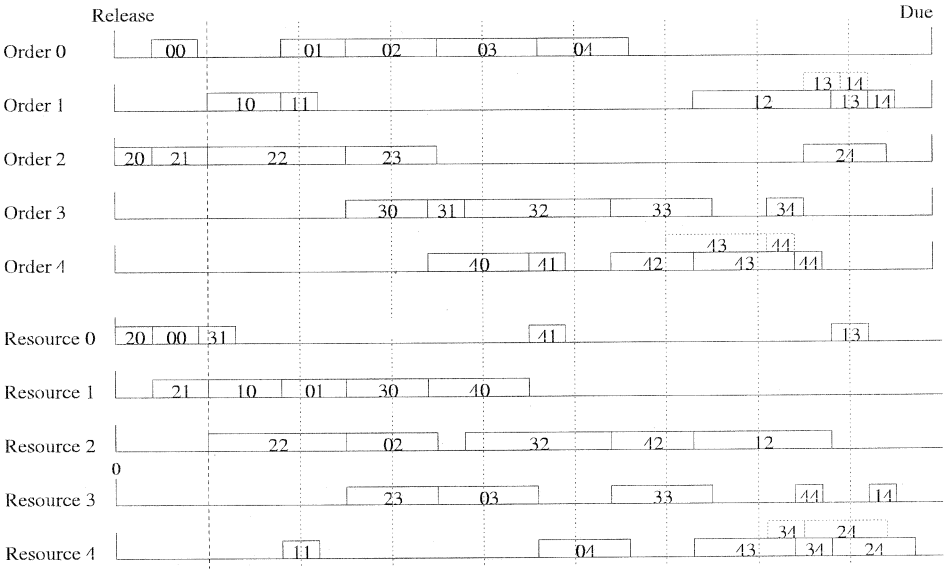

Figure 16. Cycle 3 of an example negotiation process.

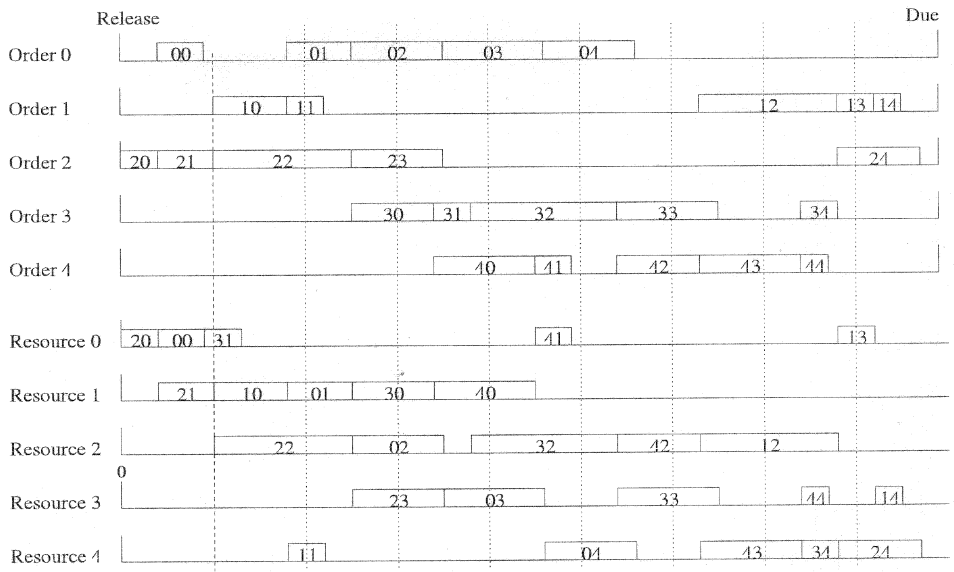

Figure 17. Cycle 4 of an example negotiation process.

violation when resource agents become active. R_4 assigns new start times to opr_{34} and opr_{24} because they have lower weight information than opr_{43}.

Figure 17 shows the fourth cycle of the negotiation process. All job and resource agents are satisfied with the current solution. Therefore, the current value assignment is an admissible schedule for the job shop scheduling constraint satisfaction problems.

6. Negotiation for optimization

Recall that in job shop scheduling, bottleneck resources have the dominant effects on both the admissibility and quality of the schedule (see figure 4). In constraint optimization problems with a tardiness objective, a job's tardiness is mostly due to its delay on the bottleneck resource. In other words, the operation processing sequence on the bottleneck resource is responsible for most of the tardiness cost of jobs. Therefore, CONA can be naturally extended to the job shop scheduling constraint optimization problems with a tardiness objective.

We developed a negotiation mechanism for group optimization, called Anchor& Ascend, where a globally favorable solution is found by iterative local negotiation of an initial proposal based on an anchor sub-solution with monotonically increased local objective cost. In Anchor&Ascend, one of the bottleneck resources is selected and assigned the role of *anchor* (*agent*) by heuristic, e.g., assign the role of anchor agent to the last bottleneck in jobs' processing sequence. The non-bottleneck resources together with the not selected bottleneck resources will be referred to as regular resources. Suppose every uth operation in each job uses the selected bottleneck resource. For each job J_i, we define pre-anchor operations to be $\{opr_{ij}\}$, $j = 1,\ldots,u-1$, the anchor operation to be opr_{iu}, and post-anchor operations to be $\{opr_{ij}\}$, $j = u+1,\ldots,n$. A valid sub-solution for the anchor agent is a sequence of the anchor operations $\{opr_{iu}\}$, $i = 1,\ldots,m$, with specified start times $\{st_{iu}\}$, in which none of the intervals $(st_{iu}, st_{iu} + p_{iu})$ overlaps with others and all $st_{iu} \geq est_{iu}$. The local tardiness objective function for the anchor agent is $C_{anchor} = \sum_{i=1}^{m}\{w_i \times max[0, (st_{iu} - lst_{iu})]\} = \sum_{i=1}^{m} c_{iu}$, where c_{iu} is the tardiness cost of opr_{iu}, m is the number of jobs, and $lst_{iu} = dd_i - \sum_{k=u}^{n} p_{ik}$. The local cost C_{anchor} is used by the anchor agent to evaluate its sub-solution and estimate the global weighted tardiness cost of the current solution. In other words, we assume all the tardiness of jobs is due to tardiness on the bottleneck resource and we use c_{iu} to predict the cost c_i of the job J_i by assuming all post-anchor operations in the job will be processed with no delay, i.e., $st_{ij} = st_{iu} + \sum_{k=u}^{j-1} p_{ik}$, $i = 1,\ldots,m$, $j = u+1,\ldots,n$. Of course, any actual delay will certainly increase c_i and increase the overall weighted tardinesscost C, i.e., $C_{anchor} \leq C$.

Anchor&Ascend expands CONA by incorporating local optimization in the selected bottleneck resource. The anchor agent conducts a local best first search for configuring its local sub-solution. Initially, the anchor agent constructs an optimal local sub-solution by heuristic local search. This anchor sub-solution is used as an island of reliability in CONA for constructing a feasible global solution. If the CONA

session is not successful and the anchor sub-solution is changed by some job agent, the anchor agent employs a set of modification operators to construct new configurations of the anchor sub-solution, and store these potential configurations into a candidate list. Then the anchor agent selects a configuration with the lowest C_{anchor} from the candidate list. A new CONA session begins with the new anchor sub-solution as the island of reliability. The process is repeated until an agreement is found.

Anchor&Ascend controls the distributed local negotiation for global optimization by assuming disparity among agents and going through a process of testing the feasibility of constructing a global solution based on different configurations of the anchor sub-solution with *monotonically increased* objective costs. It does not guarantee optimal solutions. We must emphasize that in job shop scheduling, most techniques do not provide guarantee for optimality in most objective functions. In practice, the goal is to find a high-quality solution with reasonable computational cost. In addition, Anchor&Ascend is not designed as a general mechanism for distributed optimization. It focuses on a subset of job shop scheduling problems, i.e., problems with clear bottlenecks, where it performs effectively.

The following subsections present the Anchor&Ascend distributed optimization procedure in more detail.

6.1. Initial optimization of anchor sub-solution

Initially, the anchor agent generates a processing sequence $\{opr_{iu}\}$ on the bottleneck resource according to the initial resource allocation heuristics in CONA. Being a highly contended resource, the bottleneck resource usually processes the sequence $\{opr_{iu}\}$ without any slack (gap) between adjacent operations, opr_{pu}^{k-1}, opr_{qu}^{k}, and opr_{ru}^{k+1}, where the superscripts $(k-1, k, k+1)$ denote the processing sequence on the bottleneck resource. In other words, $st_{pu} + p_{pu} = st_{qu}$, and $st_{qu} + p_{pu} = st_{ru}$. To optimize the sequence $\{opr_{iu}\}$ with minimal C_{anchor}, the anchor agent goes through an iterative process of switching pairs of anchor operations opr_{iu} to reduce C_{anchor}.[6] During this process, two heuristic subroutines[7] *jump forward* and *jump backward*, are used. In jump forward, an anchor operation opr_{iu} in the sequence is repeatedly moved forward toward the time origin by switching with one of the preceding operations to reduce C_{anchor}. In jump backward, an anchor operation opr_{iu} in the sequence is repeatedly moved backward toward time infinity by switching with one of the succeeding operations to reduce C_{anchor}. Given a sequence of operations, $S = \{opr_{iu}^{k}\}$, the subroutines are described as follows:

[6] For other objective functions (e.g., makespan), exact methods are available for this one-machine sequencing problem. However, to guarantee optimality for weighted tardiness where no exact method is available, it requires a more elaborate branch and bound procedure which is exponential in the worst case. We choose to rely on heuristics for efficiency.

