Multiagent Coordination in Tightly Coupled Task Scheduling

Jyi-Shane Liu

Department of Computer Science
National Cheng Chi University
Taipei, TAIWAN
E-mail: jsliu@cs.nccu.edu.tw

Katia P. Sycara

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
E-mail: katia@cs.cmu.edu

Abstract

We consider an environment where agents' tasks are tightly coupled and require real-time scheduling and execution. In order to complete their tasks, agents need to coordinate their actions both constantly and extensively. We present an approach that consists of a standard operating procedure and a look-ahead coordination. The standard operating procedure regulates task coupling and minimizes communication. The look-ahead coordination increases agents' global visibility and provides indicative information for decision adjustment. The goal of our approach is to prune decision myopia while maintaining system responsiveness in real-time, dynamic environments. Experimental results in job shop scheduling problems show that (1) the look-ahead coordination significantly enhances the performance of the standard operating procedure in solution quality, (2) the approach is capable of producing solutions of very high quality in a real-time environment.

Introduction

Most research on multiagent systems has considered loosely coupled agents (Huhns 1987) (Bond & Gasser 1988) (Gasser & Huhns 1989) that coordinate their actions for mutual benefit. In most of these environments, agent interaction occurs only when one agent has data, facts, views, and solutions that are of interest to other agents (Durfee & Lesser 1991), or when agents need to resolve their conflicts (Sycara 1988) (Conry, Meyer, & Lesser 1988), etc. In other words, coordination activity, although essential, does not constitute a substantial part of an agent's effort to achieve its goal. In this paper, we consider an environment where agents' tasks are tightly coupled in the sense that (1) there are only enabling relationships among subtasks and each task usually consists of more than two subtasks, thus creating cascading effects; (2) subtasks are distributed among agents and enabling relationships among agents are of multi-directions, e.g., for $task_1$, $A \rightarrow B \rightarrow C \rightarrow D$; for $task_2$, $B \rightarrow D \rightarrow C \rightarrow A$, etc., where A, B, C, D are agents, and $\rightarrow$ represents an enabling relationship, thus creating complex cause-effect relationships among agents; (3) the objective function is related to task completion time only and can not be broken down into "quality" function of subtasks, in other words, agents have no local utility function to guide their decisions. Therefore, agents need to coordinate their actions constantly and extensively in order to both complete their tasks and improve system performance. The multiagent system also needs to operate in real time that involves both scheduling and task execution. The characteristics of the environment require substantial coordination among agents, but exclude time-consuming, elaborate coordination activities.

We present an approach that consists of two parts, e.g., a standard operating procedure and a look-ahead coordination. The standard operating procedure, adopted from a generic work-flow model, is predefined according to agents' relationships. It regulates task coupling, minimizes communication, and ensures smooth real-time task execution without violating technological constraints of a task. We developed a look-ahead coordination that operates on top of the standard operating procedure and enhances its performance by increasing agents' visibility. The approach has three features. First, it is prearranged. Agents abide by an operating procedure and adopt predetermined cues/hints for adjusting their actions. This allows agents to disentangle their task coupling and coordinate their actions in real time. Second, it is self-contained. Agents consult information from others to decide their actions. Information is exchanged by message sending. Agents do not perform query. Third, it is responsive. Agents have a "perceive-and-act" type of coordination behavior. This enables the integration of task scheduling and execution in real-time multiagent systems.

The task model we consider can be formulated as distributed constraint optimization (DCOP). A constraint satisfaction problem (CSP) (Mackworth 1987) involves a set of *variables* $X = \{x_1, x_2, \ldots, x_m\}$, each having a corresponding set of *domain values* $V = \{v_1, v_2, \ldots, v_m\}$, and a set of *constraints* $C = \{c_1, c_2, \ldots, c_n\}$ specifying which values of the variables are compatible with each other. A solution to a CSP is an assignment of values (an instantiation) to all variables, such that all constraints are satisfied. Recent work in DAI has considered the *distributed* CSPs (DCSPs) (Huhns & Bridgeland 1991) (Sycara et al. 1991) (Yokoo et al. 1992) (Liu & Sycara 1995a) in which variables of a CSP are distributed among agents. Each agent has a subset of variables and coordinates with other agents in instantiating its variables so that a global solution can be found. DCOP is an extension of DCSP in which a subset of the constraints are

relaxed to achieve optimization of a given objective function (Liu & Sycara 1995b).

