Measuring the Expected Gain of Communicating Constraint Information*

Avi Rosenfeld[1], Sarit Kraus[2] and Charles L. Ortiz, Jr.[3]
[1]Department of Industrial Engineering
Jerusalem College of Technology, Jerusalem, Israel 91160
[2]Department of Computer Science
Bar-Ilan University, Ramat-Gan, Israel 92500
[3]SRI International, 333 Ravenswood Avenue
Menlo Park, CA 94025-3493, USA
Email: rosenfa@jct.ac.il, sarit@cs.biu.ac.il, ortiz@ai.sri.com

September 14, 2008

Abstract

In this paper we investigate methods for measuring the expected utility from communicating information in multi-agent planning and scheduling problems. We consider an environment where human teammates can potentially add information to relax constraint information. As these problems are NP-complete, no polynomial algorithms exist for evaluating the impact of either adding or relaxing a certain constraint will have on the global problem. We present a general approach based on a notion we introduce called *problem tightness*. Distributed agents use this notion to identify those problems which are not overly constrained and, therefore, will not benefit from additional information that would relax those constraints. Finally, agents apply traditional machine learning methods based on their specific local problem attributes to attempt to identify which of the constrained problems will most benefit from added information. We evaluated this approach within a distributed c-TAEMS scheduling domain and found that this approach was effective overall.

Keywords: Multiagent Scheduling, Adaptive Coordination, Localized Decisions

*This research was supported by the DARPA Coordinators Program under Air Force Research Laboratory Contract FA8750-05-C-0033. Sarit Kraus is also affiliated with UMIACS.

1 Introduction

The use of mixed human and agent teams can be critical in performing a variety of complex planning and scheduling tasks [22, 23, 24]. Both agents and people have diverse capabilities that must be leveraged within the planning and scheduling tasks. The importance of effective coordination within mixed human-agent teams has been shown to be critical in domains such as hazardous cleanup, emergency first-response, and military conflicts [22, 23].

This paper focuses on how agents can better focus a human operator's attention within planning and scheduling tasks. Following previous work [22, 23], we use the term "automated scheduling agent" (ASA) to refer to an autonomous multi-agent system that supports coordinated scheduling of people carrying out complex tasks in a dynamic environment. The team's quality is based on the schedules the ASA agents produce. The agents within the team have the advantage that they can process large amounts of constraint information, a characteristic that can be critical, especially in dynamic and time sensitive environments [13, 14, 24, 33]. However, the ASAs exist within a distributed multi-agent system with each ASA having only a partial, localized view of problem constraints. ASA agents are separated either geographically, due to security reasons or communication cost considerations. Thus, even discounting the system dynamics, constructing a global view of the scheduling and planning problem is not feasible [22, 23].

In the human-agent team under consideration, human operators are assumed to have access to more complete knowledge about task uncertainty because they have updated information or expert knowledge external to the ASA agents. They might be able to provide information that either relaxes or updates scheduling constraints. For example, a human operator in an emergency response environment may have updated information about weather conditions, or knowledge acquired from years of experience. This person's knowledge potentially allows ASA agents to schedule additional tasks or tasks with higher qualities, thus improving the team's schedule [22, 23].

Given such a heterogeneous team, a decision framework is typically used to weigh the estimated costs and gains associated with user interactions [5, 9, 10, 8, 22, 23]. The ASA agents must be particularly sensitive about the cost associated with obtaining information from the human operators [24]. People are often busy and one must assume that prompting a user for information incurs some cost. Thus, the first part of any proposed solution must quantify the cost related

to interrupting the user for information [22]. Additionally, the ASA agents must consider the potential impact, or gain, associated with obtaining this information. Once these agents have this combined information, they can successfully decide whether to ask the user about a particular constraint.

In this paper we focus on the second part of the human-agent decision framework just described: how to quantify and classify the expected gain from additional user information. We present a two stage approach to address this challenge.

- **Initial Constraint Assessment** - Local ASA agents first share a minimal amount of local constraint information with other team agents. Local agents need to obtain enough problem information to compute a value for a general, non domain-specific constraint "tightness" measure that we define. Specifically, we assume constraints are ordered through a constraint tree structure with similarities to partial order planners [11] and Hierarchical Task Network (HTN) often used in scheduling and planning problems [17, 18, 19, 26, 32]. Agents must share enough information to construct the basic constraint tree structure. A constraint tightness of less than one indicates this task is underconstrained and can be locally solved. Agents can then immediately identify those tasks for which added information will not impact the expected utility.