[7] A similar heuristic, pairwise interchange, is also used in neighborhood search [24].

Jump Forward

Step 1. Calculate c_{iu} for each opr_{iu}. Select an operation, opr_{pu}^{υ}, in S with the largest c_{pu}.

Step 2. For each opr_{iu}^{k}, k from $\upsilon - 1$ to 1. If $st_{iu} < est_{pu}$, go to step 3. Otherwise, if switching opr_{pu} with opr_{iu}^{k} would reduce C_{anchor}, then switch them.

Step 3. Remove opr_{pu} from S. If S is empty, stop. Otherwise, go to step 1.

Jump Backward

Step 1. Calculate c_{iu} for each opr_{iu}. Select an operation, opr_{pu}^{υ}, in S with the smallest c_{pu}.

Step 2. For each opr_{iu}^{k}, k $\upsilon + 1$ to m. If $st_{iu} < est_{pu}$, go to step 3. Otherwise, if switching opr_{pu} with opr_{iu}^{k} would reduce C_{anchor}, then switch them.

Step 3. Remove opr_{pu} from S. If S is empty, stop. Otherwise, go to step 1.

To obtain a near optimal sub-solution, the anchor agent repeats the course of applying both the jump forward and jump backward subroutines to the current sequence $\{opr_{iu}\}$ until C_{anchor} can no longer be reduced.

6.2. Modification of anchor sub-solution

The changes to a subset of start times $\{st_{iu}\}$ by a subset of job agents signal to the anchor agent that a reconfiguration of $\{st_{iu}\}$ is required. The anchor agent uses *exchange*(i, j) and *right-shift*(i, t) to modify the processing sequence. In determining the scope of applying the two operators, we consider the following: (1) since the current anchor sub-solution has the minimal local objective cost, it is desirable to modify it as little as possible in order to limit the increase in the objective cost, (2) the modification should also correspond to the changes made by job agents so that new anchor sub-solutions have a better chance of leading to a successful constraint satisfaction iteration. Therefore, *right-shift*(i, t) is applied to only the changed anchor operation, while *exchange*(i, j) is applied to a heuristic neighborhood of the changed anchor operation.

The local proposal guided by application of the two modification operators generates a set of candidate anchor sub-solutions. These new sequences are put into the list of candidate sequences if they do not duplicate existing sequences in the list. The list of candidate sequences is sorted by increasing local objective cost. Then the anchor agent chooses the first one (with the least cost) from the list to be the next anchor sub-solution and the process of constraint satisfaction by the non-anchor agents repeats until a global solution is found where all constraints are satisfied. Since the anchor agent searches for a proper sequence with monotonically increased objective cost, the global solution represents the best solution that the Anchor&Ascend procedure can find.

In particular, suppose the anchor agent has a current operation processing sequence of (A,B,C,D,E,F). If B was changed by a job agent to a later start time t', the anchor agent first restores its original valid sequence by changing start times of those changed by job agents back to their original start times. Then the anchor agent applies *right-shift*(B, t') to the current processing sequence. This results in a new sequence with a gap between A and B, and all operations after B are right-shifted accordingly with B. The anchor agent also applies *exchange*(i, j) to the current processing sequence, with i = B, C, $j = i + 1$, $i + 2$, and i = D, $j = i + 1$. In other words, five exchange operations, exchange(B,C), exchange(B,D), exchange(C,D), exchange(C,E), and exchange(D,E), are individually performed on the current processing sequence, which results in five new sequences, e.g., exchange(B,C) results in (A,C,B,D,E,F), exchange(C,D) results in (A,B,D,C,E,F).

These six applications of the two modification operators (five exchanges and one right-shift) represent a heuristic balance between minimizing cost increase (solution quality) and increasing chances of leading to successful search by other agents (search efficiency). In particular, exchange(B,C), exchange(B,D), and right-shift(B, t') directly respond to the job agent's constraint violation (unable to meet constraint imposed by the start time of B) by assigning B to later start times. Exchange(C,D), exchange(C,E), and exchange(D,E) attempt to change the condition of resource contention of regular resources such that the preceding operation of B in the job might start earlier and B might start at its original start time. For example, if one of the pre-anchor operations of C was competing with the preceding operation of B for the same resource such that the preceding operation of B could not finish before B's start time, by exchanging C with D and moving C to a later start time, the resource may schedule the preceding operation of B being processed before the pre-anchor operation of C so that B will not be bumped by its preceding operation. When there is more than one anchor operation being changed by job agents, the anchor agent only performs modifications on the earliest changed anchor operation in order to limit cost increase since anchor operations in the later sequence of the anchor agent's sub-solution have later start times and may have larger tardiness cost than anchor operations in the earlier sequence.

6.3. Algorithmic procedure of distributed optimization

The anchor agent is primarily searching for a proper sequence $\{opr_{iu}\}$ with minimal C_{anchor} that would result in a successful negotiation by job and regular resource agents in CONA. Below, we give the algorithmic procedure of Anchor&Ascend. Suppose A is the set of agents and $A_c \in A$ is the selected anchor agent. $Sol(\,)$ is an instance of a (sub)solution generated by an (or a set of) agent(s). OP is a set of modification operations that are designed to generate a neighborhood of $Sol(A_c)$. S_i is a state characterized by a $Sol_i(A^c)$.

Initialization: OLD_STATES, NEW_STATES are empty lists.

Step 0. A_c generates and optimizes a sub-solution $Sol_0(A_c)$. A_c generates a state S_0 to record $Sol_0(A_c)$ and puts S_0 in NEW_STATES.

Step 1. A_c picks the first state S_i in NEW_STATES. This is the current anchor sub-solution $Sol_i(A_c)$.

Step 2. All agents in $A - \{A_c\}$ engage in a session of negotiation for distributed satisfaction by CONA with $Sol_i(A_c)$ anchored. If the session is successful, then an globally high quality solution $Sol(A)$ is found. Terminate the procedure. Otherwise, A_c puts S_i in OLD_STATES.

Step 3. A_c performs the set of modification operations OP to $Sol_i(A_c)$, which results in a set of newly configured sub-solutions $\{Sol_{i+1}(A_c),\ldots, Sol_{i+q}(A_c)\}$.

Step 4. For each new $Sol_{i+j}(A_c)$, A_c checks if there is an identical instance in either OLD_STATES or NEW_STATES. If not, A_c generates a state S_{i+j} to record $Sol_{i+j}(A_c)$. A_c calculates the local objective cost of $Sol_{i+j}(A_c)$ as the state cost of S_{i+j} and puts S_{i+j} in NEW_STATES.

Step 5. A_c sorts NEW_STATES according to increasing state cost. Go to step 1.

The following sections present experimental studies of our cooperative problem solving approach to distributed job shop scheduling. We experimentally compare the results of our work against centralized scheduling techniques. The results show that multi-agent coordination techniques can provide substantial gains in terms of problem solving efficiency and solution quality.

7. Experimental evaluation in job shop satisfaction problems

We evaluated the performance of CONA on a suite of job shop scheduling satisfaction problems proposed in [33]. The benchmark consists of 6 groups, representing different scheduling conditions, of 10 problems, each of which has 10 jobs of 5 operations 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 bottleneck conditions. Each problem in the benchmark has at least one feasible solution.

The purpose of this experimental study is to (1) investigate the effects of coordination information in the system, (2) compare CONA's performance to other constraint-based as well as priority dispatch scheduling methods, (3) investigate the effects of initial solution configuration in the system, (4) investigate CONA's scaling up characteristics on problems of larger sizes.

7.1. 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 become active, they resolve conflicts by randomly changing the instantiation of one of the two operations 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.

The set of configurations of coordination information is shown in table 1, with the additional coordination information for each configuration underlined. Table 2 reports the comparative performance of different configurations on the suite of benchmark problems in terms of the number of problems solved out of the 60 problems and the average iteration cycles to solve a problem. The number of iteration 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 a 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 here each problem was tried only once. C0 was only able to solve 8 problems with an average of 33.3 iteration cycles for solution evolution, while C3 was able to solve all 60 problems with an average of 5.8 iteration cycles. As a light increase in the average iteration cycles in C1 (compared to C0) stems from the fact that while C1 was able to solve twice the number of problems than C0 due to boundary information during initial resource interval allocation, it had no other advantages over C0 to resolve subsequent conflicts. We confirm that adding coordination information enables the system to solve more problems within fewer iteration cycles. The results show the utility of coordination information.

Table 1

A set of coordination configurations.

C0	No coordination information
C1	Boundary
C2	Boundary + Bottleneck tag
C3	Boundary + Temporal slack + Weight Bottleneck tag + Resource slack + Change frequency

Table 2

Comparative performance between coordination configurations.

