
Autonomous Agents and Multi-Agent Systems, 10, 165–205, 2005

Efficient Management of Multi-Linked Negotiation Based on a Formalized Model

XIAOQIN ZHANG x2zhang@umassd.edu
Department of Computer and Information Science, University of Massachusetts at Dartmouth

VICTOR LESSER AND SHERIEF ABDALLAH lesser,shario @cs.umass.edu
Department of Computer Science, University of Massachusetts at Amherst

Abstract. A **Multi-linked negotiation problem** occurs when an agent needs to negotiate with multiple other agents about different subjects (tasks, conflicts, or resource requirements), and the negotiation over one subject has influence on negotiations over other subjects. The solution of the multi-linked negotiations problem will become increasingly important for the next generation of advanced multi-agent systems. However, most current negotiation research looks only at a single negotiation and thus does not present techniques to manage and reason about multi-linked negotiations. In this paper, we first 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 negotiations. Next, we introduce a formalized model of the multi-linked negotiation problem. Based on this model, a heuristic search algorithm is developed for finding a near-optimal ordering of negotiation issues and their parameters. Using this algorithm, an agent can evaluate and compare different negotiation approaches and choose the best one. We show how an agent uses this technology to effectively manage interacting negotiation issues. Experimental work is presented which shows the efficiency of this approach.

Key words: multiple related negotiations, agent reasoning and control, conflict resolution, performance optimization.

1. Introduction

Multi-linked negotiation describes a situation where one agent needs to negotiate with multiple agents about different issues (tasks, conflicts, or resource requirements), and the negotiation over one issue affects the negotiations over other issues. In a multi-task, resource-sharing environment, an agent needs to deal with multiple, related negotiation issues including: tasks contracted to other agents, tasks requested by other agents, external resource requirements for local activities, and interrelationships among activities distributed among different agents.

Consider the following example shown in Figure 1, which is a simplified supply chain containing four agents. The *Consumer Agent* represents the environment that generates tasks to be completed by the other three agents. When a new task is generated by the *Consumer Agent*, it indicates how much it is worth and its deadline. When the *Computer Producer Agent* receives task *Purchase_Computer* from the *Consumer Agent*, it also needs to sub-contract parts of the task *Get_Hardware* and *Deliver_Computer* to the *Hardware Producer Agent* and the *Transporter Agent*, respectively. The following three negotiations are interrelated: the negotiation between the *Computer Producer Agent* and the *Consumer Agent* on task

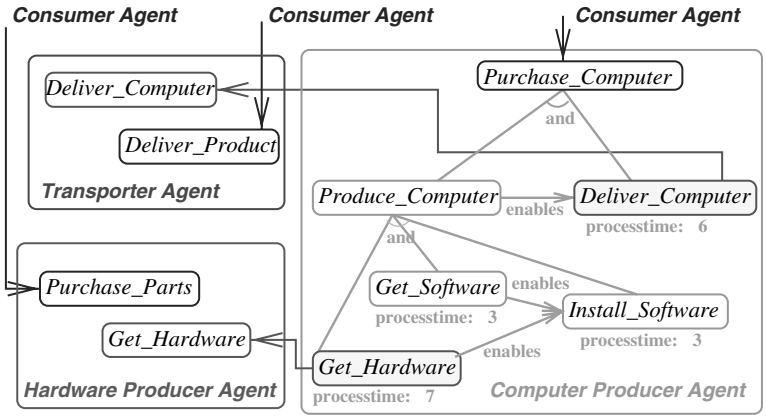

Figure 1. A supply chain scenario.

Purchase_Computer, the negotiation between the *Computer Producer Agent* and the *Hardware Producer Agent* on task *Get_Hardware*, and the negotiation between *Computer Producer Agent* and the *Transporter Agent* on task *Deliver_Computer*.

How can the agent deal with these interrelated negotiations? One approach is to deal with these negotiations independently ignoring their interactions.[1] If these negotiations are performed concurrently, there could be possible conflicts among the solutions to these negotiations; hence the agent may not be able to find a combined feasible solution that satisfies all constraints without re-negotiation over some already "settled" issues. For example, in Figure 1, suppose the *Computer Producer Agent* negotiates with the *Consumer Agent* and promises to finish *Purchase_Computer* by time 20, and concurrently the *Computer Producer Agent* also negotiates with the *Transporter Agent* about task *Deliver_Computer* and gets a contract that task *Deliver_Computer* will be finished at time 30, then the *Computer Producer Agent* will find it is impossible for task *Purchase_Computer* be finished by time 20 given that its subtask *Deliver_Computer* will not be finished until time 30.

To reduce the likelihood that this type of conflict occurs, these negotiations could be performed sequentially; the agent deals with only one negotiation at a time, and later negotiations are based on the results of earlier negotiations. This sequential process, however, is not a panacea. First of all, the negotiation process takes much longer when all the negotiations need to be negotiated sequentially, potentially using up valuable time (delaying when problem solving can begin) and reducing the potential solution space. For example, in Figure 1, suppose the deadline for completion of task *Deliver_Computer* is 20, the same as task *Purchase_Computer*. If the negotiation on task *Deliver_Computer* starts at 10 and finishes at time 12, then the execution of task *Deliver_Computer* can only start after time 12. However, if the negotiation on task *Deliver_Computer* starts at time 3, there is a larger time slot for the execution of task *Deliver_Computer*; hence, it is easier for the negotiation on task *Deliver_Computer* to succeed. Additionally, when the negotiation deadline is taken into consideration, a negotiation started later may lose any chance of success. For instance, in Figure 1, suppose the *Consumer Agent* associates a negotiation deadline

of 8 with the proposal of task *Purchase_Computer*, if the *Computer Producer Agent* replies to this proposal later than time 8 because it wants to settle its other negotiations first, it cannot get the contract on task *Purchase_Computer* accepted.

Second, even if all the negotiations are sequenced, there is no guarantee of an optimal solution or even of any possible solution. This problem can occur if the agent does not reason about the ordering of the negotiations and just treats them as independent negotiations, with their ordering being random. In this situation, the results from the previous negotiations may make later negotiations very difficult or even impossible to succeed. For instance, in Figure 1, if the *Computer Producer Agent* first negotiates about task *Purchase_Computer* before starting the negotiations on task *Get_Hardware* and task *Deliver_Computer*, and the promised finish time of task *Purchase_Computer* results in tight constraints on the negotiations on task *Get_Hardware* and task *Deliver_Computer*, these negotiations may fail and the commitment on task *Purchase_Computer* would have to be decommitted from.

One additional problem is caused by the difficulty in evaluating a commitment given that the result of later negotiations are unknown, and thus making it harder for an agent to find a local solution that will contribute effectively to the construction of a good global solution. For example, in Figure 2, agent *A* has two non-local tasks (the tasks that are performed by other agents), task *Ta*12 contracted to agent *B* and task *Ta*21 contracted to agent *C*. There is a "facilitates" relationship from *Ta*12 to *Ta*21. If *Ta*12 could be finished before *Ta*21 starts, it would reduce the processing time of *Ta*21 by 50%. Suppose agent *A* first negotiates with agent *C* and then negotiates with agent *B*; as a result of the negotiation with agent *C*, it is decided that *Ta*21 starts at time 20 and finishes by time 40, but then it is found through the negotiation with agent *B* that task *Ta*12 could only be finished by time 25. Given this later information, if the start of *Ta*21 is delayed to time 25, *Ta*21 actually could be finished at time 35 because of the *facilitates* effect. This solution would not be found, however, if the agent ignores the interactions among these negotiations.

These previous examples indicate how important it is for an agent to reason about the interactions among different negotiations and manage them from a more global perspective. If done effectively, this permits the agent to minimize the possibility of

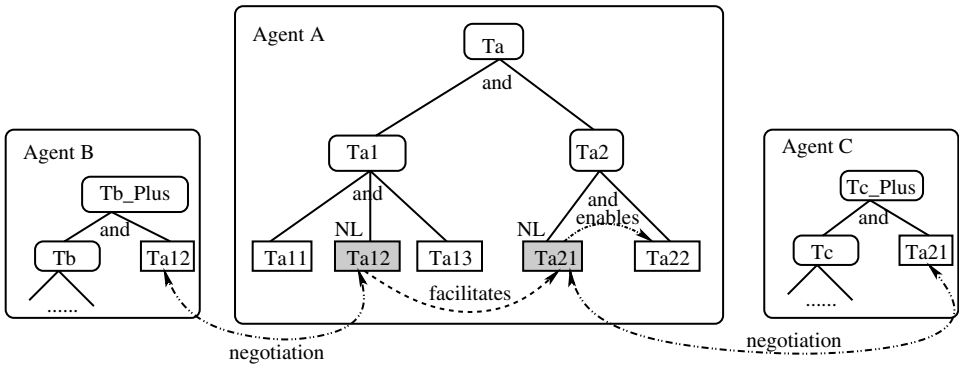

Figure 2. Negotiations linked by a "facilitates" relationship.

conflicts among the different negotiations and thus achieve better performance. Additionally, these examples show that it is difficult to deal with multi-linked negotiation problems because:

1. There are possible conflicts among related negotiations. If not resolved, these conflicts may cause the failure of the agent's local plan or reduce the agent's local utility achievement.
2. There are uncertainties associated with negotiations. Since the agent does not have perfect and complete knowledge of the other agents' states, the result of a negotiation is uncertain. The agent may have an estimation about the likely outcome of the negotiation, but it needs to be prepared for different outcomes.
3. There is a cost for negotiation. On one hand, the agent needs to allocate valuable computational and communication resources for negotiation. On the other hand, the time spent on negotiation may affect the outcome of the negotiation. For example, the longer time spent on negotiation may reduce the time available for execution hence reducing the possibility of finding a solution. Similarly, a delayed reply to a proposal may not be accepted if there are other agents who have already replied to it earlier.
4. The negotiation process needs to be interleaved with the agent's local planning and scheduling process because the agent needs to find a feasible local solution that satisfies all commitments and local constraints.

The multi-linked negotiation problem is not only a complicated problem, but also an important one because it actually happens in a number of application domains. For example, in a supply chain problem, negotiations go on among more than two agents. The consumer agent negotiates with the producer agent, and the producer agent needs to negotiate with the supplier agents. The negotiations between the producer agent and the supplier agents has a direct influence on the negotiation between the producer agent and the consumer agent. Figure 3 shows a supply chain example, where there are a number of companies, some of which produce parts for computers and some of which assemble computers, where others are distributors, stores and customers. Multi-linked problems occur throughout this system. We will also present a detailed supply chain scenario with multi-linked negotiations based on Figure 3 in Section 2, and use this scenario as an example throughout this paper. Another example of multi-linked negotiation is a distributed sensor network [5]. There are multiple sensors distributed at different locations, each of which has different coverage. Multiple targets move through the region and it takes a certain number (more than one) of sensors to track a target so as to get sufficient sensor data for acceptable tracking quality. Which sensors should be used to track which target during which time interval poses an interesting multi-linked negotiation problem.

In general, a **Multi-linked negotiation problem** occurs when an agent needs to negotiate with multiple other agents about different subjects, and the negotiation over one subject has influence on the negotiations over other subjects. The commitment of resources for one subject affects the evaluation of a commitment or the construction of a proposal for another subject. To solve a multi-linked negotiation

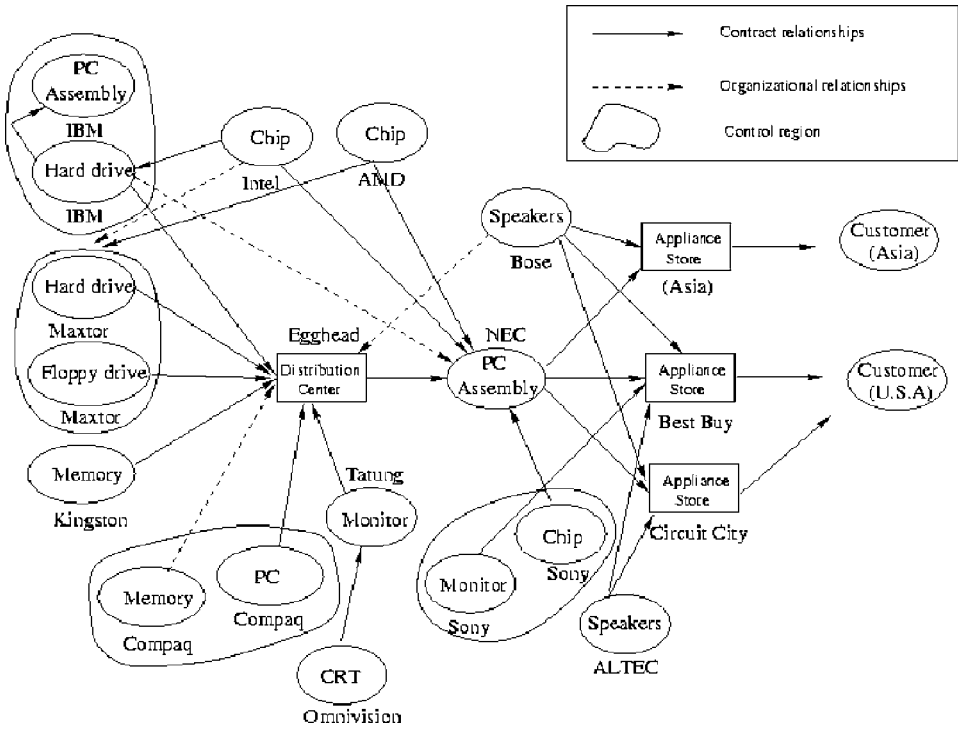

Figure 3. Supply chain example.

problem, an agent needs to find an efficient approach, which includes a temporal ordering of these negotiations and appropriate parameters for each feature in negotiations, so as to minimize the conflicts and maximize the agent's expected utility. In this paper, we first explicitly address this multi-linked negotiation problem and analyze it, then we develop a formalized model and a set of reasoning tools that enable an agent to find an near-optimal solution for this problem.

