Student Paper: Multi-Linked Negotiation in Multi-Agent System[*]

Tracking Number: 162

Xiaoqin Zhang
Computer Science Department
University of Massachusetts at Amherst
shelley@cs.umass.edu

Victor Lesser
Computer Science Department
University of Massachusetts at Amherst
lesser@cs.umass.edu

ABSTRACT

Multi-linked negotiation describes a situation where one agent needs to negotiate with multiple agents about different issues, and the negotiation over one issue influences the negotiations over other issues. Multi-linked issues will become important for the next generation of more complicated Multi-Agent Systems. However, most current negotiation research looks only at single issue negotiation and thus does not present techniques to reason and manage multi-linked issues. In this paper, we present a technique based on the use of a partial-order schedule and a measure of the schedule, called flexibility, which enables an agent to reason explicitly about the interactions among multiple negotiation issues. We show how an agent uses the partial-order schedule to effectively manage interacting negotiation issues; and how the flexibility is a key measure for ordering and managing negotiation issues. Experimental work is presented which shows this management technique for multi-linked negotiation leads to improved performance.

Keywords

multi-linked negotiation, partial-order schedule, flexibility

1. INTRODUCTION

Negotiation, an interactive communication among participants to facilitate a distributed search process, is an important technique that is used to effectively coordinate the behavior of cooperative agents in a Multi-Agent System(MAS). Negotiation is used for task allocation, resource allocation and conflict resolution. Multi-linked negotiation deals with multiple negotiation issues when these issues are interconnected. In a multi-task, resource sharing environment, an agent needs to deal with multiple related negotiation issues including: task contracted to other agents, task requested by other agents, external resource requirements for local activities and interrelationship among activities distributed among multiple agents. These issues are related to each other. The result of one issue influences the possible solutions for the other issues.

[*]This material is based upon work supported by the National Science Foundation under Grant No.IIS-9812755, DMI-0122173 and the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-99-2-0525. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory/IFTD, National Science Foundation, or the U.S. Government.

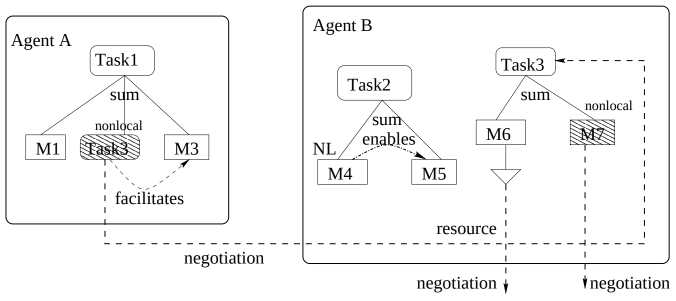

Figure 1: Directly Linked Relationship

The relationships among these negotiation issues can be classified as two types. One type of relationship is the *directly-linked* relationship: issue B affects issue A directly because issue B is a necessary resource (or a subtask) of issue A, the characteristics (such as cost, duration and quality) of issue B directly affect the characteristics of issue A. For example, as pictured in Figure 1[1], agent A has a nonlocal task “Task3” contracted to agent B while agent B needs to subcontract “M7” (a subtask of “Task3”) to another agent and request a resource for “M6” (another subtask for “Task3”) through negotiation. From agent B’s viewpoint, the negotiation with the agent who performs “M7” and the negotiation with the agent who controls the resource needed for “M6” have a *direct* influence on the negotiation with agent A on “Task3” since when and how “M7” will be performed and when the resource for “M6” is available affect when and how “Task3” can be performed.

Another type of relationship is the *indirectly-linked* relationship: issue 1 relates to issue 2 because they compete for use of a common resource. For example, as shown in Figure 2, agent A has a nonlocal task “M2” contracted to agent C while agent B has a

[1]All task plans shown in this paper use the TÆMS language [1], which is also used in our implementation and experiments.

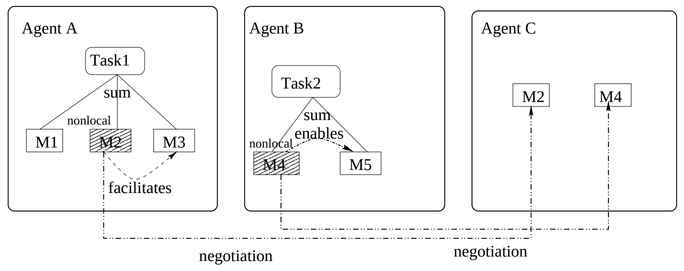

Figure 2: Indirectly Linked Relationship

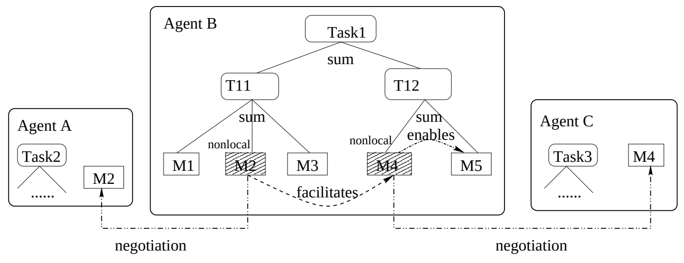

Figure 3: A Situation in Between

nonlocal task "M4" contracted to agent C, because of the limited capability of agent C, when "M2" will be performed *indirectly* affects when "M4" can be performed. Figure 3 describes a situation where there are interactions among *directly-linked* and *indirectly-linked* issues. Agent B has two nonlocal tasks: "M2" contracted to agent A and "M4" contracted to agent C. If the *facilitates*[2] relationship between "M2" and "M4" is exploited, the negotiation on "M2" and the negotiation on "M4" are *directly-linked*; otherwise, they are *indirectly-linked*.