Overall Performance	Coordination configuration			
	C0	C1	C2	C3
Avg. no. of prob. solved	8.0	15.8	36.3	60
Avg. no. iter. cycles	33.3	36.3	24.7	5.2

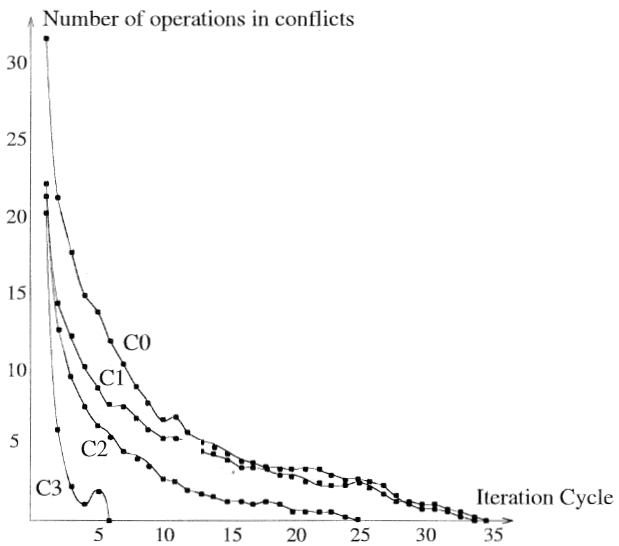

Figure 18. Convergence processes of different coordination configuration.

Figure 18 shows, for different coordination configurations, the successful overall problem solving processes[8] in terms of the number of operations involved in conflicts at each cycle. As the coordination information increases, the shape of the curve

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

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, a modification on the start times of the operations using bottleneck resources typically occurred. The curves for C0, C1, and C2 do not exhibit a peak because the system does not have a particular pattern of interaction in those coordination configurations.

7.2. Comparison with other scheduling techniques

CONA was compared to four other constraint-based heuristic search scheduling techniques, ORR/FSS, MCIR, CPS, and PCP. ORR/FSS [33] incrementally constructs a solution by a chronological backtracking search guided by specialized variable and value ordering heuristics. ORR/FSS+ is an improved version augmented with an intelligent backtracking technique [42]. Min-Conflict Iterative Repair (MCIR) [20] 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) [26] employs a search space analysis methodology based on stochastic simulation which iteratively prunes the search space by posting additional constraints. Precedence Constraint Posting (PCP) [36] conducts the search by establishing sequencing constraints between pairs of operations that share a common resource based on a *slack-based* heuristic. We also include three frequently used and appreciated priority dispatch rules – EDD, COVERT, and ATC [24] in the comparison.

Table 3 reports the performance of each technique. Each row displays the number of problems solved by each technique in each problem category, followed by the total number of problems solved and the average CPU times used. Note that the results of ORR/FSS, ORR/FSS+, CPS, and PCP were obtained from published reports of the developers of the techniques. The performance of MCIR has been shown to be significantly affected by initial solutions. MCIR used in the SPIKE system [16] with heuristic initialization can solve all 60 problems. However, with randomly generated initial solutions, MCIR can only solve about 24 problems [26]. We adopt the results reported in [26] as the main result of MCIR. All CPU times, except PCP, were obtained from Lisp implementations on a DEC 5000/200. In particular, CONA was implemented in CLOS (Common Lisp Object System).CPS, MCIR, ORR/FSS, and ORR/FSS+ were implemented using CRL (Carnegie Representation Language) as an underlying frame-based knowledge representation language. CPU times of CPS, MCIR, ORR/FSS, and ORR/FSS+ were divided by six from the published numbers as an estimate of translating to straight Common Lisp implementation.[9] Sadeh and Fox [34] report that

[9] ORR/FSS and ORR/FSS+ obtained 30 times speedup in C/C++ implementation [34]. We assumed a factor of five between Common Lisp and C/C++ implementations.

Table 3

Performance comparison.

	CONA	CPS	MCIR	ORR/ FSS	ORR/ FSS+	PCP	EDD	COVERT	ATC
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	7
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 sec.	13.07 sec.	49.74 sec.	39.12 sec.	21.46 sec.	0.3 sec.	0.9 sec.	0.9 sec.	0.9 sec.

the CPU times of a more recent version of ORR/FSS+ fell between 1.5 and 2.5 seconds. PCP was implemented in C and can solve 58 to 60 problems depending on the parameters that specify search bias. Although CONA can operate in partial parallelism, it was sequentially implemented for fair comparison. The fact that dispatch priority rules can solve up to 52 problems seems to indicate that the set of benchmark problems are not particularly difficult. Nevertheless, the results show that CONA is quite competitive as compared to the other constraint-based scheduling techniques, both in feasibility and efficiency in finding a solution.

7.3. Effects of initial solution configuration

CONA essentially employs negotiation agents to iteratively propose changes to the current solution until a valid solution evolves. Agents' local interaction direct the solution repairing process. This is different from iterative improvement (hill-climbing) methods [19, 43] that make local changes to reduce a cost function. A previous study [23] indicates that the goodness of a rough initial solution has great effects on the performance of iterative improvement methods. The performance difference of MCIR based on heuristic and random initial solutions provides evidence for this observation.

In order to investigate the effects of initial solution on the system's performance, we conducted experiments with both heuristic and random initial solutions. In the heuristic initial solution, resource agents allocate free resource intervals to operations with earliest boundary end times after the initial allocation to an operation with earliest boundary start time. In random initial solutions, the selection of operations for free resource intervals was random among the eligible operations (i.e., those operations

with earliest boundary start times before the current start time of free resource interval). CONA was ran on each problem for 10 randomly generated initial solutions.

CONA was able to solve all 60 problems with both heuristic and random initial solutions, although the number of iteration cycles required to solved a problem is slightly different. Table 4 shows CONA's performance on both heuristic and random initial solutions in terms of the number of average iteration cycles required to find a solution for a problem in each problem category. CONA has almost equal overall

Table 4

Comparative averaged iteration cycles between initial solutions.

Initialization	w/1	w/2	n/1	n/2	0/1	0/2	Overall
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

performance with heuristic and random initial solutions. The results, combined with the fact that the simple dispatch EDD priority rule can solve 47 out of 60 problems, reveal that most instances in Sadeh's problem set are not particularly difficult. There seems to exist quite a number of solutions for each problem. Therefore, even with random initial solutions, CONA can still find a solution in a few iteration cycles. However, the experimental results indicate that CONA's repairing approach, based on the interaction of local negotiation, is certainly more effective than other iterative repairing methods, such as MCIR.

7.4. Scaling up performance

In order to experimentally investigate CONA's scaling up performance, we used the same problem generator function producing the benchmark problems [10] to produce four sets of problems that involve 250 (10 jobs 25 resources), 500 (10 jobs 50 resources), 1000 (10 jobs 100 resources), and 3000 (10 jobs 300 resources) operations.[11] These problems exhibit similar scheduling conditions to the benchmark problems. Table 5 reports the average CPU times CONA spent on each problem size.

CONA implemented in Lisp can solve problems of three thousand operations within 14 minutes on a Dec 5000/200 workstation. Assuming the computation speed-up on more powerful machines and with C implementation, it is very likely that CONA

[10] The code was kindly provided by Dr. Sadeh.

[11] Although we expand the problem size along one dimension (e.g., number of resources) only, we expect that CONA will have similar scaling up performance along the other dimension (e.g., number of jobs) since the CONA mechanism does not depend on the ratio of jobs and resources. The primary factor of CONA's performance is the disparity structure of the problem (see section 7.1).

Table 5

CONA's CPU time on different problem sizes.

Problem size	CPU time
50 operations	4.8 seconds
250 operations	26.2 seconds
500 operations	60.2 seconds
1000 operations	137.6 seconds
3000 operations	814.3 seconds

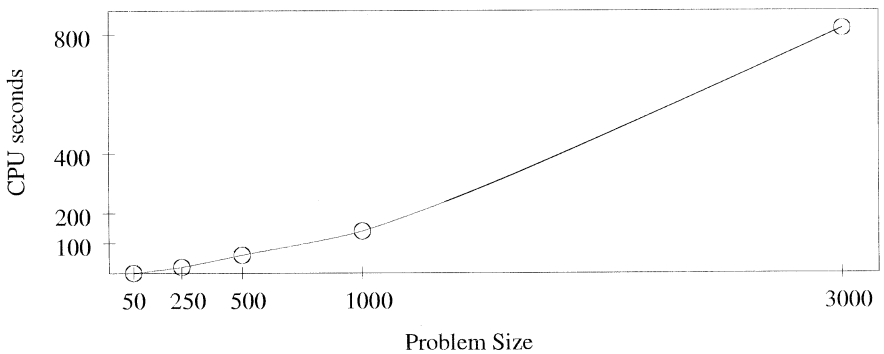

Figure 19. CONA's scaling up performance.

would be able to solve problems of several thousand operations within one minute. Figure 19 shows CONA's CPU time increase on these larger sized problems, which exhibits favorable, near-linear scaling-up performance. The results seem to suggest that CONA's search mechanism is polynomial to the size of the search space in the type of problems we tested. In fact, in scaling up evaluation based on empirical study, most systems' performance vary with the difficulty of test problems. We suspect that CONA would not do as well in problems that are more difficult (with more tight job due dates). However, it appears that CONA does have an edge to techniques based on global search in the type of problems that are moderately difficult.

8. Experimental evaluation on job shop optimization problems

The purpose of this experimental study is to (1) evaluate the performance of Anchor&Ascend using both dispatch priority rule and simulated annealing as benchmarks, (2) examine the applicability conditions of Anchor&Ascend.