In the remaining of this paper, we will first introduce a detailed multi-linked negotiation scenario and basic assumptions in this work (Section 2). Next we will present a formalized model for the problem is presented in Section 3 . Using this model, the agent can find the best ordering of the negotiations and their parameters, and hence increase its local utility achievement. A partial order schedule and a set of related algorithms will be presented in Section 4, which are necessary for the agent to reason about the time constraints and the flexibility of each negotiation. The partial order schedule and the related reasoning tools also make parallel negotiations feasible by eliminating potential conflicts. An example to show how this model works is presented in Section 5. Three sets of experimental work are presented in Section 6. Section 6.1 examines the performance of the heuristic algorithm, Section 6.2 shows that this management technique for multi-linked negotiation leads to improved performance, and Section 6.3 shows it is important for agents to reason about flexibility in a multi-linked negotiation problem. Section 7 discusses related work,

with special attention on the relationship between the approach presented in this paper and another approach based on a combinatorial auction. Section 8 summarizes this paper.

2. Supply chain example

In this section, we describe the supply chain example presented in Section 1 in greater detail. This example will be used throughout the rest of this paper to explain the multi-linked negotiation problem. However, the negotiation process and the following approach are domain-independent and are not restricted to this application.

2.1. Supply chain scenario

There are four agents in Figure 1:

1. *Consumer Agent* generates three types of new tasks: *Purchase_Computer* task for *Computer Producer Agent*, *Purchase_Parts* task for *Hardware Producer Agent*, and *Deliver_Product* task for *Transporter Agent*.
2. *Computer Producer Agent* receives the *Purchase_Computer* task from *Consumer Agent*, and needs to decide if it should accept this task and, if it does, what the promised finish time of the task should be. Figure 1 shows the local plan for producing computers; it includes a non-local task *Get_Hardware* that requires negotiation with *Hardware Producer Agent*. It also includes a non-local task *Deliver_Computer* that requires negotiation with *Transporter Agent*.
3. *Hardware Producer Agent* receives two types of tasks: *Get_Hardware* from *Computer Producer Agent* and *Purchase_Parts* from *Consumer Agent*. It need to decide whether to accept a new task and what is its promised finish time.
4. *Transporter Agent* receives two types of tasks: *Deliver_Computer* from *Computer Producer Agent* and *Deliver_Product* from *Consumer Agent*. It needs to decide whether to accept a new task and what is its promised finish time.

We first define two generalized terms to make the following description easier. In the following description, we will use the term *contractor agent* to refer to the agent who performs the task for another agent and gets rewarded for the successful completion of the task; and *contractee agent* to refer to the agent who has a task that needs to be performed by another agent and pays a reward to the other agent. The *contractor agent* and the *contractee agent* negotiate about a task and a contract is signed (a commitment is built and confirmed) if an agreement is reached during the negotiation.

In this work, the negotiation process between agents is based on an extended contract net model [10, 13]:

1. *Contractee agent* announces a task by sending out a proposal.
2. *Contractor agent* receives this proposal, evaluates it, responds to it in one of three ways: by accepting it, by simply rejecting it, or by rejecting it but at the same time making a counter-proposal.

3. *Contractee agent* evaluates the responses, it either chooses to confirm an accepted proposal, or chooses to accept a counter-proposal.
4. *Contractee agent* awards the task to the chosen *contractor agent* based on the commitment (the mutually accepted upon proposal or counter-proposal) which is confirmed by both agents; the negotiation process then ends successfully. If a mutually agreed proposal/counter-proposal cannot be found, the negotiation process fails.

This process can be extended to a multi-step process by introducing an extended series of alternative proposals and counter-proposals. However, in this paper, we only focus on the two-step (proposal, counter-proposal) negotiation process. The implications of performing a multi-step negotiation instead of a two-step negotiation can be found in [17].

A proposal which announces that a task (t) needs to be performed includes the following attributes:

1. *earliest start time* (*est*): the earliest start time of task t; task t cannot be started before time *est*.
2. *deadline* (*dl*): the latest finish time of the task; the task needs to be finished before the deadline *dl*.
3. *minimum quality requirement* (*minq*): the task needs to be finished with a quality achievement no less than *minq*.[2]
4. *regular reward* (r): if the task is finished as the contract requested, the contractor agent will get reward r.
5. *early finish reward rate* (e): if the contractor agent can finish the task by the time (ft) as it promised in the contract, it will get the extra early finish reward: $max(e * r * (dl - ft), r)$,[3] in addition to the regular reward r.
6. *decommitment penalty rate* (p): if the contractor agent cannot perform the task as promised in the contract (i.e. the task could not be finished by the promised finish time), it pays a *decommitment penalty* $(p * r)$[4] to the contractee agent. Similarly, if the contractee agent needs to cancel the contract after it has been confirmed, it also needs to pay a *decommitment penalty* $(p * r)$ to the contractor agent.

When the potential contractor agent receives a task proposal, it evaluates it and decides to either accept it or reject it. If it accepts this proposal, it needs to decide what the promised finish time should be. If it rejects the proposal, it can either simply say "no" or generate a counter-proposal which modifies some of the attributes in the proposal to accommodate its local problem-solving context.

In the above discussion, we assume the negotiation is about a task that needs to be performed; however, the negotiation can also be about a nonlocal resource requirement necessary for the completion of a task. The agent can require a resource during a time period and pay for this resource usage. In this situation, some of the attributes specified in the proposal are different from those in the above description,[5] but the basic negotiation process is the same, and the methodologies we discuss in this paper are also suitable for negotiation over resources.

2.2. Detailed example of a multi-linked negotiation problem

Suppose *Computer Producer Agent* has received the following two tasks in the same scheduling time window:[6]

task name : Purchase_Computer_A

arrival time : 5

earliest start time : *10* (*arrival time* + *estimated negotiation time* (*5*))[7]

deadline : *40*

reward : $r = 10$

decommitment penalty : $p = 0.5$

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 : *50*

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 the above information: their earliest start times, deadline, estimated process times and the rewards. It then generates the following agenda which includes the following tasks:

Agenda 2.1 $[10, 26]$ *Purchase_Computer_A*$[26, 46]$ *Purchase_Computer_B*

In this agenda, task *Purchase_Computer_A* is scheduled during time range [10, 26], and task *Purchase_Computer_B* is scheduled during time range [26, 46]. This agenda is only a high level plan and does not include the detailed actions (methods) that need to be executed. The *Computer Producer Agent* checks the local plans for these tasks[9] as shown in Figure 4 and finds there are five negotiations:

1. Negotiate with *Consumer Agent* about the promised finish time of *Purchase_Computer_A*.[10]
2. Negotiate with *Consumer Agent* about the promised finish time of *Purchase_Computer_B*.
3. Negotiate with *Hardware Producer Agent* about whether it can accept the task *Get_Hardware_A* and if it accepts this task, what is the promised finish time.
4. Negotiate with *Hardware Producer Agent* about the task *Get_Hardware_B*, with the same concerns as above.

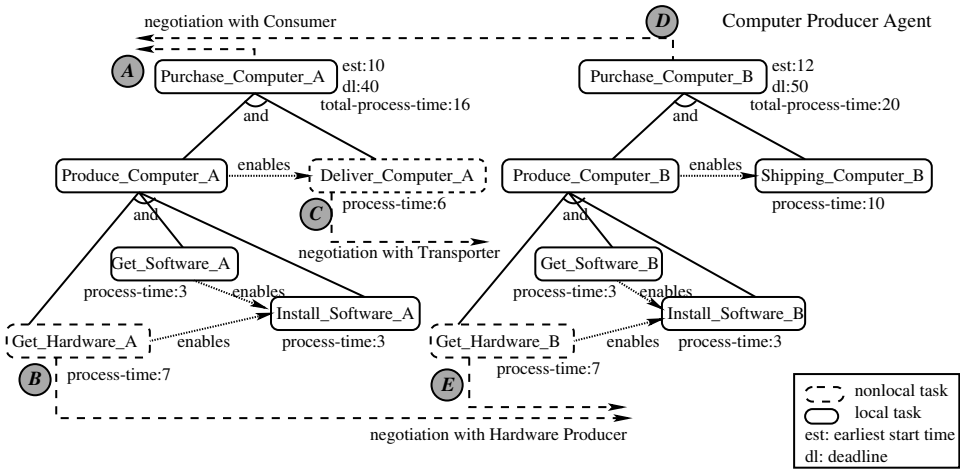

Figure 4. Computer producer agent's tasks.

5. Negotiate with *Transporter Agent* about whether it can accept the task *Deliver_Computer_A*, and if it accepts this task, what is the earliest start time and what is the promised finish time.

These five negotiations are all related. The potential relationships among multiple negotiation issues can be classified as two types. One type of relationship is the *directly linked* relationship: negotiations A and B are directly linked if negotiation B affects negotiation A directly because the subject in negotiation B is a necessary resource (or a subtask) of the subject in negotiation A. The characteristics (such as cost, duration and quality) of subject B directly affect the characteristics of subject A. For example, as pictured in Figure 1, the negotiation on the task *Purchase_Computer_A* is directly linked to the negotiation on the two tasks: *Get_Hardware_A* and *Deliver_Computer_A*. If either one of these two tasks fails, the task *Purchase_Computer_A* cannot be accomplished. Furthermore, when and how these two tasks are performed also affects the way that the task *Purchase_Computer_A* is going to be accomplished. In the same way, the negotiations about *Get_Hardware_B* and *Purchase_Computer_B* are directly linked.

Another type of relationship is the *indirectly linked* relationship: negotiation A and B are indirectly linked if the subjects in these negotiations compete for use of a common resource. For example, as shown in Figure 4, besides the task *Purchase_Computer_A*, *Computer Producer Agent* has another contract on task *Purchase_Computer_B*. Because of the limited capability of the *Computer Producer Agent*, when task *Purchase_Computer_A* will be performed affects when task *Purchase_Computer_B* can be performed. The negotiation about task *Purchase Computer_A* and the negotiation about task *Purchase_Computer_B* are *indirectly linked*.

The essential difference between directly linked and indirectly linked relationships is the following. If negotiation A and B are directly-related, then the failure of one

negotiation may cause the subject (task or resource) in the other negotiation to be infeasible or unnecessary. For example, if the subject B is a subtask of A, then the failure of negotiation on B will cause the task A to be infeasible if there is no other task that could substitute for task B; likewise, the failure of negotiation A will make the subtask B unnecessary. If negotiation A and B are indirectly linked, then there is no such influence between them. In the formalized model presented in Section 3, we will show that these two different relationships are represented differently.

2.3. *Analysis of the problem*

In general, multi-linked negotiation (including both the *directly linked* and the *indirectly linked* relationships) describes situations where one agent needs to negotiate with multiple agents about different issues, where the negotiation over one issue influences the negotiations over other issues. The characteristics of the commitment on one issue affects the evaluation of a commitment or the construction of a proposal for another issue. How can the agent deal with these interrelated negotiations? Two questions need to be answered. *The first question is in what order should the negotiations be performed.* Should all the negotiations be performed concurrently or in sequence? If in sequence, in what sequence? *The second question is how the agent assigns values for those attributes (also referred as "features") in negotiation*, such as the earliest start time, deadline, so as to minimize the potential conflicts among negotiations and maximize the utility of the agent as a result of multiple negotiations.

In a multi-linked negotiation problem, there are potentially many choices to order negotiations, such as doing some of them in parallel and some of them in sequence. Why is the order of negotiation important? First, because each negotiation issue has a negotiation deadline, set by the contractee agent, if the contractor agent cannot reply to a task proposal before the negotiation deadline, the negotiation fails. One reason for missing the negotiation deadline is that the contractor agent is busy on other negotiations before it decides to perform this negotiation. Furthermore, even if the negotiation is completed before its deadline, when the negotiation is started affects the likelihood of a successful negotiation. For example, when there are several potential contractor agents, the earlier a response to negotiation is received, the more likely the offer is accepted. Likewise, the earlier the contractee agent initiates the negotiation, the more likely the contractor agent is to accept the proposal, since the earlier a negotiation is started, the larger the space (time range) for the agent to find a feasible solution. For instance, given that the deadline for task *Get_Hardware_A* is 30, if the negotiation on this task finishes at time 10, there is a 20-time-unit range for *Hardware Producer Agent* to find a time in its local schedule to execute this task; if the negotiation finishes at time 20, *Hardware Producer Agent* only has 10-time-unit range to find a suitable time slot to execute this task. So the order of negotiation directly affects the outcome of the negotiation.