In general, multi-linked negotiation (including both the *direct-linked* and *indirect-linked* relationships) describes situations where one agent needs to negotiate with multiple agents about different issues, where the negotiation over one issue has influence on the negotiations over other issues. The commitment on one issue affects the evaluation of a commitment or the construction of a proposal for another issue.

How can an agent deal with these multiple related negotiation issues? One approach is to deal with these issues independently just like separated issues, ignoring their interactions. If these negotiations are performed concurrently, there could be possible conflicts among these issues, hence the agent may not be able to find a combined solution that satisfied all issues without re-negotiation over some already "settled" issues. For example, in Figure 1, agent B negotiates with agent A and promises to finish "Task3" by time 20, meanwhile agent B also negotiates with other agent about "M7" and gets a contract that "M7" will be finished at time 30, then agent B finds it is impossible for "Task3" be finished by time 20 given its subtask "M7" will be finished at time 30. To reduce the likelihood that this type of conflict could occur, these negotiations could be performed sequentially; this means that the agent only deals with one negotiation issue at a time, and later negotiations are based on the previous negotiation results. However, this sequential process is not a panacea. First of all, the negotiation process takes much longer time when all the issues need to be negotiated sequentially, potentially using up valuable time and secondly there is no guarantee of finding an optimal solution or even whether any possible solution will be found. The latter problem can occur if the agent does not reason about the ordering of the negotiation issues and just treats them as independent issues with their ordering be random. In this situation, the result from the previous negotiations may make the later negotiation issues very difficult or even impossible. For instance, in Figure 1, if agent B first negotiates about "Task3" before starting the negotiations on "M6" and "M7", and the promised finish time of "Task3" results in tight constraints on the resource request of "M6" and the negotiation on "M7", then these negotiation may fail and the commitment on "Task3" has to be decommitted. One more problem is the difficulty in evaluating a commitment given that latter issues are undecided, and it is thus hard for the agent to find a local solution that will contribute effectively to the construction of a good global solution. For example, in Figure 3, agent B has two non-local tasks, task "M2" contracted to agent A and task "M4" contracted to agent C. If "M2" could be finished before "M4" starts, it will reduce the processing time of "M4" by 50%. Suppose agent B first negotiates with agent C and then negotiates with agent A; through the negotiation with agent C, it is decided that "M4" starts at time 20 and finishes by time 40 , but then it is found that task "M2" could finish at time 25. Given this latter information, if the start of "M4" is delayed to time 25, "M4" actually could be finished at time 35 because the *facilitates* effect. But this solution wouldn't be found if the agent ignores the interactions among these negotiation issues.

[2]A *facilitates* relationship from "M2" to "M4" means that the completion of "M2" will positively affect the execution "M4" by reducing its cost, shortening its process time and/or improving its quality.

These previous examples show us how important it is for an agent to reason about the interactions among different negotiation issues and manage them from a more global perspective. If done effectively, this permits the agent to minimize the possibility of conflicts among these different negotiation issues, and achieve better performance. In this paper, we introduce a partial-order schedule (see Section 3) as a basic reasoning tool for the agent to deal with multi-linked negotiation. It can be used to reason about the influence of a commitment of one issue on other negotiating issues. It also can be used to reason about the parameter associated with each negotiation issue in terms of the range of acceptable answers for a commitment and how it affects the flexibility available for an agent to schedule (and reschedule) its local activities.

The reminder of this paper is structured in the following manner. Section 2 introduces a supply chain scenario that used as an example to explain the ideas. Section 3 presents the definition of a Partial-Order Schedule (POS) and related algorithms. Section 4 details how the multi-linked negotiation works using the partial-order schedule and related reasoning tools. Section 5 reports the experimental work to evaluate the effect of different negotiation strategies on the agent's performance. Section 6 discusses about related work and Section 7 concludes and presents the areas of further work.

2. THE SCENARIO

We use the following supply chain example to explain our approach to solving a situation involving multi-linked negotiation situation. However, the following algorithm and the negotiation process are domain-independent and not restricted to this example[3].

[3]In this paper, the term "contractee agent" refers to the agent who performs the task for another agent and gets rewarded for successful completion of the task; "contractor agent" refers to the agent who has a task that needs to be performed by another agent and pays the reward to the other agent. The contractor agent and the contractee agent negotiate about the task and a contract is signed (a commitment is built and confirmed) if an agreement is reached during the negotiation.

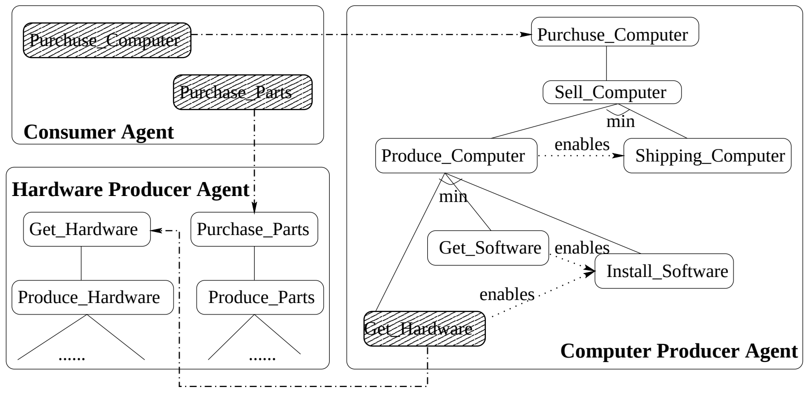

Figure 4: Three Agents

Consider the following example where agents with interrelationship shown in Figure 4.