8.1. *Performance comparison*

We evaluated the performance of Anchor&Ascend on a suite of job shop scheduling optimization problems similar to that proposed in [33]. The benchmark consists of 8 groups, representing different scheduling conditions, of 10 problems, each of which has 20 jobs of 5 operations and 5 machines. Each job goes through each machine exactly once with randomly generated visiting sequence, except for the bottleneck machines, which were always visited after a fixed number of operations. Each group of problems differs in three respects: (1) tardy factor; (2) due date range; (3) number of a priori bottlenecks. Tardy factor controls the average tightness of job due dates. Two levels of tardy factor are used to generate problems with loose average due date or tight average due date. Due date range is controlled by varying the amplitude of the intervals within which job due dates are generated. Two levels of due date range are introduced: wide, narrow. The benchmark also considered one and two a priori bottleneck conditions. In the one-bottleneck problems, the 4th operation of each job uses the bottleneck machine; in the two-bottleneck problems, the 2nd and 5th operations use the bottleneck machines.

Each job J_i has a tardiness weight w_i that proportionally penalizes the job's tardiness T_i. In addition, a marginal inventory cost inv_i is introduced to each job by its first operation. The objective of this experiment is to minimize the total schedule cost, which is the sum of the weighted tardiness cost and inventory cost of each job,

$$COST = \sum_{i=1}^{m} WT_i + INV_i,$$

where m is the number of jobs in the problem (see section 2.2). Table 6 categorizes the problem set by the parameters.

Table 6

Problem set categorization based on parameter settings.

Problem set	Number of bottlenecks	Tardy factor	Due date range
1	1	tight	narrow
2	1	tight	wide
3	1	loose	narrow
4	1	loose	wide
5	2	tight	narrow
6	2	tight	wide
7	2	loose	narrow
8	2	loose	wide

Anchor&Ascend essentially tries to find a feasible solution based on different configurations of processing sequence on the bottleneck resource with monotonically

increased local objective cost. For problems with two bottleneck resources used by the 2nd and the 5th operations in each job, Anchor&Ascend needs to select one of the bottleneck resources as the anchor agent. Anchor&Ascend operates on the assumption that the anchor agent can use its local objective cost to estimate the actual global objective cost. This assumption dictates that the latest bottleneck resource be selected as the anchor agent. Therefore, for problems with two bottleneck resources, Anchor& Ascend selects the one used by the 5th operation of each job as the anchor agent. Anchor&Ascend may take quite a long time to find a solution for problems with two bottleneck resources. In order to keep the experiments within a reasonable time frame and to report meaningful computational results, we limit the state space search of the anchor agent to 800 states, which approximately corresponds to 10 minutes on a SPARC IPX workstation. As a result, Anchor&Ascend can only solve parts of the problems with two bottleneck resources.

Although the objective of the problems is to minimize the total schedule cost of weighted tardiness and inventory, the anchor agent only considers weighted tardiness as its local objective cost. In order to take inventory cost into account in the optimization process of Anchor&Ascend, we incorporate additional local optimization heuristics to job agents. When a new configuration of the anchor sub-solution is constructed by the anchor agent, e.g., the application of *exchange*(i, j) and *right-shift*(i, t), and the selection of the new configuration with the lowest objective cost, job agents re-assign start times for their non-anchor operations based on a just-in-time&no-delay policy. Denote opr_{iu} as the anchor operation in a job J_i with n operations. The start times of the pre-anchor operations opr_{ij}, $j = 1,\ldots,u-1$, are re-assigned so that they will be processed just in time for the anchor operation to start as assigned by the anchor agent. In other words,

$$st_{ij} = st_{iu} - \sum_{k=j}^{u-1} p_{ij}, \quad j = 1,\ldots,u-1.$$

The start times of the post-anchor operations opr_{ij}, $j = u+1,\ldots,n$, are re-assigned so that they will be processed with no delay after the anchor operation has finished. In other words,

$$st_{ij} = st_{iu} + \sum_{k=u}^{j-1} p_{ij}, \quad j = u+1,\ldots,n.$$

Job agents' just-in-time&no-delay start time re-assignment right after a new configuration of the anchor sub-solution represents a simultaneous effort to reduce both the inventory cost and the weighted tardiness cost.

We chose to evaluate our Anchor&Ascend method against dispatch scheduling because of its conceptual simplicity and ease of implementation. Briefly, dispatch scheduling is a way of generating schedules by simulating the process of jobs being performed on the resources. A job is completed after it has visited all the resources in its routing. A production schedule is a description of when jobs visit each resource

Table 7

Experimental results on problems with one bottleneck resource.

Problem set	Instance	Anchor&Ascend				X-ATC rule		
		Tardy	Inv.	Total	CPU sec.	Tardy	Inv.	Total
	1	10780	5986	16766	9.8	14840	11322	26162
	2	1640	4288	5928	5.5	3150	7335	10485
	3	2410	4478	6888	10.2	4030	7895	11925
	4	3700	5068	8768	17.8	5580	8895	14475
1	5	1200	4665	5865	12.3	2180	6432	8612
	6	4880	5347	10227	8.9	5940	9326	15266
	7	2090	4110	6200	21.4	3010	6963	9973
	8	1040	4247	5287	16.3	2400	7355	9755
	9	5290	4311	9601	6.2	6370	7720	14090
	10	8580	5764	14344	13.5	11860	11147	23007
	1	4920	5449	10369	21.2	8230	10446	18676
	2	4020	3559	7579	7.9	3630	6743	10373
	3	180	4859	5039	6.3	150	7638	7788
	4	390	4582	4972	5.2	1840	8344	10184
2	5	1160	4625	5785	17.5	40	5608	5648
	6	630	5201	5831	9.4	740	9048	9788
	7	330	3488	3818	16.1	830	6474	7304
	8	1500	3989	5489	22.8	2240	6429	8669
	9	4450	3617	8067	10.6	5240	6977	12217
	10	7630	4450	12080	16.1	9730	10011	19741
	1	5250	6659	11909	20.0	8280	11839	20119
	2	270	5206	5476	4.4	810	8068	8878
	3	780	5668	6448	3.2	1540	8737	10277
	4	1390	5983	7373	3.4	2640	9406	12046
3	5	0	4754	4754	15.8	350	7191	7541
	6	1900	5948	7848	15.9	2440	10121	12561
	7	140	4879	5019	9.2	500	7701	8201
	8	1150	5780	6930	3.0	570	7773	8343
	9	1270	5052	6322	6.2	2770	8347	11117
	10	4590	6269	10859	12.2	7500	11813	19313
	1	1320	6059	7379	21.4	2520	11123	13643
	2	1870	4704	6574	20.0	1520	7811	9331
	3	0	5642	5642	3.4	10	9129	9139
	4	300	5976	6276	3.8	110	9464	9574
4	5	690	5686	6376	16.4	0	6776	6776
	6	0	6314	6314	4.2	0	10314	10314
	7	0	4099	4099	6.2	140	7661	7801
	8	860	4523	5383	5.0	910	7384	8294
	9	1770	3644	5414	18.4	1970	7414	9384
	10	3980	4874	8854	33.7	5720	10396	16116

before leaving the shop. The quality of the schedule is controlled by the priority rule with which resources select the next job to process in the queue. Since no priority rule is good for optimizing conflicting objectives, we focused on weighted tardiness cost to select a priority rule. We used the X-ATC rule to obtain the benchmark performance. The X-ATC rule has been shown to be one of the best priority dispatch rules for weighted tardiness problems [24] because of its successful priority index function and its capability to strategically insert idleness for important jobs, which is very helpful in reducing weighted tardiness. The index function of an operation opr_{ij} is defined by

$$\left(1.0 - \frac{B[rdt_{ij} - t]^+}{p_m}\right) \frac{w_i}{p_{ij}} \exp\left(-\left[\frac{s_{ij}(t) - \sum_{q=j+1}^{m_i} wt_{iq}}{k\,p}\right]^+\right),$$

where rdt_{ij} is the ready time, p_m is the minimum processing time in the queue, w_i is the weight of the job, p_{ij} is the processing time, s_{ij} is the slack time, wt_{iq} is the waiting time, and p is the average processing time in the queue. The X-ATC rule requires parameter settings of the look-ahead parameter k and the penalty parameter B. We ran the X-ATC rule with k in the range of (1.0−3.0) with a granularity of 0.25 and with B in the range of (0.4−2.0) with a granularity of 0.2, e.g., k has 9 different values, 1.0, 1.25,…, 3.0, and B also has 9 different values, 0.4, 0.6,…, 2.0. In other words, each problem was run 81 times with different combinations of parameter values. For each problem, the best result (the lowest schedule cost) was recorded.

Tables 7 and 8 show the results of Anchor&Ascend and the X-ATC rule on the problems. For each problem, we report the weighted tardiness cost, the inventory cost, and the total schedule cost of the generated schedule, and the computational cost in terms of CPU seconds for Anchor&Ascend to solve a problem. Anchor&Ascend solved all 40 one-bottleneck problems, but solved only 18 out of 40 two-bottleneck problems within 800 search states of the anchor agent. For all problems solved by Anchor&Ascend, it always finds schedules with substantially less inventory cost than that of the X-ATC rule. Except for problem number 5 in the second subset of the problems, the schedules found by Anchor&Ascend have considerably lower total schedule cost than that of the X-ATC rule. In the one-bottleneck problems, Anchor&Ascend solved 33 out of 40 problems with less weighted tardiness cost. In the two-bottleneck resource problems, Anchor&Ascend solved 11 out of 18 problems with less weighted tardiness cost.