- **Estimating the Expected Gain from Information** - A tightness measure of greater than one indicates that the given task is constrained and *may* be influenced by other scheduling constraints. Even after the first stage, agents still possess insufficient constraint data to properly judge the potential impact of information in this case. Nonetheless, we found that agents can apply machine learning techniques learned offline to their local problem attributes to estimate which of the remaining tasks are likely to have the highest potential gain from a user's information.

The next section provides the background and motivation for this work. In Section 3 we briefly describe a general planning and scheduling formalization, and use this formalization to define a problem "tightness" measure for locally identifying constraints that will not benefit from added information. In Section 4 we present the c-TAEMS language as an instance of the formalization defined in Section 3. Additionally, we describe how the presented tightness measure can be applied to this domain. Section 5 presents a method for measuring the

value of information through machine learning regression and classification tree models. Section 6 provides experimental results. In Section 7, we discuss the general applicability of these results, as well as provide several directions for future research. Section 8 concludes.

2 Related Work

The goal of this paper is to quantify the potential utility gain from added information in distributed scheduling problems. We present this challenge as the "Coordination Autonomy" (CA) problem, or the challenge of how agents decide what constraints should be forwarded to their human operator for further information. Thus, this paper is linked to previous research in the field of agent-human interactions, as well as previous distributed scheduling research.

The adjustable autonomy challenge as described by Scerri et al. [24] refers to how agents can vary their level of autonomy, particularly in dealing with other types of entities such as people. They proposed a framework where agents can explicitly reason about the potential costs and gains from interacting with other agents and people to coordinate decisions. Other frameworks have similarly suggested creating a utility based framework to weigh the potential gain and user interaction cost tradeoff [5, 8, 9, 10]. However, a key issue in applying such models within new domains is to effectively quantify the utility that can be potentially gained from other teammates.

Similar issues have arisen in the Information Gain measure that has been put forward by the machine learning community [15]. Information gain is typically used to learn the effectiveness of a certain attribute in classifying data, or how much additional information is gained by partitioning examples according to a given attribute. Thus, one approach may be to query the ASA agents' scheduler with and without a given piece of information and build a solution based on the resulting Information Gain.

However, there are several reasons why measures such as Information Gain cannot be applied to this type of problem. First, in this and many real-world applications there is a cost involved with agents exchanging constraint information and also a cost involved with profiling a user. As a result, the cost in computing the Information Gain for a given problem will likely outweigh its benefit. Previous frameworks [5, 9, 10, 8] assumed such information could be freely obtained. Second, since there is a potentially large number of tasks to schedule, the size

of the learning space will be very large. Sending queries to a local scheduler for each task with and without all potential information can become resource intensive, leading to delays. Finally, sending too much constraint information can result in a lower team quality: we have previously found that in highly constrained problems, sending additional information can prevent agents from finding the optimal solution [20]. More generally, previous work has found that agents that send all information at their disposal do not necessarily increase the team's utility. As Durfee has remarked: "ignorance is bliss" [4]. It is not always beneficial for agents to have more information.

Distributed scheduling problems belong to a more general category of Distributed Constraint Optimization Problems (DCOP) [13], and a variety of algorithms have been proposed for solving these problems [13, 14, 33]. The specific c-TAEMS language is based on the TAEMS specification [12], and is currently being used by many researchers for studying multiagent task scheduling problems [22]. c-TAEMS has been shown to directly map to the general DCOP formalization [28]. In c-TAEMS problems specifically, and DCOP problems more generally, inter-agent constraints must be coordinated to find the solution that satisfies as many of these constraints as possible. However, these problems are known to be NP-complete, even if agents freely share all information at their disposal [14, 25]. As such, no polynomial algorithms exist for checking how a scheduling problem's utility is affected by a given piece of information. Thus, the proposed solution relies on making decisions with incomplete problem information and can be classified within the research area of bounded rationality [10, 21].