In our task model, each subtask is a variable that needs to be instantiated with an execution start time. Variables are distributed among a set of agents to be instantiated in real time. The problem constraints include precedence relations between subtasks and agents' processing capacity. An objective function measures the quality of task schedule produced by the agents. Since the problem is solved in real time, the goal is not to find the optimal solution but a solution as best as possible. (Yokoo et al. 1992) describes work on distributed constraint satisfaction problems (DCSPs). The work focused on complete algorithms for solving DCSPs and was not concerned with solution optimization and time restriction. (Decker & Lesser 1995) presented a family of coordination algorithms for distributed real-time schedulers. They considered a task environment where task interrelationships can be explicitly and quantitatively represented as functions that describe the effect of agents' decisions on performance. In our task model, such a function is impossible to either define beforehand or estimate on-line with any precision.

Our work can also be viewed as addressing the problem of distributed agenda ordering, e.g., at any given time, an agent might have multiple tasks waiting to be processed; how does the agent coordinate with other agents to decide its local agenda, when its decision affects other agents and, ultimately, the performance of the group of agents? This is one of the most commonly encountered problem in DAI research and has been widely studied in many application domains, such as Distributed Vehicle Monitoring Testbed (DVMT) (Lesser & Corkill 1983). The abstract solution, perhaps a direct result from human experiences, has been using sophisticated local control coupled with exchange of meta-level information (as in the work of Partial Global Planning (PGP) (Durfee & Lesser 1991)). In our approach, agents are coordinated by local prioritizing strategies and non-local look-ahead information. The unique contributions of our work are in presenting a specific coordination solution to a tightly coupled task model and in providing a clear description of local decision making and meta-level information that is applicable in a significant class of scheduling problems.

In this paper, we present initial experimental results to test the utility of the approach and investigate its performance factors. The study was conducted in the domain of real-time job shop schedule optimization. Experimental results show that the approach is capable of producing solutions of very high quality in a real-time environment. The performance factors include (1) accuracy of agents' forecasts, (2) complexity of agents' interaction, and (3) availability of indicative information.

Job Shop Schedule Optimization

A job shop is a manufacturing production environment where a set of m jobs (or tasks) $J = \{J_1, ..., J_m\}$ have to be performed on a set of n machines (or resources) $R = \{R_1, ..., R_n\}$. Each job J_i is composed of a set of sequential operations (or subtasks) opr_{ij}, $i = 1, ..., m$, $j = 1, ..., m(i)$, $m(i) \leq n$, where i is the index of the job, and j is the index of the step in the overall job. Each operation opr_{ij} has a deterministic *processing time* p_{ij} and has been pre-assigned a unique resource that may process the operation. Jobs can have very different numbers of operations and sequences of using resources. The job shop scheduling problem involves synchronization of the completion of m jobs J on n resources (machines) R and is one of the most difficult NP-complete combinatorial optimization problems (French 1982). 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 J_i can begin only after the release date rd_i 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.

Given a job shop scheduling problem, the umber 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, and 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 time. Organizations are usually interested in optimizing a schedule according to objective functions that reflect the economic goals. In his paper, we consider a commonly used objective function, called weighted tardiness, where each job J_i is given a due date dd_i and a weight w_i that represents the importance of the job. Weighted tardiness (WT) of a schedule is defined by $WT = \sum_{i=1}^{m} w_i \times max [0, (C_i - dd_i)]$, where w_i is the weight of individual job J_i and C_i is the completion time of J_i. The goal is to produce a schedule with minimized weighted tardiness.

On-line job shop scheduling is a typical multiagent task in a tightly coupled, real-time environment. We *assign each resource to an agent* that is responsible for making decision and monitoring usage of the resource. Agents are tightly coupled with each other because of the precedence constraints between operations and the fact that they can process only one operation at a time.

A Standard Operating Procedure - Dispatch Scheduling

Since a job consists of a set of operations that has to be performed in sequential order by different agents, it is

convenient to follow a work-flow model where a job enters the shop, visits different agents to have its corresponding operations performed, and then leaves the shop. A job's *routing* is the sequential set of agents that the job visits before its completion. The *arrival time* of a job at an agent is the time at which the job leaves the previous agent in its routing, and is equivalent to the *ready time* of the operation to be performed by the agent.

Dispatch scheduling is a way of generating schedules by either simulating or actualizing the process of jobs being performed by the agents. Each agent has a buffer where arriving jobs (or equivalently, the operations ready to be processed) can wait until they are processed. Jobs are released to the buffers of the first agents in their routings after their release dates. After a job is being processed by an agent, it travels to the buffer of the next agent in the routing of the job. At any point in time, an agent is in one of four states: (1) the agent is executing an operation, (2) the agent has just finished executing an operation, and there are operations ready for execution, (3) the agent is not executing an operation, and there are operations that have just become ready for execution, (4) the agent is not executing an operation, and there is no operation ready for execution. In both states (2) and (3), an agent selects an operation from its buffer to execute.