- *Consumer_Agent:* generates two types of new tasks: "Purchase_Computer" task for the Computer_Producer_Agent, "Purchase_Parts" task for the Hardware_Producer_Agent. Each task includes following information:
  - deadline (dl): the latest-start-time for the task.
  - reward (r): if the task is finished as the contract requested, the contractee agent will get reward r.
  - decommitment penalty rate (p): If the contractee agent can not perform the task as it promised in the contract (i.e. the task could not finish by the promised finish time), it pays a decommitment penalty ($r * p$) to the contractor agent; if the contractor agent needs to cancel the contract after it has been confirmed, it needs to pay a decommitment penalty ($r * p$) to the contractee agent.
  - early finish reward rate (e): If the contractee agent can finish the task by the time (ft) as it promised in the contract, it will get the extra early finish reward: $max(e * r * (dl - ft), r)$[4] in addition to the reward r.
- *Computer_Producer_Agent:* receives "Purchase_Computer" task from the Consumer_Agent, decides if it should accept this task and if it does, what the promised finish time of the task should be. Figure 4 shows the local plan for producing computers, it includes a nonlocal task "Get_Hardware" that requires negotiation with the Hardware_Producer_Agent.
- *Hardware_Producer_Agent:* receives two types of tasks: "Get Hardware" from the Computer_Producer_Agent and "Purchase_Parts" from the Consumer_Agent. It decides whether to accept a new task and what the promised finish time for the task is.

Suppose Computer_Producer_Agent has received the following two tasks:
task name : Purchase_Computer_A
arrival time: 5
earliest start time: 10 (arrival time + estimated negotiation time(5))[5]
deadline: 70
reward: r=10
decommitment penalty rate: p=0.5
early finish reward rate: e=0.01
task name : Purchase_Computer_B
arrival time: 7
earliest start time: 12 (arrival time + estimated negotiation time(5))
deadline: 100
reward: r=10
decommitment penalty rate: p=0.6
early finish reward rate: e=0.005
The agent's local scheduler[8] reasons about these two new tasks according to above information: their earliest start times, deadline, estimated process times and the rewards, generates the following agenda which includes the accepted tasks:

[10, 50] Purchase_Computer_A
[50, 90] Purchase_Computer_B

This agenda is only a high level plan and does not include the execution details. The Computer_Producer_Agent checks the local plans for these tasks as shown in Figure 5 and finds there are four issues that need negotiation:

1. Negotiate with Consumer_Agent about the promised finish time of "Purchase_Computer_A";
2. Negotiate with Consumer_Agent about the promised finish time of "Purchase_Computer_B";
3. Negotiate with Hardware_Producer_Agent about "Get_Hardware_A": whether Hardware_Producer_Agent can accept this task and when it can be finished;
4. Negotiate with Hardware_Producer_Agent about "Get_Hardware_B": same concern as above.

These four issues are all related. "Get_Hardware_A" and "Purchase_Computer_A" are directed-linked, so are "Get_Hardware_B" and "Purchase_Computer_B"; "Get_Hardware_A" and "Get_Hardware_B" are indirected-linked, so are "Purchase_Computer_A" and "Purchase_Computer_B". We next show how Computer_Producer_Agent deals with these multi-linked negotiation issues using a partial-order schedule and related reasoning tools.

3. PARTIAL-ORDER SCHEDULE

A partial-order schedule is the basic reasoning tool that we use for multiple related negotiations. Here we present the formalization of the partial-order schedule and use an example to explain how it works for a multi-linked negotiation. Figure 6 shows the partial-ordered schedule from the example in Figure 5.

A *Partial-Order Schedule* represents a group of tasks with specified precedence relationship among them using a directed acyclic graph: $G = (V, E)$. $V = \{u\}$, each vertex in V represents a task. $E = \{< u, v > | P(u, v) \bigwedge (u, v \in E)\}$. Each edge (u, v) in E

[4]For each time unit the task finishes earlier than the deadline, the contractee agent get extra reward $e * r$, but the total extra reward would exceed the reward r.
[5]The task should not start until the contract has been confirmed