This paper's solution is to estimate the potential gain from interacting with the user through analyzing the scheduling problem's structure, even when agents only have access to limited, locally available information. The significant in this approach is that agents reason about a limited number of interactions, allowing for tractable solutions. Previous approaches considered all possible team interactions, potentially creating a need to generate very large numbers of resource intensive "what if" queries to obtain this information [22, 23, 25, 27].

The concept of developing metrics to help distributed agents select the best problem solving approach or heuristic, such as the tightness measure presented in this paper, was previously proposed by Fox et al. [6] and Sycara et al. [29]. Our work differs in its two-pronged approach: agents first identify which constraints can be locally solved based on the "tightness" here described. We then focus on using machine learning models as described in Section 5 to better quantify

and classify the estimated gain on the remaining constraints from any additional information that can be provided by the user. In contrast, previous work [6, 29] uses such measures to guide which heuristic to use in a one-stage approach to solving the problem. As a result, local agents needed to gather significantly more information regarding their constraints to aid in their selection of heuristics to solve the problem. Such an approach is not appropriate when costs are associated with sharing constraint information.

Even after this potential *gain* has been measured, it is also critical to effectively measure the estimated *cost* in interacting with a given user in the team. Only after both the gains and costs of a given user have been modeled can one create a decision theoretic framework to decide if the user should be prompted for information [5, 9, 10, 8]. Previous approaches, most noticeably work by Sarne and Grosz [23] complement our work by modeling the estimated *cost* involved with interacting with a given human user. In this paper, we assume user costs can be modeled based on their approach.

In our approach, we draw on previous work that has identified the existence of additional categories of problem complexity exist within NP-complete problems [2, 16]. Many instances of NP-complete problems can still be quickly solved, while other similar instances of problems from the same domain cannot. These studies have introduced the concept of a phase transition to differentiate classes of "easy" and "hard" instances of a problem [16]. Thus, while scheduling and planning problems may be NP-complete, we posit that certain portions of these problems are underconstrained and represent "easy" problem instances within the global scheduling problem. These instances can be locally solved by ASA scheduling agents without any additional information. Other instances, or portions within the scheduling problem, are "hard" instances and the potential impact of information cannot be easily assessed. We present an approach in which the potential impact of information is estimated through machine learning models that were previous learned offline.

However, finding such phase transitions within real-world domains is far from trivial due to the varied types of possible agent interactions that constitute these problem instances [1, 25]. In the theoretical graph coloring problems previously studied, phase transitions were found that were associated with the ratio of constraint clauses per variable [16]. Unfortunately, these graph coloring problems are relatively simple in that all constraints typically have equal weighting and every agent has equal numbers of constraints (edges). Thus, discovering phase transitions in experiments can be accomplished only through variation of a single

parameter – the ratio of graph edges to nodes. In contrast, as we now describe, many real-world domains, such as the c-TAEMS scheduling domain we have focused on, are far more complex. Novel measures are needed to quantify inter-agent actions in this and similar domains.

3 Domain Formalization and Description

In the CA problem we consider, the problem's input is an agent's local scheduling and planning constraints. The proposed output is the constraints with uncertainty that a person should provide information about. In this section, we first model in Section 3.1 a general representation suitable for a family of partial order and HTN planning and scheduling problems. We use this model to present the problem as to which constraints with uncertainty are most worthy of the user's attention. Then, in Section 3.2, we present how these definitions are used to generally compute a constraint's *problem tightness* which can locally quantify if a constraint can be locally solved or should be referred to the user for additional information. In Section 4 we present details of how c-TAEMS represents one domain instantiation of this general formalization. We also provide specific examples of the type of information the user may be able to provide and how the tightness measure of Section 3.2 can be applied to this domain.