We examined Anchor&Ascend's performance on the 14 problems in which it produced higher weighted tardiness cost than the X-ATC rule. We found that in half of the problems, Anchor&Ascend started from an initial configuration of the anchor sub-solution with higher weighted tardiness cost than that of the schedule generated by the X-ATC rule. For example, in problem number 5 of the second subset, the weighted tardiness cost of the initial anchor sub-solution in Anchor&Ascend is 780, while the X-ATC rule generated a schedule with weighted tardiness cost of 40. Similarly, in

Table 8

Experimental results on problems with two bottleneck resources.

Problem category	Instance	Anchor&Ascend				X-ATC rule		
		Tardy	Inv.	Total	CPU sec.	Tardy	Inv.	Total
5	1	16370	5227	21597	678.3	21430	10408	31838
	2	8720	4802	13522	65.8	9680	10061	19741
6	1	6460	4663	11123	196.2	8740	9818	18558
	2	11230	3742	14972	195.4	10760	7219	17979
	3	17390	3749	21139	173.7	14010	7969	21979
	4	5950	4261	10211	385.1	7670	9529	17199
7	1	7400	5950	13350	54.8	10500	11067	21567
	2	1800	5428	7228	73.1	3230	8623	11853
	3	1620	4939	6559	189.7	2840	9215	12055
	4	2860	4916	7776	667.8	2690	8896	11586
	5	2720	5778	8498	96.0	3050	10653	13703
8	1	2410	5185	7595	20.9	1980	10598	12578
	2	3430	3521	6951	147.4	4540	7155	1169
	3	760	4609	5369	91.8	820	8639	9459
	4	12040	4091	16131	20.3	8490	8080	16570
	5	7010	4634	11644	4.2	6000	8292	14292
	6	1290	3675	4965	396.1	1840	9050	10890
	7	3960	4894	8854	3.2	2690	10020	12710

problem number 8 of the third subset, Anchor&Ascend started from a weighted tardiness cost of 1150, while X-ATC produced a weighted tardiness cost of 570; in problem number 5 of the fourth subset, Anchor&Ascend started from a weighted tardiness cost of 670, while X-ATC produced a weighted tardiness cost of 0. This indicates that our heuristic optimization procedure (iterative jump forward/backward) on the bottleneck resource is inadequate to find a very good sub-solution in some cases. Since iterative jump forward/backward does not consider inserting idleness, it may miss some better configuration of operation processing that requires strategic idleness for more important jobs. Another factor is the deviation of the estimated weighted tardiness cost from the anchor agent and the actual weighted tardiness cost of the schedule. Since the anchor agent calculates the estimated weighted tardiness cost by assuming all downstream operations will be processed immediately, any actual delay will cause an increase in the final weighted tardiness cost. In the one-bottleneck problems where the 4th operation of each job uses the bottleneck resource, the final weighted tardiness cost may be somewhat higher than the estimated weighted tardiness cost of the anchor agent.

Table 9 shows the average results in each problem category. Table 10 reports Anchor&Ascend's improvement performance over the X-ATC rule. We first examine

Table 9

Average results in each problem category.

Problem category	Anchor&Ascend					X-ATC rule		
	Tardy avg.	Inv. avg.	Total avg.	CPU sec. avg.	CPU sec. dev.	Tardy avg.	Inv. avg.	Total avg.
1	4161.0	4826.4	8987.4	12.2	1.5	5936.0	8439.0	14375.0
2	2521.0	4381.7	6902.7	13.3	1.9	3267.0	7771.8	11038.8
3	1674.0	5619.8	7293.8	9.3	1.9	2740.0	9099.6	11839.6
4	1079.0	5152.1	6231.1	13.2	3.1	1290.0	8747.2	10037.2
5	12545.0	5014.5	17559.5	372.0	216.6	15555.0	10234.5	25789.5
6	10257.5	4104.0	14361.5	237.6	42.8	10295.0	8633.8	18928.8
7	3280.0	5402.2	8682.2	199.0	103.4	4462.0	9690.8	14152.8
8	4414.3	4372.7	8787.0	97.7	49.7	3765.7	8833.4	12599.1

Table 10

Performance improvement of Anchor&Ascend over the X-ATC rule in each problem category.

Problem category	Tardy improvement (%)	Inv. improvement (%)	Total improvement (%)
1	29.9	42.8	37.5
2	22.8	43.6	37.5
3	38.9	38.2	38.4
4	16.4	41.1	37.9
5	19.4	51.0	31.9
6	0.4	52.5	24.1
7	26.5	44.3	38.7
8	– 17.2	50.5	30.3
All (avg.)	17.2	45.5	34.5

the results in weighted tardiness cost. Anchor&Ascend performed considerably better in the one-bottleneck problems (categories 1, 2, 3, 4) than in the two-bottleneck problems (categories 5, 6, 7, 8). The results were as expected because the coordination strategy of anchoring on one bottleneck resource would not be as effective when there are two bottleneck resources. For either one- or two-bottleneck problems, Anchor&Ascend performed better in problems with narrow due date range (categories 1, 3, 5, 7) than in problems with wide due date range (categories 2, 4, 6, 8). We examined the difference of the initial optimal weighted tardiness cost and the final weighted tardiness cost of each category of the one-bottleneck problems compared to

that of the X-ATC rule. We found that for one-bottleneck problems with a narrow due date range, the difference was only 0.5%, while the difference was 17% for problems with a wide due date range. In other words, using weighted tardiness cost produced by the X-ATC rule as benchmark, Anchor&Ascend found schedules with weighted tardiness cost very close to the initial optimal weighted tardiness cost in problems with a narrow due date range. But the weighted tardiness cost of the final solution for problems with a wide due date range were moderately increased from the initial optimal weighted tardiness cost. We conclude that Anchor&Ascend finds less optimal solutions in problems with more variation of job due dates. For the parameter of the tardy factor, we did not find significant overall performance difference in problems with tight due dates (categories 1, 2, 5, 6) or loose due dates (categories 3, 4, 7, 8).

Anchor&Ascend substantially reduces the inventory cost on all problems. This is due to job agents' just-in-time&no-delay policy of re-assigning operation start times whenever the bottleneck resource agent configures a new bottleneck operation processing sequence. Therefore, the final schedule is a result of the balanced optimization by both job agents and the bottleneck resource agent with respect to inventory cost and weighted tardiness cost, respectively. Overall, Anchor&Ascend finds schedules with reduced weighted tardiness cost and substantially less inventory cost in most problems.

As an additional evaluation, we also compare, on a smaller scale, Anchor&Ascend against the Focused Simulated Annealing Search (FSAS) proposed in [35], whose study was conducted on the same set of job shop scheduling optimization problems. Table 11 reports the comparison on a representative problem instance in each problem category.[12)] Table 12 shows the improvement performance of Anchor&Ascend over FSAS. Note that FSAS is a stochastic procedure. The reported results were the best results over 10 FSAS runs of each problem. Overall, Anchor&Ascend always found solutions with reduced weighted tardiness cost, slightly increased inventory cost, and better total cost than the best results of FSAS.

As to computational cost, Anchor&Ascend required only 12 seconds on average for a one-bottleneck problem, but consumed almost 4 minutes on average for a two-bottleneck problem. Both CPU times were based on Lisp implementation on a SPARC IPX workstation. A single run of the FSAS procedure in C implementation required about 5 to 8 minutes on a DECstation 5000/200. Although there are substantial efficiency differences in problems with a different number of bottlenecks, Anchor&Ascend is still more efficient than the FSAS procedure. In fact, Anchor&Ascend is extremely efficient in one-bottleneck problems. The results show the significant achievement of Anchor&Ascend with its capability to optimize conflicting objectives simultaneously and effectively find a very good solution in the job shop scheduling optimization problems.

[12)] Unfortunately, we have FSAS results only on the first problem instance in each problem category, which was kindly provided to us by Mr. Nakakuki.

Table 11

Comparison between Anchor&Ascend and Focused Simulated Annealing Search (FSAS).

Problem category	Anchor&Ascend			FSAS		
	Tardy	Inv.	Total	Tardy	Inv.	Total
1	10780	5986	16766	12620	5899	18519
2	4920	5449	10369	7660	5200	12860
3	5250	6659	11909	7260	6342	13602
4	1320	6059	7379	2020	5517	7537
5	16370	5227	21597	18040	5189	23229
6	6460	4653	11113	8300	4741	13041
7	7400	5950	13350	8090	6419	14509
8	2410	5185	7597	3360	4978	8338

Table 12

Performance improvement of Anchor&Ascend over Focused Simulated Annealing Search (FSAS).