Meanwhile, in a multi-linked negotiation problem, there are several features that the agent needs to negotiate over for each subject. For a task proposal, the contractee agent needs to find the earliest start time and deadline to request for the task, how much reward to pay for this task, the early reward rate, and the

decommitment penalty, etc. The contractor agent needs to decide the promised finish time. Some of these features are related to the features of the subjects in other negotiations. For example, the deadline proposed for task *Get_Hardware_A* affects the earliest start time of task *Deliver_Computer_A*, and the deadline of task *Deliver_Computer_A* affects the promised finish time for task *Purchase_Computer_A*. The agent needs to find appropriate values for these features to avoid conflicts among them and to make sure there is a feasible local schedule to accommodate all the local tasks and commitments. Furthermore, the values of these features influence the outcomes of the negotiation and the agent's local utility. For example, the greater the reward is, the greater the likelihood that the task will be accepted by the contractor agent; however, the contractee agent's local utility decreases as the reward it pays to the contractor agent increases. Also, the later the deadline for task *Get_Hardware_A* is, the more likely that this task will be accepted by the *Hardware Producer Agent*; however, the consequence of a later deadline for task *Get_Hardware_A* is that there is less freedom for scheduling task *Deliver_Computer_A*, and the promised finish time for task *Purchase_Computer_A* is pushed back later, hence reducing the early reward that the *Computer Producer Agent* may get. A good negotiation strategy for a multi-linked negotiation problem should take an end-to-end perspective that accounts for all negotiations, and provides the agent with an appropriate order of all negotiations and a feature assignment (a set of assigned values) for those attributes under negotiation, so as to avoid the conflicts among negotiations and optimize utility.

3. Model of the problem

In this section, we first introduce a formalized model of the multi-linked negotiation problem and then present a heuristic search algorithm to find a near-optimal negotiation approach: a feature assignment and an order for a group of negotiations that an agent needs to conduct in order to optimize the expected utility.

3.1. Definition of the problem

A multi-linked negotiation problem occurs when an agent has multiple negotiations that are interrelated.

Definition 3.1. A **multi-linked negotiation problem** is defined as an undirected graph (more specifically, a forest as a set of rooted trees): $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, where $V = \{v\}$ is a finite set of negotiations, and $E = \{(u, v)\}$ is a set of binary relations on V. $(u, v) \in E$ denotes that negotiation u and negotiation v are directly-linked. The relationships among the negotiations are described by a forest, a set of rooted trees $\{T_i\}$. There is a relation operator associated with every non-leaf negotiation v (denoted as $\rho(v)$), which describes the relationship between negotiation v and its children. This relation operator has two possible values: AND and OR.

Figure 5 shows the model of the multi-linked negotiation problem (described in Figure 4) for *Computer Producer Agent*, the problem includes five negotiations. This

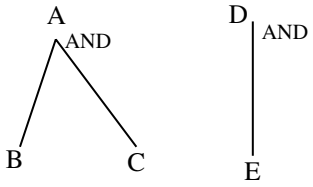

Figure 5. Interrelationships among negotiations.

model can also handle negotiating with multiple agents on one subject. For example, Figure 6 shows there are two transport agents: TAgent_1 and TAgent_2, both can be a potentially contractee for task *Deliver_Computer_A*. The negotiation with TAgent_1 and the negotiation with TAgent_2 can be modeled as C1 and C2 under C with a relation operator *OR*.

The subject in a negotiation v may be a task to be allocated or a resource to be acquired through negotiation.

From an agent's viewpoint, there are two types of negotiations:

1. **Incoming negotiation:** The negotiation about a task proposed by another agent, or a resource requested by another agent. For example, negotiation *A* (*Purchase_Computer_A*) and *D* (*Purchase_Computer_B*) in Figure 4 are incoming negotiations for *Computer Producer Agent*.
2. **Outgoing negotiation**: The negotiation about a task that needs to be sub-contracted to another agent, or a resource requested for a local task. For example, issue *B* (*Get_Hardware_A*), *C* (*Deliver_Computer_A*) and *E* (*Get_Hardware_B*) in Figure 4 are outgoing negotiations for *Computer Producer Agent*.

Definition 3.2. A negotiation v is successful if and only if a commitment has been established and confirmed for the subject in this negotiation by those agents which are involved in this negotiation.

Definition 3.3. A leaf node v is task-level successful if and only if v is successful; A non-leaf node v is task-level successful if and only if the following conditions are fulfilled:

Figure 6. Negotiation with multiple agents on one subject.

- v is successful;
- all its children are task-level successful if $\rho(v) = AND$; or at least one of its children is task-level successful, if $\rho(v) = OR$.

As in Figure 5, negotiation A is task-level successful if and only if negotiation A is successful, and negotiations B and C are also successful. In this case, *Computer Producer Agent* can actually perform task *Purchase_Computer_A* successfully.

Each negotiation $v_i (v_i \in V)$ is associated with a set of attributes $A_i = \{a_{ij}\}$. Each attribute a_{ij} either has already been determined or needs to be decided. There are two types of attributes: the *attributes-in-negotiation* (the features (attributes) of the subject to be negotiated, such as task deadline, reward (price), quantity, etc.), and the *attributes-of-negotiation itself* (i.e., negotiation start time, negotiation deadline, etc.). The attributes in negotiation are in general domain dependent. In this supply chain example, the following attributes (this is a complete and formal presentation compared to those mentioned in Section 2.1) need to be considered:

1. time range $(st(v_i), dl(v_i))$: the time range associated with a task contains the start time $(st(v_i))$ and the deadline $(dl(v_i))$. The task can only be performed during this range $(st(v_i), dl(v_i))$ to produce a valid result.
2. duration $(d(v_i))$: the process time requested to accomplish this task.
3. flexibility $(f(v_i))$: the flexibility is defined based on the time range and the duration: $f(v_i) = \frac{(dl(v_i) - st(v_i) - d(v_i))}{d(v_i)}$. The flexibility is an important feature because it directly affects the success probability of the negotiation (see detail in Section 5).
4. finish time $(ft(v_i))$: the promised finish time for the task.
5. regular reward $(r(v_i))$: if the contractee agent can finish the task by the deadline $dl(v_i)$, it gets reward $r(v_i)$.
6. early reward rate $(e(v_i))$: if the contractee agent can finish the task earlier than the deadline $dl(v_i)$, it gets extra reward $e(v_i) * (dl(v_i) - ft(v_i))$.
7. decommitment penalty$(\beta(v_i))$: the penalty paid to the other agent which is involved in negotiation v_i, when the agent decommits after v_i is successful.
8. task-level successful reward $(\gamma(v_i))$: the agent's utility increases by the amount of $\gamma(v_i)$ when v_i is a root of a tree and is task-level successful. It is calculated by subtracting the cost of v_i, including the local cost and sub-contracting cost (the reward paid to other agents), from the total reward of v_i (regular reward plus early reward).

The attributes-of-negotiation itself describes the negotiation process, they are domain in-dependent:

1. negotiation duration $(\delta(v_i))$: the time needed for negotiation v_i either to successfully complete or fail. It is assumed that negotiation duration is part of the agent's knowledge.[12]
2. negotiation start time $(\alpha(v_i))$: the start time of negotiation v_i. $\alpha(v_i)$ is an attribute that needs to be decided by the agent.
3. negotiation deadline $(\epsilon(v_i))$: negotiation v_i needs to be finished before this deadline $\epsilon(v_i)$. The negotiation is no longer valid after time $\epsilon(v_i)$, which is the same as a

failure outcome of this negotiation. For example, if task v_i is proposed for negotiation, the contractee agent needs to reply before time $\epsilon(v_i)$. Otherwise, this task proposal is no longer valid and the contractee agent would think the contractor agent is not interested in this task. Furthermore, even if the agent starts the negotiation before $\epsilon(v_i)$, it is not necessarily true that all times before $\epsilon(v_i)$ are equally good. Usually, a negotiation that is started earlier has a better chance to succeed for two reasons: the other party considers this issue before other later arriving issues, and this issue has a larger time range for negotiation. This relationship is described by the function ζ_i that takes $\alpha(v_i)$ as one of its parameters.

4. success probability ($p_s(v_i)$): the probability that v_i is successful. It depends on a set of attributes, including both attributes-in-negotiation (i.e., reward, flexibility, etc.) and attributes-of-negotiation (i.e., negotiation start time, negotiation deadline, etc.). How these attributes affect the success probability can be described as a function ζ_i (an example of this function is introduced in Section 5), which maps the values of the attribute a_{ij}, $j = 1, 2, ..., k$, to $p_s(v_i)$: $p_s(v_i) = \zeta_i(a_{i1}, a_{i2}, ..., a_{ik})$. a_{ij} $(j = 1, ..., k)$ represent the attributes that affect the success probability of this negotiation. This function is domain dependent, the agent can construct this function through the following approaches. One approach is that for the agents to communicate meta-level information before negotiation, such as the slack time in the agent's schedule, the number of other competitors, etc. This information could be used by the agent to construct the function more accurately. Another approach is for an agent to learn to construct and adjust the structure of this function based on its previous negotiation experience, provide that the similar negotiation situations are encountered multiple times. Reinforcement learning is a suitable technique for this problem.

The attributes above are similar to those used in project management [7], however, the multi-linked negotiation problem cannot be reduced to a project management problem or a scheduling problem. As Figure 7 shows, the multi-linked negotiation problem includes two sets of interrelated objects, the set of negotiations (shown in the upper box) and the subjects in these negotiations (shown in the lower box). The negotiations are interrelated and the subjects are interrelated, also the attributes of negotiations and the attributes of the subjects are interrelated too. The links among those attributes show the interrelationships among these attributes. For example, the negotiation start time and the negotiation deadline affect the success probability, the time range, the regular reward, and the earlier reward rate also affect the success probability. To solve a multi-linked negotiation problem, an agent needs to find a negotiation solution that includes the ordering of these negotiations (negotiation ordering) and appropriate values assigned to those attributes-in-negotiation (feature assignment). The goal is to find a negotiation solution that optimizes the agent's expected utility in these negotiations. The success probabilities, the task level success rewards and the decommitment penalties all contribute to the evaluation of a negotiation solution. The negotiation ordering determines the negotiation start time and/or the negotiation deadline of each negotiation, this ordering process can be viewed as a scheduling process of these negotiations. Part of the feature assignment process is to find

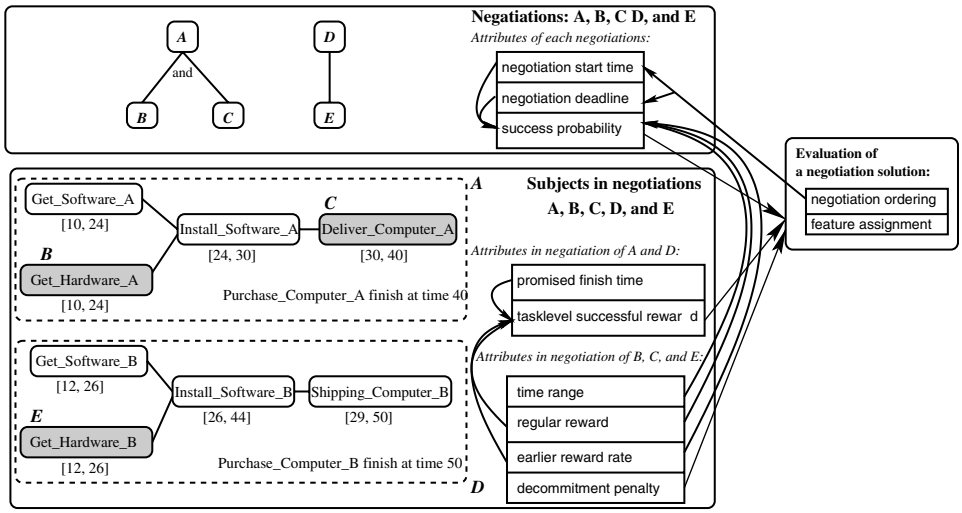

Figure 7. The structure of multi-linked negotiation problem.

consistent time ranges for those subjects in negotiations, which is another scheduling-like process. However, the whole multi-linked problem is not a classic scheduling problem given these two sets of interrelated objects. These extra dimensional complexities and interrelationships distinguish it from the classic project management/scheduling problem, where there is only one set of interrelated objects that need to be arranged in order.

3.2. Description of the solution

Given this multi-linked negotiation problem $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, an agent needs to make a decision about how the negotiations should be performed. The decision concerns the negotiation ordering and the feature assignment, and they are interrelated. The values assigned to some attributes, such as reward and flexibility, will affect the probability of the success of the negotiation, and hence will affect the ordering of the negotiations.

Definition 3.4. A **negotiation ordering** ϕ is a directed acyclic graph (DAG), $\phi = (V, E_\phi)$. If $e : (v_i, v_j) \in E_\phi$, then negotiation v_j can only start after negotiation v_i has been completed. $e : (v_i, v_j)$ is referred as a **partial order relationship (POR)**, e. A negotiation ordering can be represented as a set of **PORs**, $\{e\}$.

Definition 3.5. A negotiation schedule $\mathcal{NS}(\phi)$ contains a set of negotiations $\{v_i\}$. Each negotiation v_i has its negotiation start time $\alpha(v_i)_\phi$ and its negotiation finish time $\varepsilon(v_i)_\phi$ that is calculated based on its negotiation duration $\delta(v_i)$ and its negotiation start time $\alpha(v_i)_\phi$.