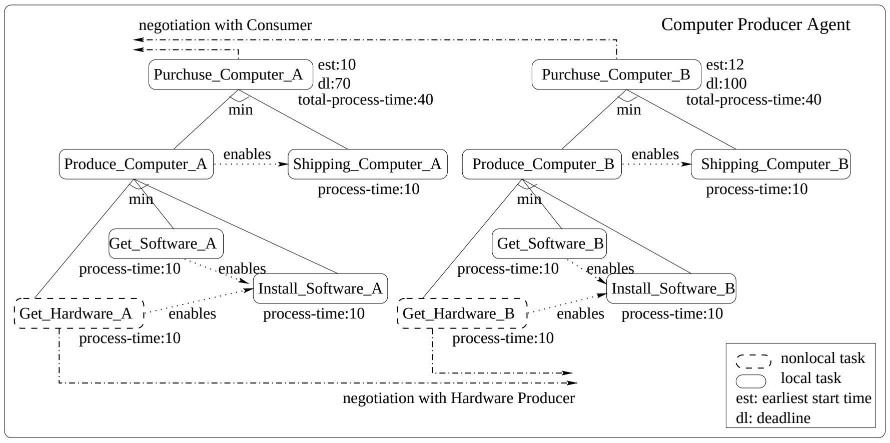

Figure 5: Computer_Producer_Agent's Local Plan

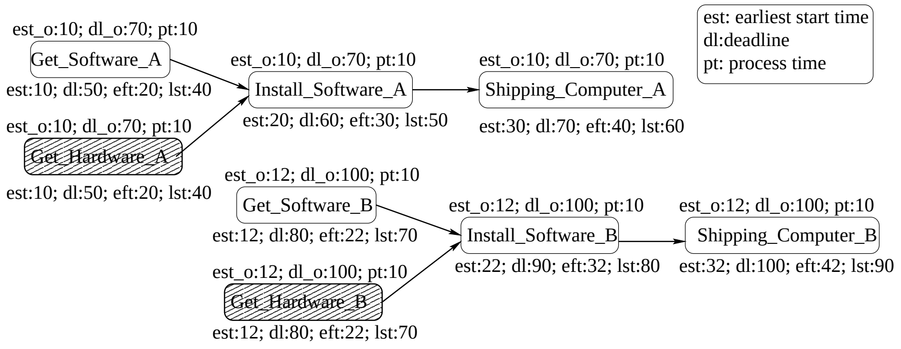

Figure 6: The Partial-Order Schedule of The Computer_Producer_Agent

denotes the precedence relationship between task u and task v, that is task u has to be finished before task v can start.

Task (t) is represented as a node in the graph, it is the basic element of the schedule. A task (t) needs a certain amount process time (t.process_time). A task can be a local task or a nonlocal task: a local task is performed locally (i.e, the "Get_Software_A" task) and a nonlocal task (i.e. the "Get_Hardware _A" task) is performed unlocally hence does not consume local process time.

The *Precondition of task t* is a set of tasks that need to be finished before task t can start: $Pre(\mathrm{t}) = \{s | s \in G \bigwedge < s, t > \in E\}$, task t can start only after all tasks in $Pre(t)$ have been finished. For example, the precondition of task "Install_Software_A" includes task "Get_Hardware _A" and task "Get_Software_A".

The *Postcondition of task t* is a set of tasks that only can start after task t has been finished: $Post(t) = \{r | r \in G \bigwedge < t, r > \in E\}$. For example, the postcondition of task "Install_Software_A" includes task "Shipping _Computer _A".

A task t has constraints of earliest-start-time (t.est) and deadline (t.dl). The earliest-start-time of task t (t.est) is determined by the earliest-finish-time of it's precondition ($eft[Pre(t)]$) and its outside-earliest-start-time constraint ($t.est\_o$):
$t.est = max(eft[Pre(t)], t.est\_o)$;
The earliest-finish-time of a task t ($t.eft$) is defined as:
$t.eft = t.est + t.process\_time$;
The earliest-finish-time of a set of tasks V ($eft[V]$) is defined as the earliest possible time to finish every task in the set V, it depends on the earliest-start-time and the duration of each task. For example, in Figure 6, outside-earliest-start-time constraint for task "Install_Software_A" is 10 (same as its super task 'Purchase_Computer_A"), the earliest-finish-time for its precondition is 20 (assume "Get_Hardware _A" could finish at its earliest possible time), then the earliest-start-time for task "Install_Software_A" is 20.

The deadline of task t ($t.dl$) is determined by the latest-start-time of its postcondition ($lst[Post(t)]$) and its outside-deadline-constraint ($t.dl\_o$):
$t.dl = min(lst[Post(t)], t.dl\_o)$;
The latest-start-time of a task t (lst(t)) is defined as:
$t.lst = t.dl - t.process\_time$;
The latest-start-time of a set of tasks V ($lst[V]$) is defined as the latest time for the tasks in this set to start without any task missing its deadline, it depends on the deadline and the duration of each task.

The *Flexibility of Task t* represents the freedom to move the task around in this schedule.
$F(t) = \frac{t.dl - t.est - t.process\_time}{t.process\_time}$.
For example, $F(Get\_Software\_A) = \frac{50-10-10}{10} = 3$.

The *Flexibility of a Schedule S* measures the overall freedom of this schedule, it is the sum of the flexibility of each activity weighted by its process time of the process time of the schedule. The flexibility of the task with a longer process time has a bigger influence on the flexibility of the schedule.
$F(S) = \sum_{t \in S} F(t) * \frac{t.process\_time}{(\sum_i t_i.process\_time)}$

A *Feasible Linear schedule*[6] is a total ordered schedule of all ac-

[6]Partial-Order Schedule is a representation and reasoning tool of a group of tasks and their interrelationship, it is not an executable schedule for the agent. To translate a partial-order schedule to an executable linear schedule, there are two different assumptions:

tivities with or without interruptible activities, that fulfills following conditions:

- Each task t takes n (n>=1) time periods ($p_i, i = 1, ...n$)for execution, $\sum_i p_i = t.processtime$;
- All precedence relationships are valid;
- All EST and DL constraints are valid;

A partial-order schedule is a *Valid Partial-Order Schedule* if there exists at least one feasible linear schedule that can be produced from this partial-order schedule without additional constraint and with the interruptible execution assumption. Without additional constraint and with the interruptible execution assumption, for a task t with the range [EST, DL], no matter what time t is executed during this range, there exists at least one feasible sequential schedule that can be produced from this partial schedule, then the range [EST, DL] for t is a *free-range* because task t can be executed during any period in this range.