For implementation, it is convenient to view that each operation has been pre-allocated to the buffer of its designated agents. The first operation of a job is only eligible to be selected for processing after the release date of the job. An operation that is not the first operation of a job is eligible to be selected only after its immediate preceding operation has finished its processing. We give an algorithmic description of dispatch scheduling as follows:

```
T = – 1 ;
For each agent A_i ;
   O_i^u = the set of unprocessed operations ;
   O_i^e = the set of eligible operations ;
while (∃O_i^u ≠ ∅ ) do
   T = T + 1 ;
   For each agent A_i ;
   if A_i is not executing an operation
      O_i^e = updated from O_i^u ;
      if (O_i^e ≠ ∅)
            opr = selected operation from O_i^e;
            set start time of opr to T ;
            remove opr from O_i^u and O_i^e;
      fi;
   fi;
od.
```

Dispatch scheduling is simple, robust, and has been used for years as a standard operating procedure in human organizations and production/service facilities. Mostly, an agent selects an operation based on a priority rule that assigns priority indices to operations waiting to be processed. For due-date-based objectives (e.g., weighted tardiness), priority rules calculate priority index of an operation using due date of the job in various ways, e.g., earliest due date, minimum slack time, etc.

From the point of view of multiagent systems, dispatch scheduling is a robust coordination mechanism at the *procedural* level. It ensures technological constraints are satisfied, e.g., each task is completed properly by agents' sequential execution of its constituted operations. Agents communicate by reading/writing information associated with operations, e.g., operation status, job due dates, etc. The system can operate in dynamic, real-time environments. However, system performance in terms of solution quality suffers from agents' myopic decisions based on only local and current conditions (characteristics of operations currently competing for execution). Our research hypothesis was that agent coordination that broadens agents' views of problem solving conditions can obtain higher quality solutions without sacrificing computational efficiency.

A Look-ahead Coordination - Coordinated Forecasts

We developed a look-ahead coordination mechanism, called coordinated forecasts (COFCAST), that operates on top of dispatch scheduling to improve its performance. COFCAST increases agents' visibility by incorporating useful indicative information (cues) based on global and future conditions. At each decision point, agents make a decision as well as survey local situations by predicting their future decisions. These forecasts are coordinated among agents and predefined indicative information is extracted. Agents then utilize the indicative information that embeds downstream and global conditions to make better decisions.

In tardiness related objectives, the subject of coordination is the operation sequencing of agents so as to reduce the tardiness cost of the final schedule. We observe that a job's tardiness cost depends on the end time of its last operation only. In other words, no matter whether a job's upstream operations are processed earlier or just in time for the last operation to end at the same time, they would have the same tardiness cost. While a job is being processed by an agent, other jobs waiting to be processed by the same agent are delayed because of agents' unit capacity. Therefore, a good schedule is a schedule in which jobs are processed only when necessary to ensure the prompt completion of their last operations. This means that if we can reduce unnecessary earliness of upstream operations, the resulting schedule will have reduced tardiness.

Based on this observation, we developed the innovative notion of *relaxed urgency* (RU), where jobs' due dates

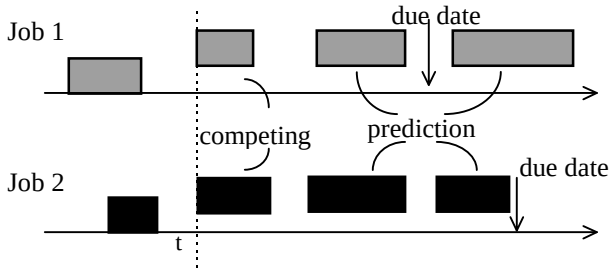

Figure 1: Example of Relaxed Urgency

used in priority rules are replaced by *relaxed due dates* that are dynamically adjusted to take downstream and global processing conditions into account. In particular, if a job is predicted to be tardy in the downstream processing, then the first unprocessed operation in the job is not regarded as urgent as it was to meet the job's original due date. Its urgency is relaxed accordingly by the tardiness of the downstream operations. For example, in Figure 1, at time *t*, the second operations of J_1 and J_2 are competing for the same resource. Based on the prediction of both jobs' downstream processing, J_2 should have a higher priority to use the resource than J_1. Given the complex interaction among agents' operation sequencing in a general job shop, the approach hinges on the ability to coordinate different forecasts from agents and extract useful information for current decisions.