3.1 General Planning and Scheduling Formalization

We generally describe the scheduling problem as follows: let $G = \{A_1, A_2, \ldots, A_n\}$ be a team of n agents trying to maximize their team's collective scheduling utility. These agents must perform a certain task, $\mathcal{T}$, which must be executed within a finite timeline, τ_s until τ_e. τ_s and τ_e are positive integers. Task $\mathcal{T}$ is composed of a set of m subtasks, T, that can be performed by G such that $T = \{T_1, \ldots, T_m\}$. These subtasks are divided into two categories: basic subtasks that can be performed by one of the agents, and complex subtasks that are composed of other subtasks. Each subtask $T_i \in T$ has a time *Window* or possible start and end times during which that task can be executed. We define the window, W_i, for subtask T_i as being from $\tau_{i,s}$ to $\tau_{i,e}$ such $\tau_s \leq \tau_{i,s} \leq \tau_{i,e} \leq \tau_e$. The window's length is defined as the time $\tau_{i,e}$ - $\tau_{i,s}$.

We assume that there is a partial order between the subtasks, $\prec$, such that $T_i \prec T_j$ indicates that T_i needs to be performed before the beginning of the execution of T_j.

Each basic subtask T_i is associated with the following elements:

1. An agent $A_i \in A$ that can perform the basic subtask.
2. A possible quality outcomes $Q_i = \{Q_{i1}, Q_{i2}, \ldots, Q_{ik_i}\}$ indicating that if agent A_i will perform T_i the quality for the team will be a member of Q_i.

   Q_i is associated with a probability distribution function $q_i : Q_i \rightarrow [0,1]$ such that $\sum_{j=1}^{k_i} q_i(Q_{ij}) = 1$.
3. A set of possible durations length of T_i, $D_i = \{D_{i1}, D_{i2}, \ldots, D_{il_i}\}$ indicating that it will take A_i to perform T_i one of the possible durations in D_i.

   D_i is associated with a probability distribution function $p_i : D_i \rightarrow [0,1]$ such that $\sum_{j=1}^{l_i} p_i(D_{ij}) = 1$.

The relationships between the subtasks are expressed using a tree, $TR = (V, E)$ such that the root of the tree is labeled with $\mathcal{T}$. The leaves of the tree are labeled with basic subtasks and the rest of the nodes are labeled with complex subtasks. For any two subtasks $T_i, T_j \in T$ associated with the nodes $v_i, v_j \in V$, respectively, such that v_i is the child of v_j in TR the window of T_i is included in the window of T_j. That is, $\tau_{i,s} \geq \tau_{j,s}$ and $\tau_{i,e} \leq \tau_{j,e}$.

Using the tree TR we extend the partial order between the subtasks, denoted $\prec^*$ as follows. For any $T_i, T_j \in T$:

1. If $T_i \prec T_j$ then $T_i \prec^* T_j$.
2. If $\tau_{i,e} \leq \tau_{j,s}$ then $T_i \prec^* T_j$.
3. If $T_i \prec T_j$ and there is paths between T_i and $T_i^{'}$ in TR and a path between T_j and $T_j^{'}$ then $T_i \prec^* T_j$.

We are now ready to define a schedule.

Definition 1. A schedule π is a set of pairs $\{< \tau_1^s, T_\ell^s >, ..., < \tau_l^s, T_\ell^s >\}$, such that

1. For $1 \leq i \leq \ell$, $T_i^s \in T$ is a basic subtask, where $\tau_{i,s}^s$, $\tau_{i,e}^s$, D_i^s, Q_i^s are the beginning and end times, the duration lengths and quality outcomes sets of T_i^s, respectively, and $\tau_i^s \geq \tau_{i,s}^s$ and $\tau_i^s + max\{D_i^s\} \leq \tau_{i,e}^s$.
2. For any two pairs $< \tau_i^s, T_i^s >$ and $< \tau_j^s, T_j^s >$, $1 \leq i, j \leq \ell$,
   (a) If $T_i^s \prec^* T_j^s$, then $\tau_i^s + max\{D_i^s\} \leq \tau_j^s$.

(b) Assume, without loss of generality that $\tau_i^s \leq \tau_j^s$. If the agent A_i associated with T_i^s is the same as the agent A_j associated with T_j^s, (i.e., $A_i = A_j$), then $\tau_i^s + max\{D_i^s\} \leq \tau_j^s$.