Problem category	Tardy improvement (%)	Inv. improvement (%)	Total improvement (%)
1	14.6	– 1.5	9.5
2	35.8	– 4.8	19.3
3	27.7	– 5.0	12.4
4	34.7	– 9.8	2.1
5	9.3	– 0.7	7.0
6	22.2	1.9	14.8
7	8.5	7.3	8.0
8	28.3	– 4.2	8.9
All (avg.)	22.6	– 2.1	10.2

8.2. Applicability conditions

The fundamental coordination strategy of Anchor&Ascend is to exploit disparity among agents. Job shop scheduling problems with clear bottleneck resources naturally involve disparity between bottleneck resources and non-bottleneck resources. The previous experimental results show Anchor&Ascend's performance on one-bottleneck and two-bottleneck problems. However, in order to get a complete picture of the relation of Anchor&Ascend's performance and disparity conditions of the problems, we need to quantify disparity conditions of job shop scheduling

We denote a bottleneck resource as BR_i, a non-bottleneck resource as NBR_i. The average processing time p^i_{av} of a resource (BR_i or NBR_i) is the average processing time

of operations requiring the use of the resource (BR_i or NBR_i). In addition, num_b is the number of bottleneck resources, and num_{nb} is the number of non-bottleneck resources in the shop. p^b_{AV} is the average p^i_{av} of bottleneck resources, i.e., $\sum_{i=1}^{num_b} p^i_{av}/num_b$. p^{nb}_{AV} is the average p^i_{av} of non-bottleneck resources, i.e., $\sum_{i=1}^{num_{nb}} p^i_{av}/num_{nb}$. We define two disparity characteristics in job shop scheduling as follows:

Disparity Ratio is the ratio of the average p^i_{av} of bottleneck resources to the average p^i_{av} of non-bottleneck resources, i.e., $p^b_{AV}/\ p^{nb}_{AV}$.

Disparity Composition Ratio is the ratio of the number of bottleneck resources to the number of resources in the shop, i.e., $num_b/(num_b + num_{nb})$.

These two parameters quantify disparity conditions of job shop scheduling problems and specify the job shop conditions that our distributed optimization heuristics (described in section 3.1) is designed for. Anchor&Ascend is intended for shops with at least one bottleneck resource, e.g., (Disparity Composition Ratio > 0) $\wedge$ (Disparity Ratio > 1).

To study the performance of the Anchor&Ascend procedure under different disparity conditions, we constructed a set of test problems[13] with a disparity ratio ranging from 1.25 to 5.0 with a granularity of 0.25, and a disparity composition ratio ranging from 0.2 to 0.8 with a granularity of 0.2. Each combination of the two parameters is represented by a subset of 10 problems that are randomly generated while controlling the disparity condition. Therefore, the problem set includes 64 subsets and a total of 640 problems. Each problem consists of 20 jobs on 5 resources with 100 operations to be scheduled. The disparity ratio represents the ratio of average processing times between the subgroup of bottleneck resources and the subgroup of non-bottleneck resources. A disparity composition ratio of 0.4 means that 2 out of 5 resources are bottleneck resources. Table 13 specifies the sequence (in bold) of using bottleneck resources in each job for each disparity composition ratio. For example, in the disparity composition ratio of 0.4, the second operation uses the first bottleneck resource, the fourth operation uses the second bottleneck resource, in each job. In cases of more than one bottleneck resource, the Anchor&Ascend mechanism requires the selection of a particular bottleneck resource as the anchor agent. In this study, the latest bottleneck resource is selected to be the anchor agent.

Experimental results are evaluated by both computational cost and solution quality. We use the total number of search states[14] the anchor agent explored before a global solution was found as an estimate of the computational cost. CPU times of

[13] This is similar to the set of problems we used to evaluate Anchor&Ascend in the previous experiments, except we manipulated the number of bottleneck resources and the average processing times of each resource. The tardy factor and due date range were fixed at a tight and narrow level, respectively. The problem generator code was kindly provided by Dr. Sadeh.

[14] Each search state represents a configuration of the anchor agent's sub-solution.

Table 13

Sequences of operations using bottleneck resources.

Disparity composition ratio	Sequences in each job
0.2	1 2 3 **4** 5
0.4	1 **2** 3 **4** 5
0.6	**1** 2 **3** 4 **5**
0.8	**1** **2** 3 **4** **5**

(1 100 300 500) states explored approximately correspond to (0.522 105 420) seconds in Common Lisp implementation on an HP-715/100 workstation. For problems of high disparity composition ratio and low disparity ratio, the anchor agent usually needs to explore a large number of search states before a solution can be found. In order to keep the experimentation within a reasonable time frame, we set a limit of 500 search states, upon which the procedure would terminate even if a global solution has not been found. Solution quality is measured by weighted tardiness cost only. Anchor&Ascend's solution quality (S_A) is evaluated by comparing it to that of the naive First Come First Served (FCFS)[15] dispatch rule (S_F) and indicated by an improvement measure, $(S_F - S_A)/S_F$. As a more serious evaluation, the solution quality of Anchor&Ascend is also compared to that of the X-ATC heuristic dispatch rule [24]. For each subset of problems representing a combination of a disparity ratio and a disparity composition ratio, an average computational cost and an average solution quality are obtained from results of 10 problems in the subset. For problems that were not solved within 500 search states, a computational cost of 500 search states is included in calculating the average computational cost, while not contributing to the average solution quality.

Figure 20 shows the computational cost over a range of combinations of disparity ratio and disparity composition ratio. Generally, computational cost increases with increasing disparity composition ratio and/or decreasing disparity ratio. Anchor&Ascend is especially efficient for problems with a disparity composition ratio of 0.2 (1 bottleneck resource out of 5 resources) and a disparity ratio of 2.0 and above, finding a solution of high quality (see figures 22 and 23) within 1 second and mostly in 1 state. The results on the computational cost also suggest that Anchor&Ascend is less applicable to problems with a disparity composition ratio higher than 0.4. Figure 21 shows the number of problems solved within 500 states in each subset. The number generally decreases with an increase in the disparity composition ratio and a decrease in the disparity ratio. The percentage of problems of disparity composition ratio (0.2 0.4 0.6 0.8) that were solved within 500 states are 97%, 66%, 34%, and 48%, respectively.

[15] Operation with the earliest ready time is dispatched first.

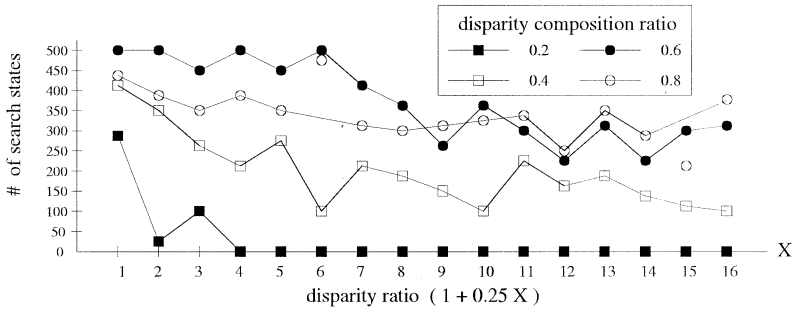

Figure 20. Computational cost in different disparity conditions.

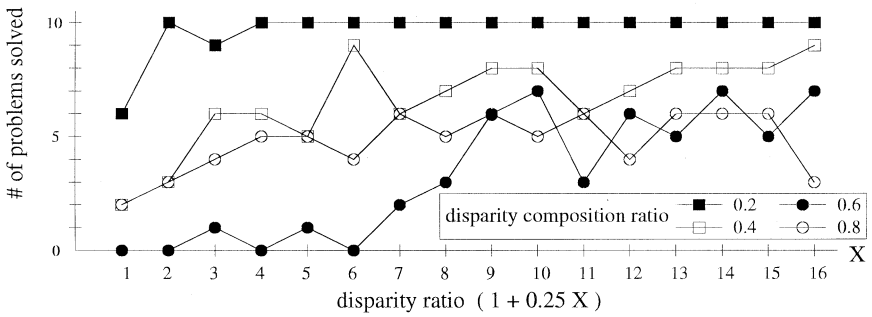

Figure 21. Number of problems solved within 500 search states
in different disparity conditions.

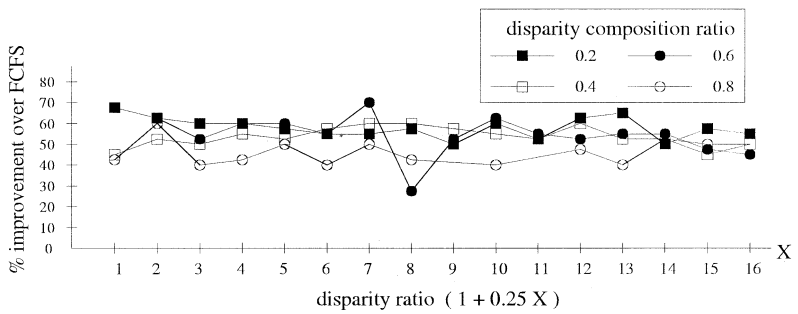

Figure 22. Solution quality of Anchor&Ascend compared to FCFS
in different disparity conditions.

Figure 22 depicts the solution quality obtained by Anchor&Ascend compared to that of FCFS. Most solutions obtained byAnchor&Ascend have an improvement of 40 to 70% over that of FCFS. Figure 23 depicts the solution quality obtained by Anchor&Ascend compared to that of X-ATC. In most problems, Anchor&Ascend has an improvement of 5 to 20% over X-ATC. In general, lower disparity composition

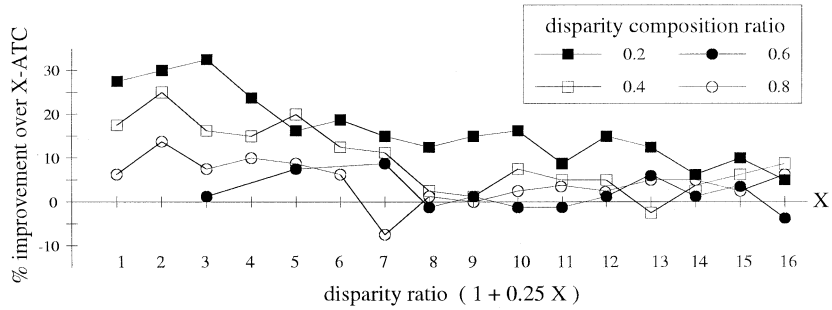

Figure 23. Solution quality of Anchor&Ascend compared to X-ATC in different disparity conditions.

seems to facilitate higher solution quality. However, no obvious relation between solution quality and disparity ratio is observed. The solution quality of problems with a disparity composition ratio of 0.6 and 0.8 has more variations and is less conclusive since the number is obtained from an average of fewer solved problems (since more problems need more than 500 search states to solve).