Using the topological sorting algorithm, a negotiation schedule $\mathcal{NS}(\phi)$ can be generated from a negotiation ordering ϕ assuming all negotiations started at their

earliest possible times.[13] Given this assumption and a start time τ[14] for a set of negotiations, the negotiation schedule generated from a negotiation ordering is unique.

As shown in Figure 8, suppose the negotiation start time $\tau = 0$, and the negotiation duration of each negotiation is the same $\delta(v_i) = 5$, then the following negotiation schedule is generated for negotiation ordering #3 in Figure 8 according to the assumption that every negotiation starts at its earliest possible time:

$$A[0, 5]B[5, 10]C[5, 10]D[0, 5]E[5, 10]$$

$A[0, 5]$ means that negotiation A starts at time 0 and finishes at time 5.

Definition 3.6. Given a start time τ, a negotiation ordering ϕ is **valid** if for every negotiation issue v_i, the finish time $\varepsilon(v_i)_\phi$ is no later than the negotiation deadline $\epsilon(v_i)$.

Definition 3.7. A feature assignment φ is a mapping function that assigns a value μ_{ij} to each attribute a_{ij} that needs to be decided in the negotiation. A feature assignment φ is **valid** if the assigned values of those attributes are consistent with each other.

"Consistent" is interpreted differently for different features. For time-related features, "consistent" means that given the assigned values of those time constraints, there exists at least one feasible local schedule for all tasks. The partial order scheduler and a related toolkit presented in Section 4 are used to test if the assigned values of the time-related features are consistent. For monetary features such as reward or price, "consistent" means that the sum of the sub-contracting cost paid to other agents is less than the total expected reward. Algorithm A.1 in the Appendix handles the consistent check for all types of features.

Definition 3.8. A negotiation solution (ϕ, φ) is a combination of a negotiation ordering ϕ and a valid feature assignment φ.

The evaluation of a negotiation solution is based on the expected task-level successful rewards and decommitment penalties given all possible negotiation outcomes for each negotiation. A negotiation has two possible outcomes: success and failure.

Definition 3.9. A negotiation outcome χ for a set of negotiations $\{v_j\}, (j = 1, \ldots, n)$ is a set of numbers $\{o_j\}(j = 1, \ldots, n), o_j \in \{0, 1\}$. $o_j = 1$ means v_j is successful, $o_j = 0$ means v_j fails. There are a total of 2^n different outcomes for n negotiations, denoted as $\chi_1, \chi_2, \ldots, \chi_{2^n}$.

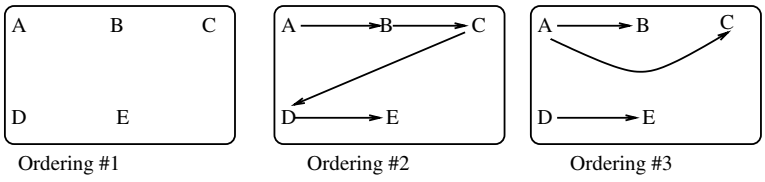

Figure 8. Three possible negotiation orderings.

Definition 3.10. The expected value of a negotiation solution (ϕ, φ), denoted as $\mathcal{EV}(\phi, \varphi)$, is defined as:

$$\mathcal{EV}(\phi, \varphi) = \sum_{i=1}^{2^n} P(\chi_i, \varphi) * (R(\chi_i, \varphi) + C(\chi_i, \phi, \varphi))$$

$P(\chi_i, \varphi)$ denotes the probability of the outcome χ_i given the feature assignment φ.

$$P(\chi_i, \varphi) = \prod_{j=1}^{n} p_{ij}(\varphi)$$

$$p_{ij}(\varphi) = \begin{cases} p_s(v_j), (p_s(v_j) = \zeta_j(\varphi)) & \text{if } o_j \in \chi_i = 1 \\ 1 - p_s(v_j) & \text{if } o_j \in \chi_i = 0 \end{cases}$$

$R(\chi_i, \varphi)$ denotes the agent's utility increase given the outcome χ_i and the feature assignment φ. $R(\chi_i, \varphi) = \sum_j \gamma_\varphi(v_j)$, v_j is a root of a tree and v_j is task-level successful according to the outcome χ_i. $C(\chi_i, \phi, \varphi)$ denotes the decommitment penalty ($C(\chi_i, \phi, \varphi) <= 0$) according to the outcome χ_i, the negotiation ordering ϕ and the feature assignment φ. $C(\chi_i, \phi, \varphi)$ is the sum of the decommitment penalties of those negotiations, which are successful, but their root nodes are not task-level successful, and such situations are unknown before these negotiation are started. $C(\chi_i, \phi, \varphi) = \sum_j \beta_\varphi(v_j)$, v_j represents every negotiation that fulfills all the following conditions:

1. v_j is successful according to χ_i;
2. the root of the tree that v_j belongs to isn't task-level successful according to χ_i;
3. according to the negotiation ordering ϕ, there is no such negotiation v_k existing that fulfills all the following conditions:
   (a) v_k and v_j belong to the same tree;
   (b) v_k gets a failure outcome according to the outcome χ_i;
   (c) v_k makes it impossible for $root(v_j)$ to be task-level successful;
   (d) the negotiation finish time of v_k is no later than the negotiation start time of v_j according to the negotiation ordering ϕ.

3.3. Description of a heuristic search algorithm

Based on the above definition, we present an algorithm that find a nearly optimal (as we show in the experimental results) negotiation solution for a multi-linked negotiation problem $\mathcal{M} = (\mathcal{V}, \mathcal{E})$.

Given a multi-linked negotiation problem $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, the start time for negotiation τ, a set of valid feature assignments $\omega = \{\varphi_k\}$, $k = 1, \ldots, m$, the complete search algorithm evaluates each pair of negotiation ordering and valid feature assignment (ϕ_i, φ_k), and then return the best one.[15] The exponential complexity of this complete algorithm prevents it from being used for real-time applications when the number of negotiations and the number of valid feature assignments are large; hence a heuristic search algorithm has been developed.

The heuristic search for the near-optimal negotiation solution is broken into two parts. One is to find a near-optimal negotiation schedule; the other one is to find a near-optimal feature assignment for a given negotiation schedule. The search for the optimal negotiation schedule is based on a simulated annealing search. Given a negotiation ordering ϕ, randomly pick a POR e, if $e \in E_\phi$, remove it from E_ϕ; otherwise add it into E_ϕ.[16] A new negotiation ordering ϕ_{new} is now generated. If the negotiation schedule $\mathcal{NS}(\phi_{new})$ is better than $\mathcal{NS}(\phi)$, move to ϕ_{new}; otherwise, move to ϕ_{new} with some probability less than 1. This probability decreases exponentially with the "badness" of this move. Three heuristics have been added to this simulated annealing process:

1. Record the best negotiation schedule so far found. When the search process ends, return the best negotiation schedule ever found rather than the current one.
2. Instead of randomly deciding whether to add a POR or remove a POR, use a parameter (*add_por_probability*) to control the probability of the operation "add" or "remove". Actually, this parameter controls the tradeoff between sequencing versus parallelizing the negotiation schedule (adding a POR forces two negotiations to be serialized).
3. Instead of completely randomly choosing a POR to change from current negotiation ordering, evaluate every POR e according to how the value of the negotiation schedule changes by adding this POR e to an empty POR set. The probability of adding POR e to the current POR set or removing POR e from the current POR set depends on this evaluation. A POR e with a higher positive evaluation has a higher probability of being added, and has a lower probability of being removed.

Consider an example with three negotiations A, B and C. Suppose the negotiation start time $\tau = 0$, and the negotiation duration of each negotiation is the same $\delta(v_i) = 5$, the evaluation of POR $(A \rightarrow B)$ is calculated as: the value of the negotiation schedule: $A[0, 5]B[5, 10]C[0, 5]$ minus the value of the negotiation schedule: $A[0, 5]B[0, 5]C[0, 5]$.

The search for the near-optimal feature assignment is based on a hill climbing search. Randomly pick another feature assignment φ_k. If it is better than current one, move to φ_k. After considering the characteristics of this problem, the following heuristics have been added to this search process:

1. According to the generation process, the change of those valid feature assignments is continuous. Based on this observation, a number of sample points with equal distance (the distance is adjustable, denoted as *sample_step*) in between can be selected from all the valid feature assignments and evaluated. Hill climbing search then can be performed for each sample point.
2. Given current chosen feature assignment, the possible operations include: moving to left and moving to right. If there is a better selection than current one, move to the better selection; otherwise the search stops and a local maxima is found.
3. Compare all local maxima and return the best one.

Both search algorithms are implemented with search limitation threshold: after certain amount of search effort, the algorithm will stop and report the result. Experiments were performed to test how well these combined heuristic algorithms work, and as we will describe in 6.1, the experimental work shows that the heuristic search algorithm finds solutions very close to the best solutions found by the complete search algorithm with significantly less effort.

4. Partial order schedule and related algorithms

In this section, we will introduce a partial order scheduler which allows the agent to reason about the time-related constraints and the flexibility associated with each negotiation issue. This toolkit is used by the agent to find valid feature assignments, which are part of the input for the heuristic search algorithm described in Section 3.3.

4.1. Partial order schedule

A partial-order schedule is the basic reasoning tool that we use for interrelated negotiations. Here we present the formalization of the partial-order schedule and use examples to explain how it works for a multi-linked negotiation. Figure 9 shows the partial-ordered schedule generated for the example in Figure 4.

A *partial order schedule*[17] represents a group of tasks with specified precedence relationships among them using a directed acyclic graph: PS = (T,R). $T = \{t | t \text{ is a task}\}$, where each vertex in T represents a task, and $R = \{(s,t) | s,t \in T)\}$, where each edge (s,t) in R denotes the precedence relationship between task s and task t ($P(s,t)$), which means that task s has to be finished before task t can be started.

A *Task* is represented as a node in the graph; it is the basic element of the schedule. A task t needs a certain amount of processing time, also referred as its duration ($t.process\_time$). A task can be a local task or a non-local task; a local task is per-

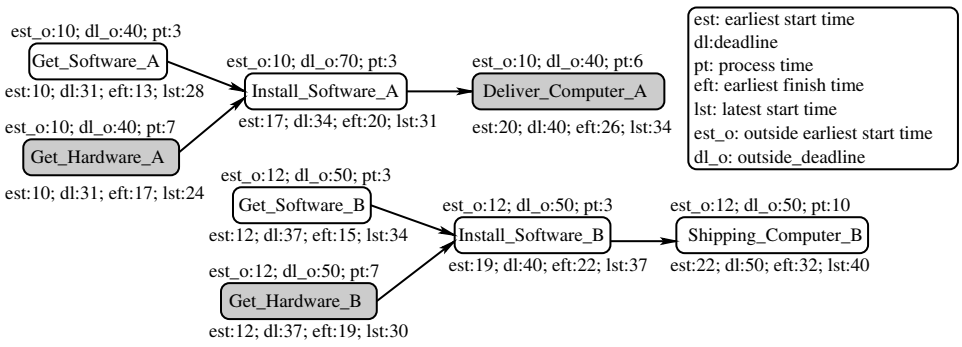

Figure 9. The partial order schedule of *Computer Producer Agent*.

formed locally (i.e., task *Get_Software_A* and task *Shipping_Computer_B*) and a non-local task (i.e., task *Get_Hardware _A* and task *Deliver_Computer_A*) is performed by another agent; hence, it does not consume local process time. The *pretasks of task t* is a set of tasks that need to be finished before t can start: $Pre(\mathrm{t}) = \{s | s \in \mathrm{T}, (s,t) \in R\}$; t can start only after all tasks in $Pre(t)$ have been finished. For example, the pretasks of task *Install_Software_A* includes task *Get_Hardware _A* and task *Get_Software_A*. The *posttasks of task t* is a set of tasks that only can start after t has been finished: $Post(t) = \{r | r \in \mathrm{T}, < t, r > \in R\}$. For example, the posttasks of task *Install_Software_B* includes task *Shipping _Computer _B*.

A task t has constraints of *earliest start time* ($t.est$) and *deadline* ($t.dl$). The *earliest start time* of a task t ($t.est$) is determined by the *earliest finish time* of its pretasks ($eft[Pre(t)]$) and its *outside earliest start time* constraint ($t.est\_o$)[18]:

$$t.est = max(eft[Pre(t)], t.est\_o).$$

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 T ($eft[T]$) is defined as the earliest possible time to finish every task in the set T; it depends on the *earliest start time* and the duration of each task. For example, in Figure 9, the *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 pretasks is 17 (assume *Get_Hardware_A* could be finished at its earliest possible time), then the *earliest start time* for task *Install_Software_A* is 17.

The *deadline* of task t ($t.dl$) is determined by the *latest start time* of its posttasks ($lst[Post(t)]$) 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 T ($lst[T]$) 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, $F(Get\_Software\_A) = (40 - 10 - 3)/3 = 9$.

A *feasible linear schedule* is a total ordered schedule of all tasks, that fulfills the following conditions:

- Each task t takes n ($n >= 1$, if t is interruptible; otherwise, $n = 1$.) time periods $(pt_i, i = 1, \ldots, n)$ for execution, $\sum_i pt_i = t.processtime$.
- All precedence relationships are valid.
- All earliest start time and deadline constraints are valid.