We have built the following algorithms to support the negotiation based on the partial-ordered schedule. The details of these algorithm are described in [10].

- *Propagate_EST_DL*: Given a set of tasks with the outside constrains of the earliest-start-time and deadline, the duration of every task and the precedence relationship among the tasks, find the t.est and t.dl for each task t according to above definition.
- *Feasible_Schedule*: Translate a partial-order schedule into an executable linear schedule if the partial-order schedule is valid, otherwise report failure.
- *Range_Evaluation*: Find if a partial-order schedule is valid without trying to find a feasible linear schedule.
- *Find_NL_Range*: Find the biggest free range for task *nlt* in a partial-order schedule.;

4. MULTI-LINKED NEGOTIATION

4.1 General Ideas

To deal with the multiple related negotiation issues, the agent needs to analyze the relationships among these negotiation issues and find what is the influence of one issue on the others. First, the agent builds a partial-order schedule including the detailed plan for every task on the agenda which is generated by the agent's local scheduler, so that the agent knows what these tasks are and how they are related to each other. The agent sorts its current negotiation issues according to their importance, their flexibilities or the difficulties of negotiation processes[7], and finds the influence of the previous issue on the later issues. If the issue is a task requested by another agent, i.e. "Purchase_Computer_A", the agent finds the earliest-finish-time (eft_a) for this task using the partial-order schedule, then the agent reasons about what the promised finish time (pft_a) should based on the following concerns. First, the promised finish time (pft_a) should be not earlier than the earliest-finish-time (eft_a) and not later than the deadline (dl_a). $eft\_a <= pft\_a <= dl\_a$, let's assume $pft\_a = eft\_a + x$, x is a number to be decided ($0 <= x <= (dl\_a - eft\_a)$). By setting a specific value of pft_a in the partial-order schedule, the agent finds how this pft_a commitment affects other related issues, i.e. the free range of task "Get_Hardware_A" and the earliest-finish-time for "Purchase_Computer_B". If the commitment of pft_a leaves task "Get_Hardware_A" a very small free range and makes the negotiation difficult, the agent could increase x to allow task "Get_Hardware_A" have a bigger free range. If the commitment of pft_a implies a finish time that is too late for task "Purchase_Computer_B", the agent could increase x to provide an earlier finish time for task "Purchase_Computer_B". The agent also needs to reason about the balance between the amount of earlier reward it could get $e * (dl\_a - pft\_a)$ and the degree of flexibility left for other undecided issues. If the flexibility is low, the possibility of failing to successfully negotiation on other issues increases. In that case, the agent may have to pay a decommitment penalty and additionally get no reward.

If the issue is that a task needs to be contracted to another agent, i.e. "Get_Hardware_A", the agent finds the biggest free range for this task and the implication of this range on other issues: what free ranges for other tasks are consistent with this range. It reserves a reasonable flexibility for every undecided issue so as to make other negotiations easier; it also could reserve reasonable flexibility for the local schedule to cope with uncertainties in the execution behavior of the current scheduled tasks. Additionally, the early reward rate (e) could be decided based on the flexibility reasoning. If an earlier finishing time for "Get_Hardware_A" increases significantly the flexibility of other issues, making other negotiation much easier or making the local schedule more robust, then the early reward rate should be set to a large number; otherwise, a small number is appropriate. The importance of flexibility means that it should be one of the attributes in negotiation: an agent needs to decide how much flexibility it requires to be maintained or how much extra reward it wants to gain.

Once the consistent free ranges are found for each negotiation issue, each negotiation can be performed during these range concurrently without affecting each other or causing conflicts. By finding free ranges for each negotiation issue, the multi-linked negotiation problem is unlinked into several single issue negotiation problems. However, this may not always be the best approach in every situation. Another alternative approach is for the agent to make the decision to sequence these issues (not necessarily one by one, also could be group by group) according to their importance or their urgency. This partial sequencing of a set of negotiation issues leads to an increase in the likelihood of achieving an overall solution that solves all the negotiation issues in an effective manner. For example, in Figure 1, if the agent knows that the resource for "M6" is shared by many other users and may have limited availability, it could find the largest free range for task "M6" and only leave other issues with minimum ranges. This largest range is used in requesting a resource for "M6" so as to increase the possibility of a successful contract. Once the time slot for this resource is available, the range for "M6" could be reduced to fit its time slot, and thus other issues could have larger ranges for their negotiation.

Besides building the first proposal, the partial-order schedule also could be used to evaluate counter-proposals from other agents. If the counter-proposal includes a range outside the initial proposed range, the agent can check if it is consistent with other issues that

the task is interruptible or UN-interruptible. The interruptible execution assumption is that the agent can switch to another task during the execution of one task, and it can switch back at some point and continue the execution of the incomplete task. The UN-interruptible execution assumption does not allow execution of a task to be split into parts. In our work here we adopt the interruptible execution assumption.