In COFCAST-RU, agents forecast their future processing by assigning *predicted start times* and *predicted end times* to a partial set of the unprocessed operations. To coordinate agents' forecasts, we assign a *predicted ready time* to each operation, which is dynamically adjusted during the scheduling process. Initially, an operation's predicted ready time is set to its earliest start time $est_{ij} = rd_i + \sum_{k=1}^{j-1} p_{ik}$, where p_{ik} is the processing time of an operation opr_{ik}. We consider two actions of forecast coordination. First, an agent forecasts future processing only on operations that are "in view", i.e., its predicted ready time is less than or equal to the end time of the selected operation. This reduces the likelihood of making incorrect forecast by excluding operations that are not ready for processing in near future. Second, operations' predicted ready times are dynamically adjusted according to the predicted start times of their upstream operations. Specifically, if the predicted end time of an operation opr_{ij} is later than the predicted start time of $opr_{i(j+1)}$, then the predicted ready time of $opr_{i(j+1)}$ is set to the predicted end time of opr_{ij}. This adjustment of predicted ready times accounts for agents' processing interaction and increases forecast credibility.

An agent's forecast is done after an operation has been selected for processing. According to the priority rule, an agent sequences the set of unprocessed operations that are in view and assigns predicted start times and predicted end times to the set of operations. Then, jobs' relaxed due dates are adjusted by the agent according to the prediction on the set of operations. For a job J_i with the first unprocessed operation opr_{ij}, the *relaxed due date* dd_i^r is set by, $dd_i^r = max[dd_i, max_{q=j+1}^{m(i)} (pet_{iq} + \sum_{r=q+1}^{m(i)} p_{ir})]$, where pet_{iq} is the predicted end time of an operation opr_{iq}. In other words, relaxed due date dd_i^r of a job J_i is adjusted by its downstream operation with the greatest predicted tardiness. If none of the downstream operations is predicted to be tardy, dd_i^r is set by its original job due date dd_i. Note that the notion of relaxed urgency is realized by dynamically adjusting jobs' relaxed due dates according to downstream processing forecasts.

In summary, at each point of scheduling an operation, an agent performs four actions: (1) select an operation based on a priority rule using relaxed due dates, and assign its start time and end time, (2) based on a priority rule, assign predicted start times and predicted end times of a partial set of unprocessed operations that are in view, (3) adjust jobs' relaxed due dates based on the prediction, and (4) coordinate future forecasts by adjusting operations' predicted ready times. We describe the algorithmic procedure as follows, where prt_{ij} is the predicted ready time of an operation, pst_{ij} is the predicted start time, pet_{ij} is the predicted end time, and est_{ij} is the earliest start time.

(Initialization)
  For $i = 1, \ldots, m, j = 1, \ldots, m_i$;
    $prt_{ij} = est_{ij}$;
`while` (an agent A_k becomes idle at time t);
(Schedule an operation)
  O_k^e = the set of operations eligible for scheduling ;
  opr_{ij} = selected operation from O_k^e, using relaxed due dates ;
  $st_{ij} = t$;
  $et_{ij} = t + p_{ij}$;
(Forecast future processing)
  O_k^u = the set of unprocessed operations ;
  O_k^v = the set of operations in view (updated from O_k^u);
    ($\forall\ opr_{pq} \in O_k^v$, $prt_{pq} \le et_{ij}$)
  S_k^v = sequence of O_k^v by priority rule;
  assign pst_{pq} and pet_{pq} for opr_{pq} in S_k^v according to the sequence beginning at et_{ij};
(Update relaxed due date)
  J_s = the set of jobs of operations in S_k^v;
  For each job J_p in J_s;
    $dd_p^r = max\ [\ dd_p, max_q^{m(p)} (pet_{pq} + \sum_{g=q+1}^{m(p)} p_{pg})\ ]$;
(Coordinate future forecast)
  For each job J_p in J_s ;
  opr_{pq} = the first operation remaining to be processed ;
  For $g = q + 1$ to $m(p)$;
    `if` (pst_{pg} has not been set);