In general, our experiments show that Anchor&Ascend is most efficient in a high disparity ratio and a low disparity composition ratio, in which solutions of very good quality can be easily obtained with little computational cost. No obvious relation is observed between the solution quality and the disparity ratio, although an increase in the disparity composition ratio generally results in a decrease in solution quality. In addition, the disparity composition ratio is more important than the disparity ratio in the applicability of Anchor&Ascend. The experimental results seem to indicate a cut-off value of 0.4 (≤ 0.4) as the disparity composition ratio in the applicability of Anchor&Ascend.

9. Discussion

Our cooperative problem solving approach to both job shop scheduling satisfaction and optimization problems revolves around disparity among subproblems and the subsequent dominance effects on solution construction and global optimality. We expected the approach to be most effective in problems that exhibit extreme disparity among subproblems, e.g., low disparity composition ratio and/or high disparity ratio. For a given problem under the condition of a low disparity composition ratio, there are few agents with dominance effects on the solution construction. For a low disparity composition ratio, the amount of interaction required to construct a feasible global solution is much less extensive. This also implies that the disparity ratio (relative task difficulty between agent categories) has only a secondary effect on problem solving efficiency as compared to the disparity composition ratio.

In the CONA study, the set of job shop scheduling satisfaction problems has an overall disparity ratio of 1.71, with a standard deviation of 0.01 and a disparity

composition ratio of either 0.2 (one bottleneck) or 0.4 (two bottlenecks). In the Anchor&Ascend study, the set of job shop scheduling optimization problems we used to evaluate Anchor&Ascend in comparison with X-ATC and Focused Simulated Annealing Search has a disparity ratio of 1.83 in the one-bottleneck problems and a disparity ratio of 1.73 in the two-bottleneck problems. Both disparity ratios have a standard deviation of 0.01. Our cooperative problem solving approach solved these two benchmark sets with considerable efficiency in comparison with other techniques. The applicability study (see figure 20) reveals that these two sets of problems are approximately at the middle point in the range of computational cost of our approach in one-bottleneck and two-bottleneck problems.

Overall, the applicability conditions results together with the performance evaluation results confirm our intuitive analysis, e.g., the approach is most effective in problems that exhibit extreme disparity among subproblems. However, the scope of applicability of Anchor&Ascend exceeds our original expectation. We were surprised that the approach is able to solve even partial sets of problems with a high disparity composition ratio, e.g., 0.6 and 0.8, with moderate computational cost. In addition, the disparity ratio can be as low as 1.25 (in the range of 1.25 to 5.0) for the approach to work, which is not an ideal problem condition for an approach that is designed based on disparity among subproblems. Overall, the experimental results attest to the considerable advantage of exploiting disparity among subproblems in designing coordination mechanisms for distributed job shop scheduling problems.

Some readers might wonder why different sets of problems and techniques are used for satisfaction and optimization evaluations. This involves the fundamental difference of the two types of problems. In a satisfaction problem, an assignment of values to variables that satisfies all constraints is a solution. All solutions are of equal quality. Typically, we need only one solution and any one will do, while in an optimization problem, solutions are measured by a set of given objectives. We are looking for the best solution we can find. Therefore, techniques that are designed for solving satisfaction problems are not readily applicable to optimization problems. Similarly, it is not meaningful to evaluate an optimization technique on satisfaction problems. In addition, we adopted problem sets that are publicly available as benchmarks for each type of problems and that happened to be different. In a broader sense, solution quality of satisfaction problems is implicitly embedded in (or controlled by) the specification of problems (e.g., due date tightness). Therefore, techniques that solve more satisfaction problems can be viewed as producing higher solution quality. As shown in the experimental results (table 3), dispatch rules solved fewer problems than most of the advanced AI techniques. For techniques that solve an equal number of problems, we can only say they produce equally good solutions. It is not meaningful to impose some scheduling measures to the solutions of satisfaction problems because they were not considered during the problem solving process of the techniques.

An additional, perhaps even more interesting, finding in our experiments is related to the bimodal characteristics of most NP-complete problems. Cheeseman [4]

conducted a study on constraint satisfaction problems and showed that there is at least an “order parameter” that separates problems into regions of solvability. It was conjectured that constraint optimization problems might exhibit similar phase transition. Essentially, the Anchor&Ascend approach iteratively attempts to solve a series of constraint satisfaction problems until it succeeds. In our experimental results, we noticed that problems tend to diverge into subsets that are either easy or hard to solve. The tendency of divergence is more evident at a high disparity composition ratio and at a low disparity ratio, where Anchor&Ascend is less effective. This observation provides empirical evidence to the phase transition property of NP-complete optimization problems.

Our approach and the experimental results also enrich the practice of job shop scheduling where consideration of shop conditions has been focused on the number of bottlenecks, resource utilization rate, and job tardy factor. We provide two additional measures of shop conditions that emphasize the relative loadings on shop resources. Our experimental results show that these two parameters (disparity composition ratio and disparity ratio) can allow us to identify shop conditions when CONA and Anchor&Ascend are most likely to be applicable and effective. Furthermore, in many real job shops where shop conditions can be adjusted ahead of time, our experimental results could guide decisions on varying the mix of job batches or changing resource loadings, such that a high quality solution can be found efficiently.

Finally, the effects of exploiting disparity among subproblems are two-sided. On the one hand, the approach is substantially efficient in problems with extreme disparity. On the other hand, the applicability is limited. For example, it might not be as efficient in problems with no clear bottleneck resources, or in problems where more than two resources are bottlenecks. The presence of clear bottlenecks seems to provide the efficiency of our cooperative problem solving approach. We characterized disparity conditions and experimentally identified the applicability scope of the approach.

10. Related work

Manufacturing environments have long been considered as requiring distributed approaches due to their complexity and the inherent distribution of activities. There have been quite a number of works in this area with different foci. We review some of the better known examples of distributed production control systems.

Parunak [32] presented a contract net [7] approach to factory control. A prototype system, YAMS, was developed to apportion tasks by the bidding and awarding mechanism. YAMS models a factory as a hierarchy of work cells. Each work cell corresponds to a node in a contract net and is a negotiating entity that can communicate both vertically and laterally. The system deals with real-time task allocation and control.

Smith and Hynynen [37] presented a system consisting of cooperative opportunistic schedulers that operate according to a hierarchical factory model and communi-

cate via message passing. The approach to scheduling decentralization was based on the multiple levels of description provided by a factory hierarchy. Three dimensions are considered for scheduling distribution – portion of manufacturing process considered, level of precision of the maintained schedule, and time horizon. Vertical and lateral communication between nodes are considered as constraint posting and status updating. Overall, the work suggested a modeling framework for decentralized factory scheduling.

Burke and Prosser [3] presented a distributed asynchronous scheduling system (DAS) that has four components – (1) a structural representation of resources, operations, and plans, (2) an active representation of the schedule, (3) scheduling agents, and (4) a mechanism for coordinating scheduling effort. A hierarchy of units in three levels, corresponding to individual resources, groups of similar resources, and the whole factory, is envisioned. Agents representing individual resources are responsible for allocating start times to operations. The middle-level agents are responsible for the delegation of work to and retraction of work from individual resources. The top-level agent performs the release of work into the factory and conflict resolution by inter-agent backtracking and constraint relaxation. Scheduling effort is coordinated implicitly by prioritizing of messages among agents.

Hadavi et al. [13] presented the Requirement Driven Scheduling (ReDS) architecture that performs dynamic, real-time factory scheduling. The system consists of four primary components – preprocessing, feasibility analysis (FA), detailed scheduling (DS), and sequencing. The preprocessor determines the time boundary of a new order given all the needed resources. A qualitative analysis is performed in the FA to see if constraint relaxation, such as adding shifts, extending due dates, are needed for the new order. With the input from FA, DS is responsible for generating a tentative schedule in the granularity of a day. Sequencing of orders within a day is conducted by dispatching. The system also includes a simulation module that allows “what-if” type questioning and verification of the generated schedule, and a statistician module that collects data for adjustment of the system and management decisions.

Sycara et al. [41] presented an approach to distributed job shop scheduling based on a distributed constraint heuristic search. Jobs are partitioned and delegated to agents in an ad hoc way. Each resource is monitored by some agent who is responsible for the registering and granting of requests for resource reservation. Agents perform asynchronous heuristic search on their subproblems based on a variable and value ordering heuristic that relies on a probabilistic resource demand profile. A distributed asynchronous back jumping algorithm was developed to extend the distributed search. A communication protocol via message passing that coordinates agents’ problem solving was also presented. Experiments were conducted to determine the feasibility of the approach and to test the parameters that influence system performance.