[7]Meta level information is helpful for agent to estimate the difficulty of the negotiation process. For example, Computer_Producer_Agent could check with Hardware_Producer_Agent to find its flexibility for the next N time units and thus be able to make a good guess about how easy it is to get "Get_Hardware_A" and "Get_Hardware_B" finished during next N time units.

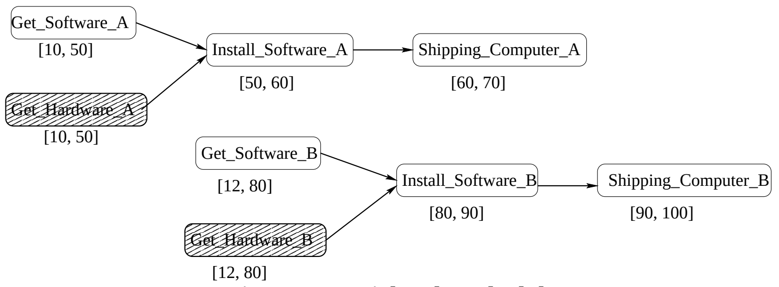

Figure 7: partial-order schedule

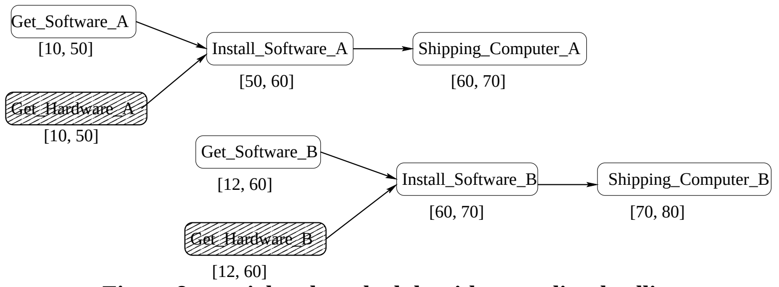

Figure 8: partial-order schedule with an earlier deadline

have already been decided or find its implication on those undecided issues, and decides if it is acceptable.

The next section provides an extended example to explain these ideas.

4.2 Indirectly Related Issues

Figure 7 shows the partial-ordered schedule with two nonlocal tasks "Get_Hardware_A" and "Get_Hardware_B", which are indirectly linked. The largest possible range for "Get_Hardware_A" is [10, 50]; the largest possible range for "Get_Hardware_B" is [12, 80]. In this situation, these two ranges are consistent, they are the free ranges. No matter what time they are finished at or before their deadlines (their postcondition tasks start no earlier than their deadline), there always exists a local feasible schedule (generated by the Feasible_Schedule algorithm):
$[10, 20]$ Get_Software _A
$[20, 30]$ Get_Software _B
$[50, 60]$ Install_Software_A
$[60, 70]$ Shipping _Computer _A
$[80, 90]$ Install_Software_B
$[90, 100]$ Shipping_Computer_B

So the negotiation on these two nonlocal tasks can be performed concurrently based on these two ranges. However, this schedule has very little flexibility, the three tasks "Shipping _Computer _A", "Install_Software_B" and "Shipping _Computer _B" have zero flexibility, that means if anything unexpected happens (i.e. the task "Install_Software_B' takes a little bit longer time than expected), the whole schedule will fail. So, during the negotiation, Computer_Producer_Agent may not want to build a commitment exactly like [10, 50] for "Get_Hardware_A", it needs to reserve some flexibility for its local schedule.

Suppose the deadline of the task "Produce_Computer _B" is set at 80, the largest range for "Get_Hardware_A" is [10, 50], while the largest possible range for "Get_Hardware_B" is [12, 60], as shown in Figure 8. This time, these ranges are not consistent, since the Range_Evaluation algorithm finds it is impossible to have a feasible linear schedule given the time slot below is overloaded:
$[60, 70]$ Shipping _Computer _A, Install_Software_B

The Find_NL_Range algorithm is used to find the consistence range of these two tasks. Assume these tasks are sorted according to their flexibility in increasing order:
F(Get_Hardware_A) = 3
F(Get_Hardware_B) = 3.8
So the agent works on the task "Get_Hardware_A" first. The range for task "Get_Hardware_B" is set to a minimum range (i.e. here the minimum range is defined as a range with flexibility 1) [12, 32], so the task "Get_Hardware_A" could find a larger possible range: the range is found as [10, 50]. After the range of "Get_Hardware_A" is decided, the Find_NL_Range algorithm found the range [12, 40] is the largest range for task "Get_Hardware_B", which is consistent with the range of "Get_Hardware_A". Hence the concurrent negotiation could be performed using these two ranges. On the other hand, suppose the agent feels that "Get_Hardware_B" is more important than "Get_Hardware_A" and is more difficult to find an acceptable contract, it could first use the maximum range [12, 60] for 'Get_Hardware_B" as the basis for negotiation. The time slot in the resulting contract then defines the free range for negotiating a contract for "Get_Hardware_A".