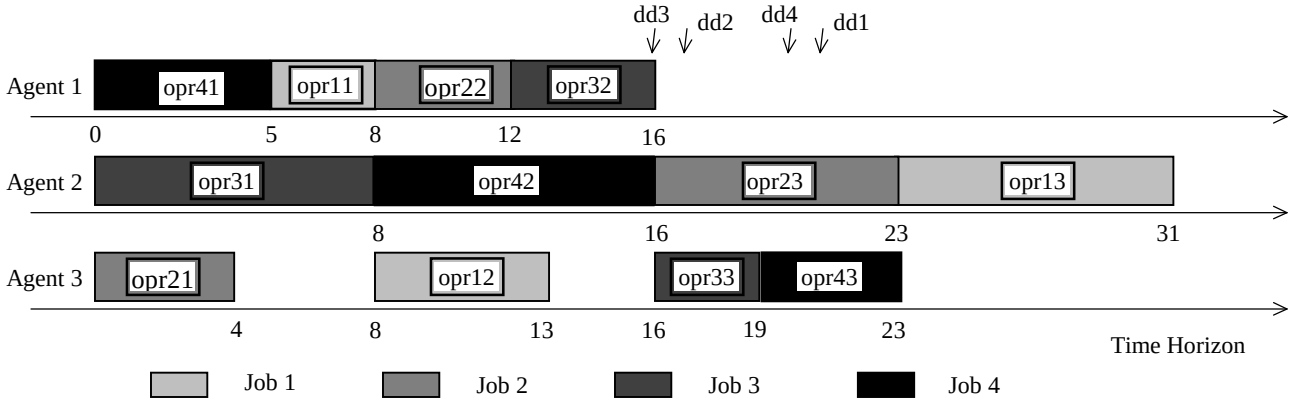

Figure 2: Example of dispatch scheduling

continue on next job in J_s ;
fi;
if ($pst_{pg} < pet_{p(g-1)}$);
$pst_{pg} = pet_{p(g-1)}$;
fi;
od.

An Example

We briefly illustrate the effect of coordinated forecasts with a simple example schedule shown in Figure 2. The problem has four jobs that need to be performed by three agents. The schedule was generated by dispatch scheduling with a simple due-date-based heuristic, e.g., minimum slack time $s_{ij} = dd_i - \sum_{k=j}^{m} p_{ij} - t$. Due dates of J_1 (dd_1), J_2 (dd_2), J_3 (dd_3), and J_4 (dd_4), are 21, 17, 16, and 20, respectively. At t = 0, A_1 selects opr_{41} because it has less slack time (s_{41} = 20 – (5 + 8 + 4) – 0 = 3) than opr_{11} (s_{11} = 21 – (3 + 5 + 8) – 0 = 5). Both A_2 and A_3 schedule opr_{31} and opr_{21}, respectively, because they are the only ready operations. At t = 5, both opr_{11} and opr_{22} are ready for A_1. Since s_{11} = 21 – (3 + 5 + 8) – 5 = 0 and s_{22} = 17 – (4 + 7) – 5 = 1, A_2 selects opr_{11}. The process continues until all operations are performed by the agents. The total tardiness cost of the schedule is (31 – 21) + (23 – 17) + (19 – 16) + (23 – 20) = 22, from J_1, J_2, J_3, and J_4, respectively.

Figure 3 shows a schedule generated by the COFCAST-RU enhanced dispatch scheduling. We focus on the forecast of A_2 since it changes the schedule. After opr_{31} is scheduled, opr_{42}, opr_{23}, and opr_{13} are all in A_2 's view, e.g., $prt_{42} = est_{42}$ = 5, $prt_{23} = est_{23}$ = 8, $prt_{13} = est_{13}$ = 8, ≤ et_{31} = 8. A_2 predicts its future processing sequence as (opr_{42}, opr_{23}, opr_{13}) based on minimum slack time. Therefore, pst_{42} = 8, pet_{42} = 16, pst_{23} =16, pet_{23} = 23, pst_{13} = 23, and pet_{13} = 31. With this forecast, A_2 updates relaxed due dates of J_4, J_2, and J_1, e.g., $dd_4{}^r = max$ [20, 16 + 4] = 20, $dd_2{}^r = max$ [17, 23] = 23, and $dd_1{}^r = max$ [21, 31] = 31. At t = 5, A_1 calculates slack times of opr_{11} and opr_{22} using relaxed due dates $dd_1{}^r$ and $dd_2{}^r$, and finds that s_{22} = 23 – (4 + 7) – 5 = 7 < s_{11} = 31 – (3 + 5 + 8) – 5 = 10. Therefore, opr_{22} is selected, instead of opr_{11}. Similarly, opr_{32} is scheduled before opr_{11}. The total tardiness cost of the schedule is (33 – 21) + (23 – 17) + (16 – 16) + (20 – 20) = 18, from J_1, J_2, J_3, and J_4, respectively. The example shows that, with the indicative information of relaxed due dates, A_1 adjusts its decisions to take A_2's processing conditions into account. This look-ahead coordination improves quality of the generated schedule.

Evaluation of the Approach

We hypothesized that in a tightly coupled, real-time environment, system performance can be improved by increasing agents' visibility on global conditions and extracting useful cues for agents' decision adjustment. Agents' predictions of future decisions are used and written on a shared memory so that agents can obtain a broader view of problem solving conditions. We developed relaxed due date as an useful indication of global conditions that is incorporated in agents' decision rules. Analytically, we can identify a number of factors of this look-ahead coordination: (1) the accuracy of the decision rule used in agents' decision forecasts, (2) the credibility of the relaxed due date information indicating global conditions, which is affected by the complexity of agents' interaction, (3) the availability of indicative information for decision adjustment.