A partial-order schedule is a *valid* if and only if there exists at least one feasible linear schedule that can be produced from this partial order schedule without additional constraints and with the interruptible execution assumption.[19]

Without additional constraints and with the interruptible execution assumption, for a task t with the range [est, dl], no matter when task t is executed during this range, if there exists at least one feasible linear schedule that can be produced from this partial schedule, then the range [est, dl] for task t is a *free range* because task t can be executed during any period in this range.

Without additional constraints and with the interruptible execution assumption, for a set of tasks $t_i(i = 1, 2, \ldots, n)$, with the range $[est_i, dl_i](i = 1, 2, \ldots, n)$, respectively, no matter what time t_i is executed during the range $[est_i, dl_i]$, if there exists at least one feasible linear schedule that can be produced from this partial schedule, then the ranges $[est_i, dl_i](i = 1, 2, \ldots, n)$ for tasks $t_i(i = 1, 2, \ldots, n)$ are *consistent ranges*. Negotiation over tasks $t_i(i = 1, 2, \ldots, n)$ can be performed in parallel using these consistent ranges without worrying about conflicts. Figure 10 shows the consistent ranges for the tasks in the supply chain example. This means, the negotiation for task Get_Hardware_A, Get_Hardware_B, and Deliver_Computer_A can be performed in parallel using the time range [10, 24], [12, 26] and [30, 40]. Figure 11 presents a feasible linear schedule given these consistent ranges. The two numbers in a box below a task represent the consistent range for this task, and the two numbers above a task indicate the start time and the finish time for this task in one linear schedule. It should be noticed that for each task the start time

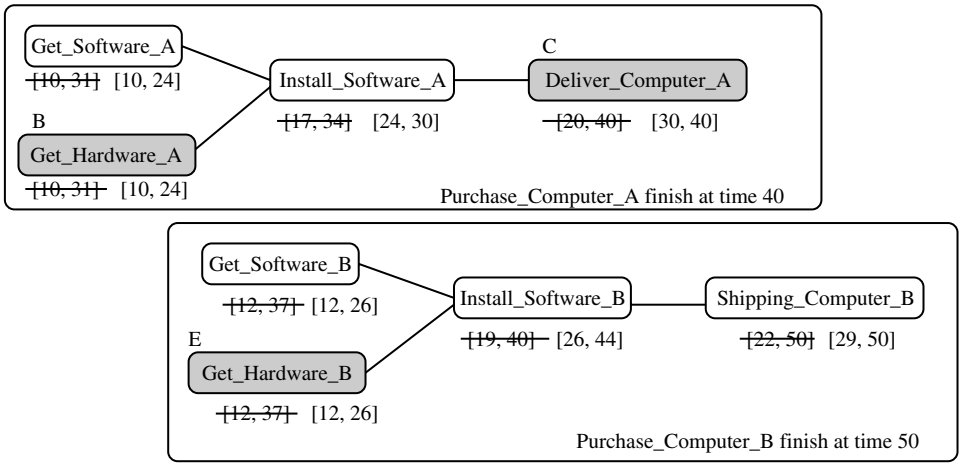

Figure 10. The consistent ranges for tasks in negotiation: Get_Hardware_A, Get_Hardware_B, and Deliver_Computer_A.

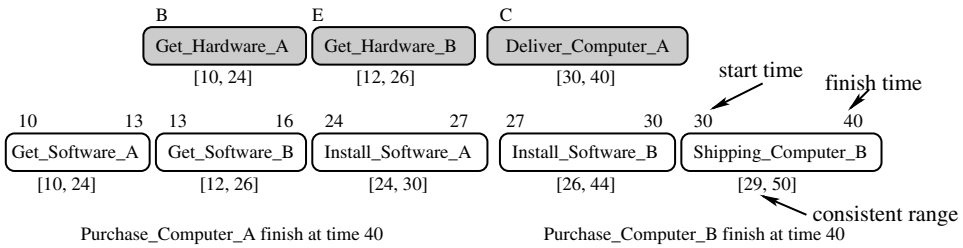

Figure 11. The feasible linear schedule for those tasks in Figure 10.

and the finish time fall into its consistent range, they also can be moved freely during this range.

The partial order schedule work is related to the Graphical Evaluation and Review Technique(GERT) [8] which is used for project scheduling and management. The major difference between the GERT work and ours is that the GERT work is not oriented to negotiation; all activities are local and can be managed with authority. Thus, with GERT there is no reasoning about free range, consistent ranges and schedule flexibility that we feel are critical for an agent to effectively manage multi-linked negotiation. Without reasoning of these factors, it is difficult to negotiate efficiently on multiple related issues.

4.2. Algorithms

We have built the following algorithms to support the negotiation based on the partial order schedule. We only describe the functions of these algorithms, the detailed processes are presented in [16]. The complexities of these algorithms are provided accordingly, n represents the number of input tasks.

Algorithm 4.1. Propagate_EST_DL (Complexity: $O(n^2)$).
Given a set of tasks with the outside constraints of the earliest start times and deadlines, durations and precedence relationships, this procedure finds the earliest start time ($t.est$) and the deadline ($t.dl$) for each task t according to the definitions in Section 4.1.

Algorithm 4.2. Get_Earliest_Finish_Time (Complexity: $O(n^2)$)
Given a set of tasks T, each task t has earliest start time ($t.est$) and its duration ($t.process\_time$), this procedure calculates the earliest finish time of a set of tasks T ($eft[T]$).

Algorithm 4.3. Get_Latest_Start_Time (Complexity: $O(n^2)$)
Given a set of tasks T, each task t has its *deadline* ($t.dl$) and its duration ($t.process\_time$), this procedure calculates the latest start time of a set tasks T ($lst[T]$).

Algorithm 4.4. Feasible_Schedule (Complexity: $O(n^4)$)
Given a partial order schedule (T, R), each task has its earliest start time and duration with respect to its pretask, posttask and its outside constraints, this procedure generates a feasible linear schedule if the partial order schedule is valid; otherwise it reports failure.

Theorem 4.1. *If there exists a feasible linear schedule, the Feasible_Schedule algorithm can find one.*
The proof of this theorem is presented in [16].

Besides Algorithm 4.4, we have also developed Algorithm 4.5 to answer the question of whether a partial order schedule is valid without trying to find a feasible linear schedule.

Algorithm 4.5. Range_Evaluation (Complexity: $O(n^2)$)
This procedure determines if a partial order schedule is valid.

The basic idea of Algorithm 4.5 is to check every possible time range $[est, dl]$ by constructing all possible combinations of every task's earliest start time and deadline. For all tasks falling into this range, if the sum of process times of these tasks is greater than the time available $(dl - est)$, there is no feasible linear schedule; otherwise, there exists a feasible linear schedule, because every task t can find a place between its earliest start time and its *deadline*.

This proves the following theorem:

Theorem 4.2. *A partial order schedule is valid if and only if the procedure 4.5 returns true.*

Using the above procedure, we have constructed the following algorithm to find the free range of a non-local task used for the negotiation.

Algorithm 4.6. Find_NL_Range (Complexity: $O(n^2)$).
Given a partial order schedule (T, R) containing a task *nlt*, this procedure finds the largest free range for task *nlt*.

If there is more than one non-local task, we need to sort them according to some characteristics (i.e., flexibility, importance, difficulty of negotiation, etc.), and work on them one by one. When the Find_NL_range procedure works on one task *nlt_i*, the range for those tasks before it ($nlt\_1, \ldots, nlt\_(i-1)$) has already been decided and cannot be changed. The range for those tasks after it ($nlt\_(i+1), \ldots$) are set to a range that is as small as possible, so as to allow this task *nlt_i* to have the most freedom.

All of the above algorithms and procedures provide a toolkit for the agent to reason about its proposals and evaluate counter-proposals from other agents.

5. Example

In this section, we demonstrate how the definition and the algorithm work on the supply chain examples in Figure 4.

To make the output easier to understand, only negotiation A (Purchase_Computer_A), B (Get_Hardware_A) and C (Deliver_Computer_A) are considered in the following example. For incoming negotiation A, regular reward $r(A) = 19$, the attribute that needed to be decided is the promised finish time ft; the task-level successful reward depends on the promised finish time ft:

$$\gamma(v) = r(v) + e(v) * (dl(v) - ft(v)).$$

For outgoing negotiation B and C, the attributes needed to be decided are the start time (st) and the deadline (dl). It is assumed that the negotiation durations are already known to the agent, $\delta(A) = 3$, $\delta(B) = 4$, $\delta(C) = 4$. The negotiation start times need to be decide by the agent as part of the problem of constructing a negotiation ordering. It is also assumed that the success probability depends on the flexibility $f(v)$, which is calculated based on the time range $(st(v), dl(v))$ and the process time $d(v)$

$$\left(f(v) = \frac{dl(v) - st(v) - d(v)}{d(v)}\right).$$
$$p_s(v) = p_{bs}(v) * (2/\pi) * (\arctan(f(v) + c)))$$

p_{bs} is the **basic success probability** of this negotiation v when the flexibility $f(v)$ is very large. c is a constant parameter used to adjust the relationship. In this example, the following functions are used to determine the success probabilities for B and C:

$$p_s(B) = p_{bs}(B) * (2/\pi) * (\arctan(f(B) + 2.5));$$
$$p_s(C) = p_{bs}(C) * (2/\pi) * (\arctan(f(C) + 5));$$
$$p_{bs}(B) = 0.95, p_{bs}(C) = 0.99.$$

The different constant parameters for $p_s(B)$ (2.5) and $p_s(C)$ (5) specify that issue C has a higher *success probability* than issue B given the same flexibility, as shown in Figure 12. The following parameters are randomly generated: the success probability of A, the negotiation deadline, the early reward rate of A, and the decommitment penalty.

For every attribute that needs to be decided: start time (st), deadline (dl) and the promised finish time (ft), the agent can find its maximum possible range using the partial-order schedule as shown in Figure 13. The agent searches over the entire possible value space (Appendix, Algorithm A.1), and use the partial-order schedule to test if a feature assignment is valid. A set of valid feature assignments is found and used to find the optimal negotiation solution combining ordering constraints and feature assignment.

Table 1 shows the output of the complete search algorithm (see Appendix, Algorithm A.3) on six different cases in Figure 5, based on different negotiation deadlines, early reward rates and decommitment penalties. In both cases 1 and 2, the negotiation deadline $\epsilon = 6$ is used, which results in a negotiation ordering that has the three negotiations performed in parallel. In case 2, A has a higher earlier reward rate $e(A)$, and all negotiations have lower decommitment penalties β than in case 1, so the negotiation solution in case 2 arranges task A to finish 21 time units earlier than the requested deadline, and earns an extra reward of 4.0. In exchange, B and C have smaller

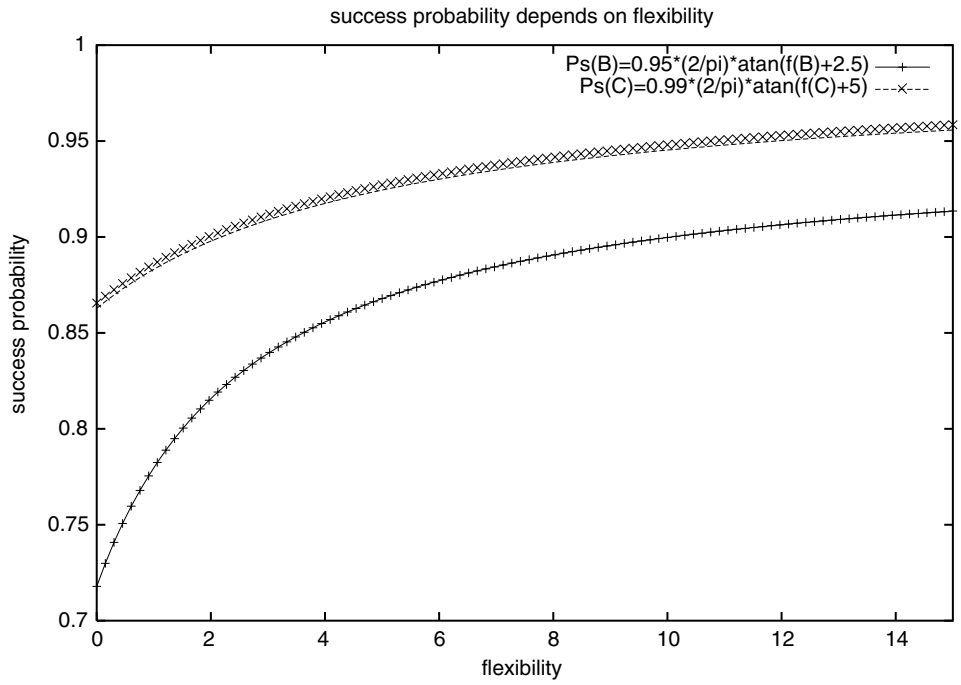

Figure 12. Success probability depends on flexibility.