4.3 Directly Related Issues

Figure 7 also shows examples of *directly-linked* issues. Computer_Producer_Agent needs to find a promised finish time for the task "Purchase_Computer_A" and task "Purchase_Computer_B", which are directly linked to task "Get_Hardware_A" and task "Get_Hardware_B" respectively. The earliest-finish-time can be calculated by assuming that the nonlocal task finishes at its earliest possible time, i.e. the earliest-finish-time for the task "Purchase_Computer_A" is 40 given the task "Get_Hardware_A" finished at time 20. However, this assumption leaves zero flexibility for "Get_Hardware_A" and hence may cause failure of the negotiation on this task. Alternatively, the agent could decide how much flexibility (f_i) it needs to reserve for each nonlocal task (nlt_i) based on following concern:

1. the negotiation difficulty of task nlt_i based on its estimation and experience ;
2. the decommitment cost of the task T_i (T_i is the task whose plan includes nlt_i) ;

	Policy	Tasks Received	Tasks Accepted	Task Canceled	Task Early Finished	Decommit Penalty	Early Reward	Utility
Computer Producer	1	60	59	27	33	123	283	391
Computer Producer	2	60	60	0.5	0	2.9	0	413
Computer Producer	3	60	60	1.7	53	8.3	297	697
Hardware Producer	1	87	87	27	29	0	36	268
Hardware Producer	2	84	84	9.6	0	0	0	256
Hardware Producer	3	87	87	11	17	0	32	294

Table 1: comparison of performance

3. the early reward rate of the task T_i;

Based on these concerns, the range reserved for task nlt_i could be: [nlt_i.est, nlt_i.est + (1+f_i)*nlt_i.process_time]. Sorting tasks T_i by the early reward rate in decreasing order, the promised finish time of task T_i can be calculated using the Feasible_Schedule algorithm.

In this example, assuming that Computer_Producer_Agent decides to reserve flexibility 1 for each of nonlocal task, and also decides to calculate the finish time of the task "Purchase_Computer_A" first because it has a higher early finish reward rate, then we have following results:

1. The range reserved for "Get_Hardware_A" is [10, 30];
2. The finish time for "Purchase_Computer_A" is 50, the early reward it will get is: (70-50)*0.01*10 = 2;
3. The range reserved for "Get_Hardware_B" is [12, 32];
4. The finish time for "Purchase_Computer_B" is 70, the early reward it will get is: (100-70)*0.005*10 = 1.5;

The local feasible schedule is:
$[10, 20]$ Get_Software _A
$[20, 30]$ Get_Software _B
$[30, 40]$ Install_Software_A
$[40, 50]$ Shipping _Computer _A
$[50, 60]$ Install_Software_B
$[60, 70]$ Shipping _Computer _B

Based on above schedule, the range for "Get_Hardware_B" could be updated as [12, 50] since it does not need to be finished before time 50. All these four issues can be negotiated concurrently based on the above results. On the other hand, if the decommitment penalty for "Purchase_Computer_A" and "Purchase_Computer_B" is high and the schedule of Hardware_Producer_Agent is busy, the agent could first negotiate on "Get_Hardware_A" and "Get_Hardware_B" and subsequently negotiate on "Purchase_Computer_A" and "Purchase_Computer_B", .

5. EXPERIMENT

We have implemented an agent architecture including the agent controller, agent negotiater and execution components. All above algorithms and procedures associated with reasoning about the partial-order schedule have been implemented so as to enable the reasoning in the multi-linked negotiation process as indicated in the examples described previously. We designed the following experiment to study how the different negotiation strategies which involve different reasoning efforts affect the agent's performance.

The experimental environment is set up based on the scenario described in Section 2. Three agents were built using the JAF agent framework [9]. New tasks were randomly generated with decommitment penalty rate $p \in [0, 1]$, early finish reward rate $e \in [0, 0.1]$, and deadline $dl \in [45, 105]$, and arrive at the contractee agents periodically. The local scheduler of the agent schedules all incoming new task according to their earliest-start-times, deadline, process times and the rewards and generates an agenda (i.e. agenda on page 3) including the accepted tasks. From this agenda, the agent can find the scheduled finish time of each task. It could continue the negotiation about these incoming tasks just based on the information from this agenda without further reasoning about the detailed plan for each task (Actually, that is what the agent does when using the "Earliest-Finish-Time Policy" and the "Deadline Policy"). At the same time, if the local plan of these accepted tasks involves any nonlocal task *nlt*, then the Find_NL_Range procedure is used to find the earliest-start-time and the deadline of the task nlt, the agent would then start negotiation with the other agent about task *nlt* based on this time range.

In this experiment, Computer_Producer_Agent needs to deal with the multi-linked negotiation issues related to the incoming task "Purchase_Computer" and the outgoing task "Get_Hardware". The following three different negotiation strategies were tested:

1. Earliest-Finish-Time Policy. The agent finds the scheduled finish time of the task from its agenda and promises it as the finish time in the contract with the intention to maximize the early finish reward.
2. Deadline Policy. The agent promises the finish time which is the same as the deadline of the task with no consideration of the early finish reward.
3. Flexibility Policy. The agent analyzes its detailed partial-order schedule, if nonlocal tasks are found, it arranges reasonable flexibility (1, in this experiment) for each nonlocal task, and based on this arrangement, the finish time of the incoming task is decided and promised to the contractor agent.

In above all three cases, the multiple negotiations are performed concurrently based on the free ranges found by the partial-order schedule. However, with the first two policies, the agent does not reason about the interaction among issues or managing the flexibilities for each issue.