In job shop scheduling, a more accurate priority rule produces better schedules. The first factor is related to the accuracy of the priority rule used in dispatching an operation since agents use the same priority rule to predict future decisions. The second and the third factors are related to the shop conditions. Most of the agents' interaction conditions can be measured by the number of bottleneck resources in the shop. The complexity of agents' interaction increases as the number of bottleneck resources increases. The other shop condition of concern is the due date tightness of jobs. Since jobs' due dates are relaxed only when their downstream operations are predicted to be tardy, this indicative information is less

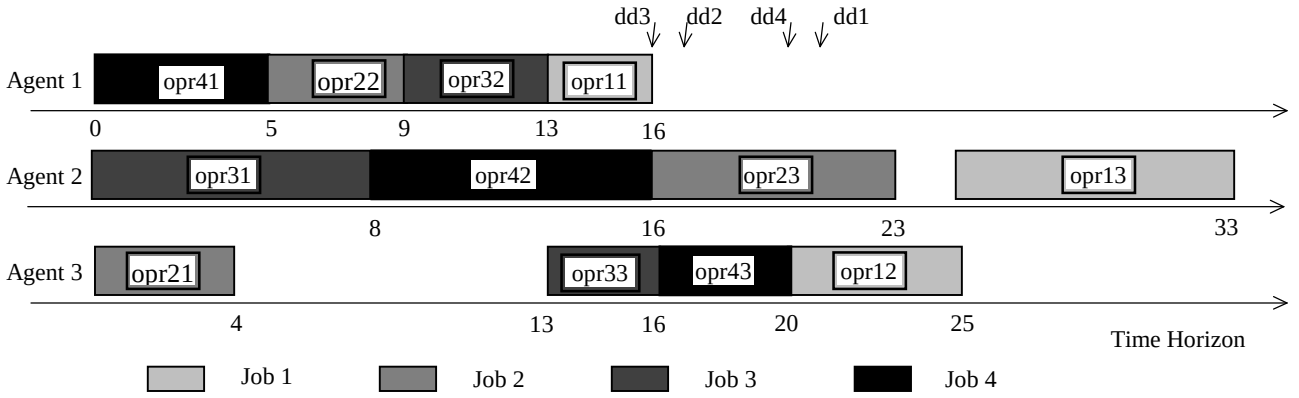

Figure 3: Example of dispatch scheduling enhanced by look-ahead coordination

available in shops with loose due dates than in shops with tight due dates.

We conducted an empirical study to test our hypothesis and analysis of the approach. Our goals are to: (1) compare the performance of COFCAST-RU enhanced dispatch scheduling and regular dispatch scheduling, (2) examine the effect of each of the three factors we identified on system performance. The experiments were conducted on a set of problems created in (Narayan et al. 1994) that consists of a total of 270 problems. Each problem has 50 jobs of 10 different routes and 5 resources. The jobs arrive dynamically with a Poisson distribution. Each job has one to five operations, and is assigned a due date and a weight that represents its importance. The objective function is the weighted tardiness of the schedule. We consider a set of priority rules - WCOVERT, S/RPT+SPT, CR+SPT, ATC, that are commonly used in Operations Research, and their more aggressive versions - X-WCOVERT, X-SRPT/SPT, X-CR+SPT, X-ATC, that strategically insert resource idle times that can be utilized to process more important jobs. For detail of these priority rules, please refer to (Morton & Pentico 1993).

For the purpose of experimentation, we implemented the coordination technique based on a blackboard model, e.g., agents communicate by reading/writing information on a shared memory space. This implementation short-cut does not affect our study of the performance of the look-ahead coordination. Our coordination technique is realistic for the following reasons: (1) a standard operating procedure is perhaps one of the most feasible approach in such a tightly coupled, real-time environment; (2) agents exchange very simple messages (e.g., operation start times, jobs' relaxed due dates, etc.) and need no response from other agents; (3) the look-ahead coordination adds only little overhead to the standard operating procedure.

Experimental Results

We report our experimental results in performance indices. The performance index (PI) of a method x on a problem is calculated by $PI_x = 100\% \times (S_x - S_B) / (S_S - S_B)$, where S_x is the score of method x, S_B is the best score known, and S_S is the score of a ``strawman". We used the naïve First Come-First Serve (FCFS) rule as the ``strawman". Because job shop schedule optimization is NP-complete and because for many of these problems there is no optimum known, we consider as the optimal values the results from an extensive search technique, e.g., Tabu Search, reported in (Narayan et al. 1994). The performance index can be interpreted as the percentage of error of each method from the estimated optimal.