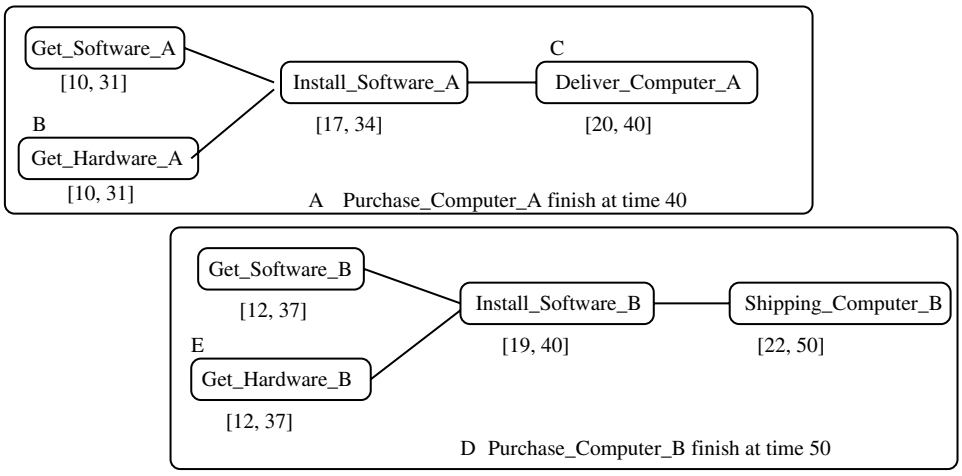

Figure 13. Partial-order schedule.

flexibilities $f(B)$ and $f(C)$, hence lower success probability $p_s(B)$ and $p_s(C)$. In cases 3 and 4, the negotiation deadline $\epsilon = 9$. In case 3, A has a much lower success probability $p_s(A)$ than in case 4, so negotiation A is scheduled before negotiation B and C. In cases 5

Table 1. Examples of optimal negotiation solutions.

Negotiation deadline (ϵ)	v	Early reward rate ($e(A)$)	Decommit penalty (β)	Negotiation schedule	$et = dl - ft$	Early reward $(er) = e(A) * (dl - ft)$	Flexibility $f(v)$	Success probability $p_s(v)$
#1	A	0.012	22.2	A[0–3]	0	0		0.9
$\epsilon = 6$	B		1.32	B[0–4]			3.0	0.84
	C		1.32	C[0–4]			0.83	0.88
#2	A	0.189	1.95	A[0–3]	21	4.0		0.92
$\epsilon = 6$	B		0.12	B[0–4]			1.0	0.78
	C		0.12	C[0–4]			0.5	0.88
#3	A	0.117	16.6	A[0–3]	0	0		0.19
$\epsilon = 9$	B		0.991	B[3–7]			3.0	0.84
	C		0.99	C[3–7]			0.67	0.88
#4	A	0.006	16.6	A[4–7]	0	0		0.64
$\epsilon = 9$	B		0.99	B[0–4]			2.43	0.83
	C		0.99	C[0–4]			0.67	0.89
#5	A	0.043	17.7	A[0–3]	0	0		0.15
$\epsilon = 11$	B		1.06	B[3–7]			2.43	0.83
	C		1.06	C[7–11]			0.83	0.88
#6	A	0.142	12.6	A[8–11]	9	1.3		0.84
$\epsilon = 11$	B		0.75	B[0–4]			1.43	0.80
	C		0.75	C[4–8]			1.0	0.89

and 6, the negotiation deadline $\epsilon = 11$ and negotiation A, B and C are sequenced according to the success probabilities; the negotiation with the lower success probability starts earlier. In case 6, A has a higher earlier reward rate $e(A)$, and all negotiations have lower decommitment penalties β than case 5, so the negotiation solution in case 6 arranges task A to finish 9 time units earlier than the requested deadline; this earns an extra of reward 1.3. In exchange, B and C have smaller flexibilities $f(B)$ and $f(C)$ and hence lower success probabilities $p_s(B)$ and $p_s(C)$. It is also important to notice that in all cases, B gets larger flexibility than C, but has a similar success probability to that of C. This occurs because it is much easier for C to achieve a successful negotiation according to the function that defines the relationship between the success probability and the flexibility. This result demonstrates that this type of reasoning is possible given the formal model described in Section 3.

6. Experimental work

We have implemented all the algorithms and reasoning tools described in previous sections. To evaluate how these mechanisms work, we have built those agents that described in the supply chain scenario (Section 5). These agents are implemented using JAF (Java Agent Framework)[4], which provides the basic functions such as communication and execution, for the agent, so we can focus on building the negotiation component. The experiments are performed in the multi-agent system simulator (MASS) [4], which provides a concrete, re-runnable, well-defined environment to test multi-agent negotiation. We designed and performed three sets of experiments for different purposes as described below.

6.1. Performance of heuristic algorithm

The first purpose is to test how well the heuristic algorithm works compared to the complete search algorithm. The experimental setting is based on the example described in Section 5. New tasks were randomly generated with decommitment penalty $\beta \in [0, 25]$, early finish reward rate $e \in [0, 0.2]$, and deadline $dl \in [60, 70]$, and arrived at the contractee agents periodically. We use the same task structures as described in Figure 4, tasks vary with randomly generated parameters. This scenario represents a class of problems where one agent needs to deal with both directly related and indirectly related negotiation problems. The deadlines of tasks are randomly generated from a range, which allows the agent to choose different negotiation orderings. The following values (see Algorithm A.4 for more details) were used in these experiments: $add\_por\_probability = 0.55$, $TEMP\_MAX = 5$; $TEMP\_STEP = 0.1$; $sample\_step = 10$, $search\_limit = 10^6$.

Table 2 shows the performance of this heuristic search algorithm compared to the complete search algorithm. The quality of the negotiation solution found is very close to the best solution found by the complete search. This heuristic algorithm saves a large amount of search effort compared to the complete search when the number of negotiations and the number of possible feature assignments increase. The heuristic search spends more effort than the complete search when the search

Table 2. Performance of heuristic search algorithm (NN: Number of negotiations; NF: Number of valid feature assignments (the data points are grouped according to NN and NF); Quality: the quality of the approach found by the heuristic search compared to the best approach found by the complete search (with quality normalized to 1.0); CS: the number of search steps of the complete search. HS: the number of search steps of the heuristic search; Ratio: the ratio of heuristic search steps to complete search steps. DS: Number of data samples).

NN	NF	Quality	CS	HS	Ratio	DS
3	[0, 50)	0.982	336	520	1.547	89
	[50, 100)	1.000	832	590	0.709	3
5	[0, 50)	1.000	1759	1967	1.119	48
	[50, 100)	0.998	3861	1766	0.457	6
6	[0, 50)	1.000	9353	1869	0.200	43
	[50, 100)	0.998	19502	1734	0.089	111
	[100, 150)	0.998	31086	1674	0.054	123
	[150, 200)	0.996	44058	1674	0.038	108
	[200, 250)	0.995	57253	1692	0.030	88
	[250, 300)	0.994	70292	1670	0.024	57
	[300, 350)	0.997	82736	1638	0.020	46
	[350, 400)	0.995	95213	1644	0.017	28
	[400, 450)	0.994	108185	1662	0.015	25
	[450, 500)	0.998	121479	1667	0.014	17

space is very small (with a few negotiations and a few of feature assignments). This problem can be fixed by choosing the values of the search parameters dynamically according to the size of current search space, instead of using the fixed values as we did in these experiments. For example, when the number of negotiations (NN) and the number of valid feature assignments (NF) is small, we can set the *search_limit* as a small number so that the search can stop earlier; because a good-enough solution can be found with less search effort in a small search space.

6.2. Different negotiation strategies

The second purpose is to test how different negotiation strategies affect the agent's performance under multi-linked negotiation situation. We compare the negotiation strategy generated from the reasoning based on the formalized model with some other simpler strategies. Under this experimental setup, Computer Producer Agent needs to deal with multi-linked negotiations related to the incoming task *Purchase_Computer* and the outgoing task *Get_Hardware* and *Deliver_Computer*. The following three different negotiation strategies were tested:

1. *Sequenced negotiation*. The agent deals with the negotiations one by one, first the outgoing negotiations, then the incoming negotiations. The finish time promised is the same as the deadline requested from the other agent, and the outgoing negotiations get the largest possible flexibilities.
2. *Parallel negotiation*. The agent deals with the negotiations in parallel. It arranges reasonable flexibility (1.5, in this experiment) for each outgoing task, and based

on this arrangement, the finish time of the incoming task is decided and promised to the contractee agent.

3. *Decision-based negotiation*. The agent deals with the negotiation as the best negotiation solution generated by the complete search algorithm.

The entire experiment contains 40 group experiments. Each group experiment has the system running for 1000 time clicks for three times and each time Computer Producer Agent uses one of the three different approaches. During 1000 time clicks, there are 60 new tasks received by Computer Producer Agent. Table 3 shows the comparison of Computer Producer Agent's performance using different strategies. When the agent uses the sequenced negotiation strategy, more tasks are canceled because of the missed negotiation deadlines. When the agent uses the parallel negotiation strategy, the agent pays a higher decommitment penalty because the failure of the sub-contracted task prevents the incoming task to be task-level successful. The decision-based approach is obviously better than the other two approaches.[21] It chooses a negotiation strategy dynamically according to negotiation deadlines and other attributes. Under this experimental setup, it chooses the case where all negotiations are performed in parallel about 13% of the time; it chooses the case where all negotiations are performed sequentially about 38% of the time, and the other times it chooses the case where some negotiations are performed in parallel. This strategy enables the agent to receive more early reward and pay fewer decommitment penalties.

The experimental result shows that in a multi-linked negotiation situation, it is very important for the agent to reason about the relationship among different negotiations and make a reasonable decision about how to perform negotiation. This decreases the likelihood of the need for decommitment from previously settled negotiations and increases the likelihood of utility gain.

6.3. *Experiments on flexibilities*

The third purpose is to study how the different flexibility policies in negotiation, which involve different types of reasoning strategies, affect the agent's performance. The experimental environment is set up based on the scenario described in Figure 14. It is a simplified scenario from the example shown in Section 2. This scenario represents a class of problem where one agent needs to deal with both directly and indirectly related negotiation problems. New tasks were randomly generated with

Table 3. Comparison of computer producer agent's performance using different negotiation strategies.

Policy	Task canceled	Decommit penalty	Early reward	Utility
Sequenced	37.25	73.82	0	358.09
Std.Dev.	2.6	11.8	0	57.4
Parallel	23.70	333.20	29.06	385.20
Std.Dev.	2.6	47.6	17.0	86.8
Decision-based	25.78	56.65	185.79	779.16
Std.Dev.	2.4	23.5	47.8	62.3

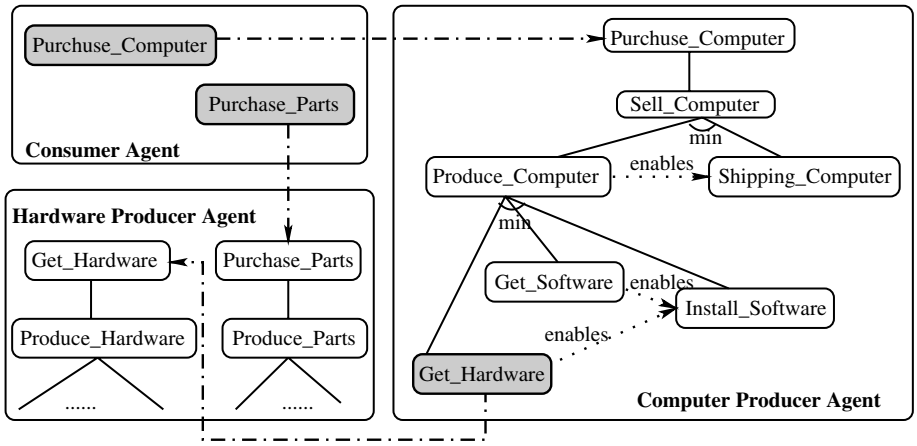

Figure 14. Three agents scenario.

decommitment penalty rate $p \in [0, 1]$, early finish reward rate $e \in [0, 0.1]$, and deadline $dl \in [45, 105]$ (this range allows different flexibilities available for those subcontracted tasks), and arrived at the contractor agents periodically. The local scheduler of the agent schedules all incoming new tasks occurring in a scheduling time window according to their earliest start times, deadlines, process times and rewards and generates an agenda (such as agenda 2.1 on p. 7). 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 non-local task *nlt*, then the Find_NL_Range procedure (Algorithm 4.6) 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. The entire experiment contains 32 group experiments. Every group experiment runs 3 times for 1000 time clicks each, each time using one of the three different polices (All agents use the same policy at the same time).

In this experiment, *Computer Producer Agent* needs to deal with the multi-linked negotiations related to the incoming task *Purchase_Computer* and the outgoing task *Get_Hardware*. The following three different negotiation policies 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 of maximizing the early finish reward. In the example of Section 2.2, *Computer Producer Agent* will accept both task *Purchase_Computer_A* and task *Purchase_Computer_B*, with the promised finish time 26 and 46, respectively, according to agenda 2.1 on page 7.
2. *Deadline Policy*: The agent promises the finish time that is the same as the *deadline* of the task with no consideration of the early finish reward. In the example of

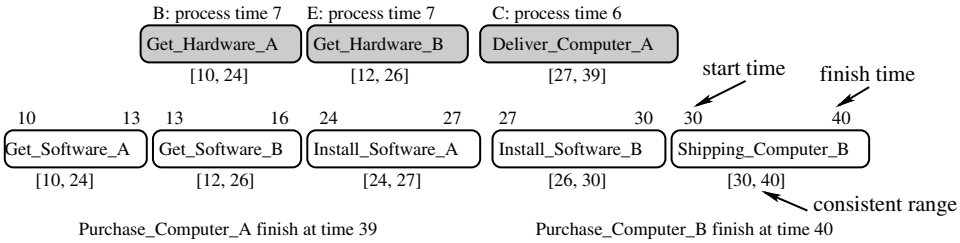

Figure 15. The feasible schedule with flexibility of 1 for each non-local task.