The experiments are performed in the MASS simulator environment [3]. Every group experiment has the system running for 1000 time clicks three times, each time using one of the three different polices. Table 1 shows the comparison of the agent's performance using difference policies. For the Computer_Producer_Agent, who has multi-linked negotiation issues, the flexibility policy is obviously better than the other two policies; it gets more early reward and pays fewer decommitment penalties.[8] For Hardware_Producer_Agent,

[8]Using t-test, With the 0.01 Alpha-level, the following hypothesis is accepted: when using the flexibility policy, Computer_Producer_Agent achieves an extra utility that is more than

the Earliest-Finish-Time Policy and the Flexibility Policy make no difference to the agent's decision making processes, since the agent has no sub-contracted task that needs consideration. The reason that the Earliest-Finish-Time Policy provides less utility is because the Computer_Producer_Agent cancels more task requests (because the finish times Hardware_Producer_Agent could provide are too late) and hence the Hardware_Producer_Agent has fewer tasks to perform and gains that the less reward. These experiment shows that in a multi-linked negotiation situation, it is very important for the agent to reasoning about relationship among different negotiation issues and leave reasonable flexibility for them. This type of reasonings decreases the likelihood of decommitment for previously settled issues and thus gains more utility.

6. RELATED WORK

To our knowledge, there is no work that has addressed the *directly-linked* relationship in the negotiation process. There is some work that takes into account the *indirectly-linked* relationship among multiple negotiation issues. Level commitment[5] allows agent to decommit by paying a decommitment penalty. A statistical model is used to predict future events so that the agent can calculate the opportunistic cost for the current commitment. When a new task arrives, the agent can backtrack from its previous decision by paying a decommitment penalty to get a better local solution. Also Sandholm[6] has developed a complex contract type - clustering-swap-multiagent (CSM contract) which allows tasks to be clustered, and then swapped between agents and even circulated among agents. He has proved that this CSM-contracts is sufficient for reaching global task allocation optimum in a finite number of contracts. This work deals with *indirectly-linked* issues by introducing complicated contract types, however it does not reason about the interrelationship among tasks and the influence of the temporal constraints on tasks as in our work. In research on the distributed meeting scheduling [7] problem, multiple meeting scheduling sessions were allowed to going on concurrently. Two different commitment strategies were explored: one where the agent blocked the proposed time and the other where the time was not blocked until an agreement is reached. Adaptive selection of the commitment strategy according to environment factors is recommended. However, in both of these works, the agent does not explicitly reason about the relationship among different issues under negotiation. In order to propose offers or counter-offers to minimize the conflict and optimize the combined outcome.

Our partial-order schedule work is related to the Graphical Evaluation and Review Technique(GERT) [4] which is used for project scheduling and management. The big difference between this work and ours is that this work is not oriented to negotiation, all activities are local and can be managed with authority, thus they do not reason about free ranges, consistent ranges and schedule flexibilities which we feel are critical for agent to effectively manage multi-linked negotiation.

7. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have studied multi-linked negotiation and looked at different relationships such as *directed-linked* and *indirectly-linked* relationships. We built a partial-order schedule representation and a set of related algorithm as a toolkit to deal with multi-linked negotiation. Additionally, we explored how flexibility is an important factor in successful negotiation and how the agent use reasonable flexibility strategy based on allocating flexibility to linked negotiation issues so as to achieve higher performance. In the future work, we would like to study how flexibility helps an agent deal with uncertainty in execution of task and the arrival of new tasks. In this work, we assumed that local task execution was deterministic which is not true for most application domains. We also plan to use a more complex measure of flexibility [2] which characterizes the interaction among tasks beside the time issue. Additionally, we would like to study what is a good strategy for an agent to decide whether to perform all negotiation issues concurrently and if not, what sequence should they be done. We also want to explore how meta-level information about other agent's loads would help in this decision-making process.

64% of the utility gained when using the the Earliest-Finish-Time Policy.

8. REFERENCES

[1] Decker, K., Lesser, V. R. Quantitative Modeling of Complex Environments. In International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behavior., Volume 2, pp. 215-234, 1993.

[2] Deshmukh, A. V., Talavage, J. J., and Barash, M. M. Complexity in Manufacturing Systems: Part 1 - Analysis of Static Complexity IIE Transactions, vol 30, number 7, pp. 645-655, 1998.

[3] Horling, Bryan, Lesser, Victor, Vincent, Regis. Multi-Agent System Simulation Framework. In 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, EPFL, Lausanne, Switzerland, August 2000.

[4] Pritsker, A.A.B. GERT Networks (Graphical Evaluation and Review Technique) The Production Engineer, October 1968

[5] Sandholm, T. and Lesser, V. 1996. Advantages of a Leveled Commitment Contracting Protocol. Thirteenth National Conference on Artificial Intelligence (AAAI-96), pp. 126-133, Portland, OR, .

[6] Sandholm, T. 1996. Negotiation among Self-Interested Computationally Limited Agents. Ph.D. Dissertation, University of Massachusetts at Amherst, Department of Computer Science.

[7] Sandip Sen and Edmund H. Durfee A Formal Study of Distributed Meeting Scheduling Group Decision and Negotiation, volume 7, pages 265-289, 1998.

[8] Wagner, Thomas and Lesser, Victor. Relating Quantified Motivations for Organizationally Situated Agents. In Intelligent Agents VI: Agent Theories, Architectures, and Languages, Springer

[9] Vincent, R.; Horling, B.; Lesser, V. An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator. In Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems. Volume 1887, Wagner & Rana (eds.), Springer, pp. 102127, 2000

[10] Zhang, Xiaoqin and Lesser, Victor. Dealing With Multi-Linked Negotiation with Reasoning. Technicial Report of Computer Science Department, UMass., 2001.