rules	dispatch	w/ COFCAST	imp.
X-WCOVERT	5.46	6.08	–11.4%
WCOVERT	6.69	6.85	–2.4%
S/RPT+SPT	7.20	6.40	+11.1%
X-S/RPT+SPT	6.02	5.04	+16.3%
CR+SPT	5.65	4.65	+17.7%
X-CR+SPT	4.20	3.23	+23.1%
ATC	4.75	3.30	+30.5%
X-ATC	3.38	1.82	+46.2%

Table 1: Performance of COFCAST-RU on dispatch scheduling

Table 1 reports the average performance on the problem set by regular dispatch scheduling and COFCAST-RU enhanced dispatch scheduling with each priority rule we considered. COFCAST-RU improves system performance with six out of eight priority rules. With X-ATC rule, COFCAST-RU improved the scheduling quality of dispatch scheduling by 46.2%, and obtained a performance index of 1.82, e.g., 1.82% from the estimated optimal. The results show that COFCAST-RU is able to improve the performance of dispatch scheduling and is quite effective with both ATC and X-ATC rules.

Computationally, dispatch scheduling is very fast. For example, for a problem of 10 jobs and 5 machines, e.g., 50 operations, it took only 0.1 CPU seconds to generate a schedule. The look-ahead coordination is computationally efficient. It requires only 1.6 times the computational cost of regular dispatch scheduling in our experiment.

The results also reveal the effect of the accuracy of a priority rule. In general, COFCAST-RU improves dispatch scheduling better when the priority rule becomes more accurate, e.g., from SRPT/SPT to CR+SPT, to ATC. COFCAST-RU does not work well when COVERT rule is used because its priority index function does not differentiate jobs with large slack times, e.g., they are all assigned an index of zero. This is problematic for making forecast as it may lead to erroneous information and bad decision adjustment. In addition, for the same priority rule, the effect of COFCAST-RU was magnified by the aggressive version (X-) of the rule. Overall, the results show that the success of the look-ahead coordination is proportional to the accuracy of agents' decision rules.

Priority Rules	COFCAST-RU Improvement		
	Bot.=1	Bot.=2	Bot.=5
S/RPT+SPT	17.1%	13.5%	3.4%
X-SRPT+SPT	27.0%	17.0%	6.7%
CR+SPT	27.9%	19.8%	6.9%
X-CR+SPT	37.9%	23.6%	10.5%
ATC	37.6%	32.1%	20.8%
X-ATC	55.6%	45.8%	36.2%

Table 2: Performance improvement of COFCAST-RU by numbers of bottleneck resources

Table 2 reports the performance of COFCAST-RU in terms of improvement percentage over regular dispatch scheduling in problems with different number of bottleneck resources. COFCAST-RU's improvement percentage monotonically drops as the number of bottleneck resources increases. The reason is that when there are more than one bottleneck resource, interaction among resources becomes more complex.

While agents extract indicative information (relaxed due dates) from different forecasts by selecting the one predicting the most tardiness, the credibility of this information is reduced as the number of bottleneck resources increases. Overall, the results show that the look-ahead coordination is affected by the complexity of agents' interaction. However, the effects are less substantial when more accurate decision rules are used.

Table 3 reports the performance of COFCAST-RU in terms of improvement percentage over regular dispatch scheduling at different levels of due date tightness. In problems with loose due dates, e.g., tardy=0.5, COFCAST-RU performed less well than regular dispatch scheduling with less accurate rules. However, COFCAST-RU's improvement percentage sharply increases when due dates become tighter. This is related to the fact that the availability of indicative information depends on due date tightness. In COFCAST-RU, indicative information (e.g., relaxed due date) is available only when jobs are predicted to be tardy. Therefore, in problems with tighter due dates, COFCAST-RU performs considerably well in improving dispatch scheduling by using more indicative information. In problems with loose due dates, occasional indicative information seems to mislead agents' decisions when the decision rule is less accurate. Overall, the results show that the availability of indicative information has the most significant effect on the look-ahead coordination.

Priority Rules	COFCAST-RU Improvement		
	Tardy=0.5	Tardy=0.7	Tardy=0.9
S/RPT+SPT	-14.8%	12.6%	21.9%
X-SRPT+SPT	-10.5%	18.3%	30.6%
CR+SPT	-18.8%	17.8%	32.8%
X-CR+SPT	-20.2%	20.6%	48.7%
ATC	1.3%	27.2%	38.1%
X-ATC	11.4%	37.4%	61.2%

Table 3: Performance improvement of COFCAST-RU by due date tightness

Conclusions

We have presented an approach for multiagent coordination in tightly coupled, real-time environments. The approach consists of a standard operating procedure and a look-ahead coordination. The main contribution of the paper is the development of a computationally efficient coordination technique that can easily be integrated with a standard operating procedure (e.g., dispatch scheduling) to improve system performance in tightly coupled, real-time environments. We have applied the approach to job shop scheduling, one of the most difficult NP-complete combinatorial optimization problems. Experimental results show that the approach effectively enhances the performance of dispatch scheduling for optimizing objective of weighted tardiness. We have also obtained similar results for other objective functions, e.g., makespan. Our future work includes extension to agents in charge of multiple resources and jobs with substitutable resources.