Section 2.2, *Computer Producer Agent* will accept both task *Purchase_Computer_A* and task *Purchase_Computer_B*, with the promised finish time 40 and 50, respectively, according to their deadline requests.

3. *Flexibility Policy*: the agent analyzes its detailed partial-order schedule. If non-local tasks are found, it arranges for reasonable flexibility (1, in this experiment) for each non-local task, and based on this arrangement, the finish time of the incoming task is decided and promised to the contractee agent. In the example of Section 2.2, *Computer Producer Agent* will accept both task *Purchase_Computer_A* and task *Purchase_Computer_B*. The promised finish time for task *Purchase_Computer_A* is 39, and the promised finish time for task *Purchase_Computer_B* is 40, according to the feasible schedule shown in Figure 15.

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

Table 4 shows the comparison of the agents' performance using different policies. For the *Computer Producer Agent* (CPA), who has multi- linked negotiations, the flexibility policy is obviously better than the other two policies; it gives the agent higher utility because it generates more early reward and it causes fewer decommitment penalties.[22] For the *Hardware Producer Agent* (HPA), the *Earliest Finish Time Policy* and the *Flexibility Policy* make no difference in the agent's decision making processes, since the agent has no sub-contracted task that needs consideration. The reason that

Table 4. Comparison of performance using different negotiation policies in multi-linked negotiation.

	Policy	Tasks received	Tasks accepted	Tasks canceled	Early finished	Decommit penalty	Early reward	Utility
CPA	Earliest finish time policy	60	59	27	33	123	283	391
CPA	Deadline policy	60	60	0.5	0	2.9	0	413
CPA	Flexibility policy	60	60	1.7	53	8.3	297	697
HPA	Earliest finish time policy	87	87	27	29	0	36	268
HPA	Deadline policy	84	84	9.6	0	0	0	256
HPA	Flexibility policy	87	87	11	17	0	32	294

CPA: *Computer Producer Agent*; HPA:*Hardware Producer Agent*.

the *Earliest Finish Time Policy* generates less utility for HPA is that because the CPA cancels more task requests (because the finish times that the HPA could provide are too late for CPA who also uses the *Earliest Finish Time Policy* at this time), and hence the HPA has fewer tasks to perform and gains less reward. Because the CPA is involved in the multi-linked negotiation, it pays lots of decommit penalties when it adopts the *Earliest Finish Time Policy* when it finds that the finish time it promised cannot be fulfilled. For the HPA, who does not need sub-contract task to other agents, the *Earliest Finish Time Policy* produces more utility than the *Deadline Policy* because it brings some early reward without paying any decommit penalty. These experiments shows that in a multi-linked negotiation situation, it is very important for the agent to reason about the relationships among different negotiations and maintain reasonable flexibility for them. This type of reasoning decreases the likelihood of decommitment from previously settled negotiations and thus gains more utility.

7. Related Work

To our knowledge, there is no other 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 negotiations such as the distributed meeting scheduling [12] problem and the distributed resource allocation problem [2]. However, those problems are different from our problem in the following ways: the negotiation is cooperative by nature and the agent can altruistically withdraw its request to help others succeed; the tasks are simple, no need for subcontracting; no time pressure on negotiation and no penalty for decommitment. The negotiation problem presented in this paper is much more complicated. Additionally, in these works, the agents do not explicitly reason about the relationships among different negotiations, in order to propose offers or counter-offers (choose the appropriate parameters in the offer) to minimize the conflict and optimize the combined outcome. The ordering of different negotiations is not taken into consideration in either of these approaches, which we feel is important for the agent to find a good negotiation approach. Sandholm [9] has developed a complex contract type – ''clustering- swap-multi-agent'' that allows tasks to be clustered, and then swapped between agents and even circulated among agents. This work deals with *indirectly linked* negotiations 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.

A combinatorial auction could be another approach to multi-linked negotiation problem, in which there are multiple items for sale, participants who may place bids on arbitrary subsets of those items, and an auctioneer who must determine which awardable combination of bids maximizes revenue. It allows agents to select a shared plan for the group through a distributed computation process [6]. It is also used to form a supply chain [15]. However, we do not feel that combinatorial auction is a panacea for this multi-linked negotiation problem or a better approach than the approach we described in this paper given the following reasons.

First of all, in combinatorial auction, the agent does not reason about the ordering of negotiations, since all items are announced at the same time, meaning all issues are

negotiated concurrently. However, this assumption does not fit with the directly linked negotiation situation. For instance, in this PCT example shown in Figure 16, the *Computer Producer Agent* receives a task proposal *Purchase_Computer* (A) from the *Consumer Agent*. To accomplish this task, the *Computer Producer Agent* needs to subcontract task *Get_Hardware* (B) and task *Deliver_Computer* (C). If we put this example into the combinatorial auction framework, we will find that there is no way that these three negotiation issues A, B, and C can be performed concurrently without conflict. Using the combinatorial auction model with time constraints [6], the *Computer Producer Agent* needs to first announce the two tasks *Get_Hardware* and *Deliver_Computer*, and wait for other agents to bid for these two tasks, and then select the combined bids with consistent time constrains and minimized cost. Based on these selected bids, the *Computer Producer Agent* can go back to negotiate with the *Consumer Agent*. Using this model, the ordering of negotiations A, B and C is always $(B,C) \rightarrow A$. This could be a solution, but by no means to be the best solution under all circumstances. As we have analyzed before and also as the experimental results shown, the agent should dynamically choose the negotiation ordering based on the negotiation deadlines, decommitment penalties, the estimations of successful probabilities, and other environmental context so as to maximize the expected utility. However, the combinatorial auction model neither reasons about these attributes nor provides the agent with the flexibility to choose from different negotiation orderings. This limitation prevents the agent from finding a better negotiation solution.

Second, the agent using a combinatorial auction model neither actively reasons about the interrelationships among these related negotiation nor tries to direct the negotiations to a hopefully optimal solution, but just waits passively and select the solution from whatever is available, which does not guarantee finding a (good) solution. Let us continue with the previous example, using the combinatorial auction model, the *Computer Producer Agent* simply announces the two tasks *Get_Hardware* and *Deliver_Computer* and waits for the other agents' bids. When the *Hardware Producer Agent* and the *Transporter Agent* construct their bids for these two tasks respectively, they have no idea of how these two tasks relate to each other, all they can do is to construct the bids based on their local problem solving context. Suppose based on the "first come, first serve" rule, these two agents arrange these new tasks after their current tasks. Assume that the bid from the *Hardware Producer Agent* is "*Get_Hardware, cost* \$100, *time range:* 10–17" based on its current task finishes at 10 and it takes 7 time units to perform task *Get_Hardware*, and the bid from the *Transporter Agent* is "*Deliver_Computer, cost* \$5, *time range:* 15–21" based on its current task finishes at 15 and it takes 6 time units to perform task *Deliver_Computer*. However, based on these two bids, the *Computer Producer Agent* cannot find a consistent solution because there is no time left for the task *Install_Software*. Actually the solution does exist if the *Transporter Agent* would leave some slack time before starting task *Deliver_Computer*. The *Transporter Agent* does not have the necessary information that leads to this decision. To solve this problem in combinatorial auction, it can be requested that the contractee agent generate all possible bids and send them all to the contractor agent. However, this solution causes large amount of communication (as shown in Figure 16, upper part), and large number of bids makes the winning-determination (WD) process more difficult and time-

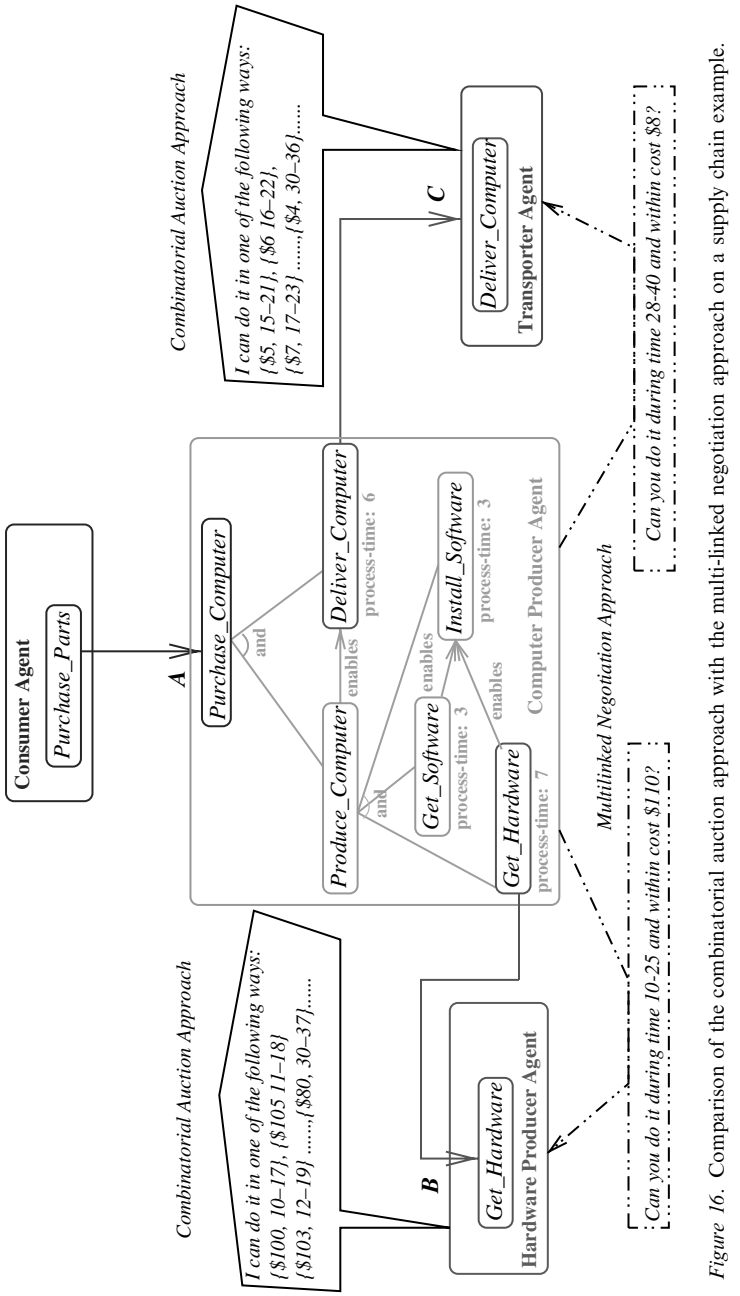

Figure 16. Comparison of the combinatorial auction approach with the multi-linked negotiation approach on a supply chain example.

consuming.[23] This example shows that combinatorial auction is not a suitable model for multi-linked negotiation with complicated task relationships. In comparison, in our approach (as shown in Figure 16, lower part), the *Computer Producer Agent*, who has the most complete information, leads the negotiation by analyzing the relationships among negotiations and arranging appropriate time ranges for related subjects in negotiation, which resulting in a more efficient negotiation process and a better solution in the end.

Thirdly, the general WD problem for combinatorial auction is NP-complete [3]. Current WD algorithms [11, 3] are based on depth-first search and using different types of heuristics. So, from the computational complexity perspective, combinatorial auction and our approach are at the same level of complexity.

The above analysis shows that combinatorial auction could be another approach to multi-linked negotiation, but it has limitation that does not permit efficient management of the negotiations where there are complex relationships. The approach in this paper provides a more general model and solution to multi-linked negotiation problem.

8. Summary

In this paper, we defined the multi-linked negotiation problem and demonstrate how an agent could deal with the multi-linked negotiation problem. Multi-linked negotiation deals with multiple negotiations, where these negotiations are interconnected — the negotiation over one issue affects other negotiations. To solve a multi-linked negotiation problem, the agent needs to find out in what order the negotiations should be performed, and how to negotiate on each issue to avoid conflict among them. First, we construct a partial order schedule, which allows the agent to reason about time-related constraints and flexibility on each issue. This reasoning process is important for the agent to perform conflict-free negotiation and manage flexibility in negotiation. Furthermore, we presented a formalized model of the multi-linked negotiation problem that enables the agent to represent and reason about the relationships among different negotiations explicitly. Using this model, a heuristic search algorithm is developed to that finds the nearly optimal approach in reasonable time. Experimental work shows that this management technique for multi-linked negotiation leads to improved performance over other simpler approaches.