To the best of our knowledge, our study represents one of the first attempts to rigorously evaluate a cooperative problem solving approach based on comparison to centralized approaches. We argue that such a comparison is inevitable in order to have

a complete picture of the trade-offs between centralized and distributed approaches. We show that our cooperative problem solving approach, applied to problems with clear bottlenecks, produces equivalent or superior performance to competing centralized scheduling techniques and, therefore, provides technical support to the increasing need of distributed production management.

11. Conclusion

We have presented a distributed, cooperative problem solving approach to job shop scheduling problems. The approach involves lateral negotiation among agents with different types of constraints. By exploiting the special problem structure that involves disparity among agents, we designed coordination strategies based on anchoring on the most constrained agent. We also developed a set of coordination information and agents' negotiation heuristics to facilitate the effective operations of the cooperative problem solving process. The utility of the approach was verified by rigorous experimental evaluation against competing scheduling techniques. Our experimental results show that our approach outperforms or gives comparable performance with other state-of-the-art scheduling techniques on a benchmark suite of problems. The power of the approach results from our judicious multi-agent coordination scheme that minimizes inter-agent communication by having agents communicate through multiple, simple distributed memories, each of which is associated and shared by a limited set of agents, and exploits problem characteristics (e.g., disparity among subproblems) that help focus coordination.

We believe that this study has at least three implications for the field of computational models in management science. First, it represents one of the first cooperative problem solving approaches to combinatorial group decision problems on jointly governed decision variables with separate and different constraints. Since most businesses employ some sort of functional division, such situations abound. For example, in determining the specification of a new product, engineering people are concerned with technical soundness, manufacturing people focus on its ease of fabrication, while marketing people care about the cost of the product and the speed of delivery. Our study provides highly efficient conflict resolution between different constraints in problems of combinatorial complexity.

The second implication comes from our agent communication scheme that employs efficient, small and distributed shared memories, each of which is associated with and shared by a limited number of agents. This allows effective communication among many decentralized entities in tightly coupled problems without substantial computational overhead or communication bottlenecks. We believe that this communication scheme can facilitate more efficiency in many distributed systems.

The third implication is related to the notion of exploiting special problem structure. Recent research in distributed artificial intelligence has shown that there is no general coordination approach that works well in all situations. To achieve the operational goals of distributed cooperative problem solving, e.g., computational

efficiency, solution quality, etc., appropriate coordination approaches must be designed according to problem characteristics. We used problem structure as the basis of designing agent coordination. Defining problem structure and identifying salient features on which to base good coordination techniques is a non-trivial problem, especially for highly nonlinear, combinatorial problems, such as job shop scheduling. We examined one of the most recurrent structures involving disparity among subproblems. Our experimental results show that the approach is very useful in constructing effective coordination mechanisms for NP-hard problems.

We are currently investigating the power of the approach on problems with different structure and across a variety of optimization objectives.

References

[1] B. Ackland, A. Dickinson, R. Ensor, J. Gabbe and P. Kollaritch, CADRE – a system of cooperative VLSI design experts, *Proceedings of the IEEE International Conference on Computer Design*, 1985.

[2] A.H. Bond, The cooperation of experts in engineering design, in: *Distributed Artificial Intelligence*, Vol. 2, L. Gasser and M.N. Huhns, eds., Morgan Kaufmann, 1989, pp. 462–486.

[3] P. Burke and P. Prosser, A distributed asynchronous system for predictive and reactive scheduling, Technical Report AISL-42, Department of Computer Science, University of Strathclyde, 1989.

[4] P. Cheeseman, B. Kanefsky and W. Taylor, Where the really hard problems are, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991.

[5] S.E. Conry, K. Kuwabara, V.R. Lesser and R.A. Meyer, Multistage negotiation for distributed constraint satisfaction, IEEE Transactions on System, Man, and Cybernetics 21(1991).

[6] R.W. Conway, W.L. Maxwell and L.W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

[7] R. Davis and R.G. Smith, Negotiation as a metaphor for distributed problem solving, Artificial Intelligence (1983)63–109.

[8] E.H. Durfee, *Coordination of Distributed Problem Solvers*, Kluwer Academic, 1988.

[9] 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(1980)213–253.

[10] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*, Wiley, 1982.

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

[12] L. Gasser and M.N. Huhns (eds.), *Distributed Artificial Intelligence*, Vol. 2, Morgan Kaufmann, Los Altos, CA, 1989.

[13] K. Hadavi, M.S. Shahraray and K. Voigt, ReDS – a dynamic planning, scheduling, and control system for manufacturing, Journal of Manufacturing Systems 9(1990)332–344.

[14] M.N. Huhns (ed.), *Distributed Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1987.

[15] T. Ishida, CoCo: A multi-agent system for concurrent and cooperative operation tasks, *Proceedings of the 9th International Workshop on Distributed Artificial Intelligence*, 1989.

[16] M.D. Johnson and S. Minton, Analyzing a heuristic strategy for constraint satisfaction and scheduling, in: *Intelligent Scheduling*, M. Fox and M. Zweben, eds., Morgan Kaufmann, 1994, pp. 257–289.

[17] P.T. Kidd, *Agile Manufacturing: Forging New Frontiers*, Addison-Wesley, 1994.

[18] T.W. Malone, Modeling coordination in organizations and markets, Management Science 33(1987) 1317–1332.

[19] S. Minton, M. Johnston, A. Philips and P. Laird, Solving large-scale contraint satisfaction and scheduling problems using a heuristic repair method, *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990.

[20] S. Minton, M. Johnston, A. Philips and P. Laird, Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence 58(1992)161–205.
[21] K. Miyashita and K. Sycara, Case-based incremental schedule revision, in: *Knowledge-Based Scheduling*, M. Fox and M. Zweben, eds., Morgan Kaufmann, 1993.
[22] K. Miyashita and K. Sycara, Cabins: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair, Artificial Intelligence 76(1995)377–426.
[23] P. Morris, On the density of solutions in equilibrium points for the queens problem, *Proceedings of AAAI-92*, 1992, pp. 428–433.
[24] T.E. Morton and D.W. Pentico, *Heuristic Schdeuling System: With Applications to Production Systems and Project Management*, Wiley, New York, 1993.
[25] U. Mukhopadhyay, L. M. Stephens, M. N. Huhns and R.D. Bonnel, An intelligent system for document retrieval in distributed office environments, Journal of the American Society for Information Science 37(1986)123–135.
[26] N. Muscettola, HSTS: Integrated planning and scheduling, in: *Knowledge-Based Scheduling*, M. Fox and M. Zweben, eds., Morgan Kaufmann, 1993.
[27] B.A. Nadel, Constraint satisfaction algorithms, Computational Intelligence 5(1989)188–224.
[28] Y. Nakakuki and N. Sadeh, Increasing the efficiency of simulated annealing search by learning to recognize (un)promising runs, *Proceedings of AAAI-94*, 1994, pp. 1316–1322.
[29] S. Nirenburg and V. Lesser, Providing intelligent assistance in distributed office environments, in: *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1988, pp. 590–598.
[30] G. O'Hare, Designing intelligent manufacturing system: A distributed artificial intelligence approach, Computers in Industry 15(1990)17–26.
[31] J.Y.C. Pan and J.M. Tenenbaum, Toward an intelligent agent framework for enterprize integration, *Proceedings of the 9th National Conference on Artificial Intelligence*, 1991, pp. 206–212.
[32] H.V.D. Parunak, Manufacturing experience with the contract net, in: *Distributed Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1987, pp. 285–310.
[33] N. Sadeh, Look-ahead techniques for micro-opportunistic job shop scheduling, Technical Report CMU-CS-91-102, School of Computer Science, Carnegie Mellon University, 1991.
[34] N. Sadeh and M.S. Fox, Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem, Artificial Intelligence (1996).
[35] N. Sadeh and Y. Nakakuki, Focused simulated annealing search: An application to job shop scheduling, Annals of Operations Research 63(1994)77–103.
[36] S.F. Smith and C.C. Cheng, Slack-based heuristics for constraint satisfaction scheduling, *Proceedings of AAAI-93*, 1993, pp. 139–144.
[37] S.F. Smith and J.E. Hynynen, Integrated decentralization of production management: An approach for factory scheduling, *Proceedings 1987 Symposium on Integrated and Intelligent Manufacturing*, ASME Annual Winter Conference, 1987.
[38] S.F. Smith, P.S. Ow, C. Lepape, B. Mclaren and N. Muscettola, Integrating multiple scheduling perspectives to generate detailed production plans, *Proceedings 1986 ASME Conference on AI in Manufacturing*, 1986, pp. 123–137.
[39] T. Sugawara, A cooperative LAN diagnostic and observation expert system, *Proceedings of the IEEE Phoenix International Conference on Computer and Communication*, 1990.
[40] K. Sycara, Resolving goal conflict via negotiation, *Proceedings of AAAI-88*, 1988, pp. 245–250.
[41] K. Sycara, S. Roth, N. Sadeh and M. Fox, Distributed constraint heuristic search, IEEE Transactions on System, Man, Cybernetics 21(1991)1446–1461.
[42] Y. Xiong, N. Sadeh and K. Sycara, Intelligent backtracking techniques for job shop scheduling, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 1992, pp. 14–23.
[43] M. Zweben, A framework for iterative improvement search algorithms suited for constraint satisfaction problems, *Proceedings of the AAAI-90 Workshop on Constraint-Directed Reasoning*, 1990.