The approach is also potentially useful for extending the

contract net protocol (CNP) (Davis & Smith 1983). While CNP has been extended to deal with a competitive setting (Fischer et al. 1995) and varying levels of commitment by bounded rational self-interested agents (Sandholm & Lesser 1995), temporal planning (e.g., deadlines, makespan) is very important in many real world problems (e.g., project management). The approach provides a coordinated temporal look-ahead capability that is potentially useful for extending CNP in problems that involve temporal objectives. In the envisioned CNP extension, a manager agent provides additional information, e.g., task deadlines and interdependency. A contractor agent uses this information and an extension of our coordination procedure to estimate its local schedule and see whether it can perform the task within the specified deadlines. This would be helpful for the contractor agent in deciding whether to bid for the task. This capability is particular useful when (1) tasks have deadlines and interdependency, and (2) when a contractor agent receives penalties for not performing a task by its deadline. We are currently investigating this CNP extension.

References

Bond, A. H., and Gasser, L. eds. 1988. *Readings in Distributed Artificial Intelligence.* San Mateo, Calif.: Morgan Kaufmann.

Conry, S. E.; Meyer, R. A.; and Lesser, V. R. 1988. Multistage Negotiation in Distributed Planning. In *Readings in Distributed Artificial Intelligence*, 367-384. San Mateo, Calif.: Morgan Kaufmann.

Davis, R., and Smith, R. G. 1983. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence* 20:63-109.

Decker, K. S., and Lesser, V. R. 1995. Designing a Family of Coordination Algorithms. In Proceedings of the First International Conference on Multi-Agent Systems, 73-80. San Francisco, Calif.

Durfee, E. H. , and Lesser, V. R. 1991. Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation. *IEEE Transactions on Systems, Man, and Cybernetics* 21(5): 1167-1183.

Fischer, K.; Muller, J. P.; Pischel, M.; and Schier, D. 1995. A Model for Cooperative Transportation Scheduling. In Proceedings of the First International Conference on Multiagent Systems, 109-116. San Francisco, Calif.

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

Gasser, L., and Huhns, M. N. eds. 1989. *Distributed Artificial Intelligence.* Vol. 2. Los Altos, CA.: Morgan Kaufmann Publishers.

Huhns, M. ed. 1987 *Distributed Artificial Intelligence.* Altos, Calif.: Morgan Kaufmann Publishers.

Huhns, M., and Bridgeland, D. 1991. Multiagent Truth Maintenance. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6): 1437-1445.

Lesser, V., and Corkill, D. 1983. The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks. *AI Magazine* 4(3): 15-33.

Liu, J., and Sycara, K. P. 1995a. Emergent Constraint Satisfaction through Multiagent Coordinated Interaction. In *From Reaction to Cognition*: 107-121. Castelfranshi, C., and Muller, J. P. eds. Vol. 957 of Lecture Notes in Artificial Intelligence.

Liu, J., and Sycara, K. P. 1995b. Exploiting Problem Structure for Distributed Constraint Optimization. In Proceedings of the First International Conference on Multi-Agent Systems, 246-253. San Francisco, Calif.

Mackworth, A. K. 1987. Constraint Satisfaction. In *Encyclopedia in Artificial Intelligence,* 205-211. Shapiro, S. C. ed. New York: Wiley.

Morton, T. E., and Pentico, D. W. 1993. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.* New York: Wiley.

Narayan, V.; Morton, T. E.; and Ramnath, P. 1994. X-Dispatch Methods for Weighted Tardiness Job Shops, Technical Report, #1994-14, Graduate School of Industrial Administration, Carnegie Mellon Univ.

Sanholm, T., and Lesser, V. 1995. Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. In Proceedings of the First International Conference on Multi-Agent Systems, 328-335. San Francisco, Calif.

Sycara, K. P. 1988. Resolving Goal Conflicts via Negotiation. In Proceedings of AAAI-88, 245-250.

Sycara, K. P.; Roth, S.; Sadeh, N.; and Fox, M. 1991. Distributed Constraint Heuristic Search. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6): 1446-1461.

Yokoo, M.; Durfee, E.; Tshida, T.; and Kuwabara, K. 1992. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In Proceedings of the 12th IEEE International Conference on Distributed Computing Systems, 614-621.