In this work, we model the success probability as a function that depends on a set of features, but we have not worked out how the agent can construct such a function. In the future, we'd like to use meta-level information and learning technologies for an agent to construct and adjust the structure of this function. Also, the model and the algorithm presented here are for individual agents, to extend this model to a multi-agent system is another direction of our future work. Additionally, in this work, the result of the negotiation is limited to two outcomes: "success" or "fail". Actually, when negotiation is successful, there are potentially many different outcomes depending on the parameters in the commitment, such as different promising finish times. Depending upon the different outcomes, the agent can adjust its other negotiations that are related to this negotiation. The negotiation process can be

modeled as a Markov decision process, and the negotiation solution can be generated as a policy: perform the negotiation according to the results of the previous negotiations. This is another direction of our future work.

Notes

1. This approach seems too naive, but is commonly used. Most research only deals with single negotiation; little work has been done to study the relationships among different negotiations with complex task structures (Section 7 provides more discussion of related work).
2. In this framework, we allow a task to be completed in different ways which may lead to different quality achievements, different durations and different costs.
3. It is assumed that for each time unit the task being finished earlier than the deadline, the contractor agent gets extra reward $e * r$, but the total extra reward would not exceed the reward r.
4. Using this model, the penalty only depends on the decommitment rate and the regular reward in the contract. Actually a more complicated model can be introduced where the time of decommitment is taken into consideration, i.e., a decommitment announced earlier has less penalty than a decommitment in the last minute.
5. For example, the minimum quality requirement is not applicable for a resource requirement. A quantity requirement may be necessary to specify how much resource is needed.
6. The agent will not schedule every time a new task arrives, but will schedule all tasks that fall into the same scheduling time window.
7. The task cannot be started until the contract has been confirmed.
8. In this work, we use MQ scheduler as agent's local scheduler, which is based on the MQ framework [14] that allows agents to reason about different organizational objectives.
9. There are different ways to perform a task, which are represented as different methods in the task structures. In Figure 4, *Computer Producer Agent* chooses to deliver the computer through the transporter agent (*Deliver_Computer_A*) for task *Purchase_Computer_A* while ship the computer through a package mailing system (*Shipping_Computer_A*) for task *Purchase_Computer_B*. This decision is made by the agent's scheduler depending on the difference of the characteristics of these methods and the problem-solving context.
10. There are other attributes in the proposal that also can be negotiated over, such as *regular reward*, *earlier reward rate*, and *decommitment penalty*. We only mentioned *promised finish time* here as an example, because it is closely related to other negotiations.
11. Isolated nodes can be either independent or indirectly linked, depending on whether they compete for the same resource. Let us take the computational resource as an example: if the time window [est, dl] for the two negotiation subjects are overlapped, they are indirectly linked; otherwise, they are independent.
12. In this case, we used an expectation of the negotiation duration, which could be learned from experience.
13. It assumes the negotiation on an issue starts immediately after all the negotiations that precede this negotiation have been finished according to the negotiation ordering.
14. The start time specifies the earliest start time for all negotiations. It is also possible to specify a separately earliest start time for each negotiation.
15. If the set of valid feature assignments is a complete set of all possible valid feature assignments, this algorithm is guaranteed to find the best negotiation solution. However, when the attributes have continuous value ranges, it is impossible to find all possible valid feature assignments. We use a depth- first search (DFS) algorithm that searches over the entire value space for all undecided attributes by pre-defined search step size and finds a set of valid feature assignments (See Appendix, Algorithm A.1).
16. The algorithm checks whether adding POR e to ϕ causes a circle. If so, e will not be added, and the algorithm will randomly choose another POR and continue.
17. In this paper, the term "partial-order schedule" refers to a representation of a group tasks with specified precedence relationships, which also includes the associated definitions in this section. The term "partial-order scheduler" is used to refer to the procedure which actually produces the partial-order schedules for tasks, and a set of associated reasoning algorithms presented in Section 4.2.

18. *Outside earliest start time* for task t is the earliest possible start time decided by the problem-solving context. As a given parameter, it is not changeable during the partial order reasoning process. For example, if the current time is 15, the task cannot start before time 15. In a similar way, the *outside deadline* constraint is the task's *deadline* decided by the problem solving context.
19. Partial order schedule is a representation and reasoning tool of a group of tasks and their interrelationships. 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: the task is interruptible or non-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 non-interruptible execution assumption does not allow execution of a task to be split into parts. In this work we adopt the interruptible execution assumption, however, we also do not consider there is cost for interrupting and resumption of a task.
20. This function describes a phenomenon where initially the likelihood of a successful negotiation increases significantly as the flexibility grows, and then levels off afterwards. This function mirrors our experience from the experiments in Section 6.3, which shows that after a certain point, additional flexibility does not significantly improve the success probability. Obviously this function could be affected by the meta-level information from the other agent.
21. Using a *t*-test, with the 0.001 alpha-level, the following hypothesis H_o is rejected: when using the decision-based approach, Computer Producer Agent achieves an extra utility that is equal to 100% of the utility gained when using the sequenced negotiation strategy, and 78% of the utility gained when using parallel negotiation strategy, compared to the hypothesis H_a: when using the decision-based approach, Computer Producer Agent achieves an extra utility that is more than 100% of the utility gained when using the sequenced negotiation strategy, and 78% of the utility gained when using parallel negotiation strategy.
22. Using a *t*-test, with the 0.01 alpha-level, the following hypothesis H_o is rejected: when using the flexibility policy, *Computer Producer Agent* achieves an extra utility that is equal to 64% of the utility gained when using the *Earliest Finish Time Policy*, compared to the hypothesis H_a: when using the flexibility policy, *Computer Producer Agent* achieves an extra utility that is more than 64% of the utility gained when using the *Earliest Finish Time Policy* .
23. There has been some recent work on preference elicitation [11] that potentially could reduce the number of bids need to be sent. However, it is our intuition that to make this preference elicitation process successful, it would need the similar type of reasoning process as shown in our work.

References

1. W. Conen and T. Sandholm, "Preference elicitation in combinatorial auctions: Extended abstract", in *ACM Conference on Electronic Commerce (ACM-EC)*, Tampa, FL, October 14–17, 2001.
2. M. Frank, A. Bugacov, J. Chen, G. Dakin, P. Szekely, and B. Neches, "The marbles manifesto: A definition and comparison of cooperative negotiation schemes for distributed resource allocation", in *AAAI Fall 2001 Symposium on Negotiation Methods for Autonomous Cooperative Systems,* 2001.
3. Y. Fujishima, K. Leyton-Brown, and Y. Shoham, "Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches". in *Proceedings of International Joint Conference on Artificial Intelligence IJCAI'99*, Stockholm, Sweden, 1999.
4. B. Horling, R. Vincent, and V. Lesser, "Multi-agent system simulation framework". in *16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation.* EPFL, August 2000.
5. B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser, "Distributed sensor network for real time tracking". in *Proceedings of the 5th International Conference on Autonomous Agents*, ACM Press: Montreal pp. 417–424, June 2001.
6. L. Hunsberger and B. J. Grosz, "A combinatorial auction for collaborative planning", in *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, 2000.
7. J. J. Moder, C. R. Phillips, and E. W. Davis. *Project Management with CPM, PERT and Precedence Diagramming*. Blitz Pub Co, 1995.

8. A. Pritsker, "Gert networks graphical evaluation and review technique". *The Production Engineer*, 1968.
9. T. Sandholm and V. Lesser, "On automated contracting in multi-enterprise manufacturing". in *Proceedings of the Improving Manufacturing Performance in a Distributed Enterprise: Advanced Systems and Tools*.
10. T. Sandholm and V. Lesser, "Issues in automated negotiation and electronic commerce: Extending the contract net framework", in *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS95)*, 1995.
11. T. Sandholm and S. Suri, "Improved algorithms for optimal winner determination in combinatorial auctions and generalizations", in *National Conference on Artificial Intelligence (AAAI)*, 2000.
12. S. Sen and E. H. Durfee, "A formal study of distributed meeting scheduling", *Group Decision and Negotiation*, vol. 7, 265–289, 1998.
13. R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver", *IEEE Transactions on Computers*, 1980.
14. T. Wagner and V. Lesser, "Evolving real-time local agent control for large-scale mas", in J. Meyer and M. Tambe, (eds.) *Intelligent Agents VIII (Proceedings of ATAL-01)*, Lecture Notes in Artificial Intelligence: Springer-Verlag, Berlin, 2002.
15. W. Walsh, M. Wellman, and F. Ygge, "Combinatorial auctions for supply chain formation", in *Second ACM Conference on Electronic Commerce*, 2000.
16. X. Zhang, *Sophisticated Negotiation In Multi-Agent Systems*. Ph.D. thesis, University of Massachusetts: Amherst, 2002.
17. X. Zhang, V. Lesser, and R. Podorozhny, "New results on cooperative, multistep negotiation over a multi-dimensional utility function" in *AAAI Fall 2001 Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pp. 1–10, 2001. http://mas.cs.umass.edu/~xqzhang/pub/NewResult-Neg-01.ps.

A. Appendix

Algorithm A.1 *Find a set of valid feature assignments.*
Input: $\mathcal{M} = (\mathcal{V}, \mathcal{E})$
For each attribute a_{ij}, *if* a_{ij} *is already decided, the value of* a_{ij} *is decided_ value*(a_{ij}); *if* a_{ij} *is undecided, the maximum possible range for* a_{ij} *is:* [*min_value*(a_{ij}), *max_value*(a_{ij})], *the search step size:* $step_{ij}$.
Output: a set of valid feature assignments ω.

Generate the possible value set Ψ_{ij} *for attribute* a_{ij};
If a_{ij} *is already decided,* $\Psi_{ij} = \{decided\_value(a_{ij})\}$;
Else $x = min\_value(a_{ij})$;
Repeat
add x *to* Ψ_{ij};
$x = x + step_{ij}$;
Until $x > max\_value(a_{ij})$
Generate all possible feature assignments φ_k *based on the possible values in* Ψ_{ij};
If valid (φ_k), *add* φ_k *into* ω;
Return ω;

Algorithm A.2 *Evaluate a negotiation schedule with all possible feature assignments and find the best feature assignments and the best value.*
Input: negotiation schedule ϕ, a set of valid feature assignments $\omega = \{\varphi_k\}$, $k = 1, \ldots, m$.
Output: the best value with the best feature assignment.

```
begin
  for(i = 0; i <= m; i += sample_step)
        add φ_i to search_set;
  for each φ_i in search_set
        for(t = 0; t < search_limitt; ++)
            if (EV(φ, φ_{i+1}) > EV(φ, φ_i))
                i = i + 1;
            else if (EV(φ, φ_{i-1}) > EV(φ, φ_i))
                i = i - 1;
            else
                break;
        if (EV(φ, φ_i) > best_value)
            best_value = EV(φ, φ_i);
            best_assignment = i;
  return(best_value, best_assignment);
end
```

Algorithm A.3 *Complete search: Find the best negotiation strategy.*
Input: $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, the start time for negotiation τ, a set of valid feature assignments $\omega = \{\varphi_k\}$, $k = 1, \ldots, m$.
The complete search algorithm evaluates each pair of negotiation ordering and valid feature assignment $\mathcal{EV}(\phi_i, \varphi_k)$, then return the best one.
Output: the best negotiation strategy.

```
Generate all valid negotiation orderings {φ_i};
best_value = minimum_value;
best_ordering = null;
best_assignment = null;
for each negotiation ordering φ_i
    for each valid feature assignment φ_k
        if EV(φ_i, φ_k) > best_value
            best_value = EV(φ_i, φ_k);
            best_ordering = φ_i;
            best_assignment = φ_k;
return (best_ordering, best_assignment)
```

Algorithm A.4 *Heuristic search: Find the best negotiation strategy.*
Input: $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, *the start time for negotiation* (τ), *a set of valid feature assignments* $\omega = \{\varphi_k\}$, $k = 1, \ldots, m$, *the probability to add a por:add_ por_probability. TEMP_MAX, TEMP_STEP: search parameters.*
Output: the best negotiation strategy.

```
begin
  Generate all possible PORs = {(v_i, v_j)|v_i, v_j ∈ V}
  total_value = 0;
  total_inverse_value = 0;
  base_value = evaluate_schedule(NS(V, ∅), ω);
  for each por ∈ PORs
      φ(por) = (V, por)
      por.value = evaluate_schedule(NS(φ(por)), ω).value − base_value;
      por.inverse_value = 1.0/por.value;
      total_value = total_value + por.value;
      total_inverse_value = total_inverse_value + por.inverse_value;
  for each por ∈ PORs
      por.in_probability = por.value/total_value;
      por.out_probability = por.value/total_inverse_value;
  for(t = TEMP_MAX; t >= 0; t− = TEMP_STEP)
  generate a random number r between [0, 1];
      if(r < add_por_probability)
          choose a por e from PORs/current_ordering
          according to in_probability
          new_ordering = current_ordering ∩ e
      else
          choose a por e from current_ordering according to
          out_probability
          new_ordering = current_ordering − e
      evaluation_result = evaluate_schedule(NS(φ(new_ordering)), ω);
      change_value = evaluation_result.value-current_value;
      if (change_value > 0||random < e^(−change_value/t))
          current_value = evaluation_result.value;
          current_assignment = evaluation_result.assignment;
          current_ordering = new_ordering;
      if (change_value > best_value)
          best_value = current_value;
```

$best\_assignment = current\_assignment;$
$best\_ordering = current\_ordering;$
return $(best\_ordering, best\_assignment);$
end