Let $\pi = \{< \tau_1^s, T_1^s >, ..., < \tau_l^s, T_\ell^s >\}$ be a schedule. The goal of the agents is to find a schedule $\pi = \{< \tau_1^s, T_1^s >, ..., < \tau_l^s, T_\ell^s >\}$ such that for any other schedule $\pi' =< \tau_1^{s'}, T_1^{s'} >, ..., < \tau_k^{s'}, T_k^{s'} >$

$$\sum_{<\tau_i^s, T_i^s> \in \pi} min\{Q_i^s\} \geq \sum_{<\tau_i^{s'}, T_i^{s'}> \in \pi'} min\{Q_i^{s'}\}$$

.

It is clear that additional information from the user reducing the number of members of Q_is and D_is can improve the schedule the agents can find. In fact, we generally refer to constraints of interest as those with uncertainty in the values of Q_i and D_i. Thus, the added value information is gained through reducing the uncertainty in these values. Without any such information, agents supporting the automated scheduling process assume that the worst-case scenario must be planned for (e.g. that the task will have the smallest possible quality or the smallest value for Q_i, or the task will take the longest possible time or the largest value for D_i).[1]

This problem is magnified by the fact that each agent has only partial information of the problem. An agent's local view consists of parts of the tree. We assume that the local view of agent $A \in G$ consists of all the leaves of the tree such that A is associated with the basic subtasks of these leaves. Furthermore, if an agent's local view consists of a node that is associated with a subtask $T_i \in T$ and there is at least another subtask T_j such that $T_i \prec T_j$ or $T_j \prec T_i$ then agent A knows that such a constraint exists, but may not know T_j. This paper considers the following problem: given the local view of an agent, how can it estimate the basic subtasks for which additional information from the user on the quality or the duration of this subtask may improve the team's schedule[2].

The above general formalization is common to partial order planners [11], recipes based cooperative planning [7] and contains similarities to Hierarchical Task Network (HTN) representations used by many classical planners such as

[1]Domains may exist where the worst-case execution time (WCET) assumption is not necessarily appropriate as a default for scheduling. For example, some systems may have some built in resilience to some degree of error. Nonetheless, the described approach is critical as the input from the user may still significantly relax the ASA's constraints and improve the scheduler's quality.

[2]This paper's focus is not on how to find the best schedule, π, from local agents' views. This problem has been addressed in previous research [27, 30].

SIPE-2 [17], SHOP [18, 19, 32], as well as by the TAEMS and c-TAEMS language [12]. In HTN based representations, the root node branches into *tasks* and *subtasks*. The leaves of the tree contain the basic subtasks which are called *methods* in c-TAEMS and primitive tasks in other HTN planners. These basic subtasks represent the actions that can be directly executed by each of the agents within the team [12, 17, 18, 19, 32].

3.2 Defining Tightness to Quantify Constraint Types

Based on the above general definition, we propose a *problem tightness* measure to identify which types of basic subtask constraints are most worthy of further information. Our hypothesis is that different types of problems can be identified, similar to the phase shifts found within simpler graph coloring optimization problems previously studied [2, 16]. However, novel measures of problem difficulty are needed to help identified problems so phase shifts can be discovered.

The "tightness" measure presented in this section quantifies if a constraint is definitely underconstrained, and will not benefit from additional information. However, before relying on this locally measured value, agents must first confirm that no partial orders exist that might affect this measure. This is done by a small amount of localized communication, as described below.

In this paper, we typically assume that only one agent is assigned to basic subtasks within a given task window. Thus, W_i is assigned to only one agent within the team of n agents, and that agent can independently measure its "tightness" in W_i. This assumption can be easily relaxed through localized communication between all m agents under a task window W_i. This communication, between the agents $\{A_1, A_2 \ldots, A_m\}$ in W_i can then allow each agent in $\{A_1, A_2 \ldots, A_M\}$ to build the total set of all possible task durations and qualities within W_i.

Based on these definitions and assumptions, we define an agent's quality *tightness* as:

Tightness-Quality(T_i) = $\frac{Quality_{max}(T_i)}{Quality(Window(T_i))}$

where $Quality_{max}(T_i)$ returns the maximal quality from the possible values of T_i, $\{Q_{i1}, Q_{i2}, \ldots, Q_{ij}\}$, and $Quality(Window(T_i))$ returns the minimal expected quality of all *other* subtasks that can be executed within the task window of T_i. Note that if quality uncertainty exists in these other tasks, we assume the worse case, and the lowest quality value is used in computing the value of $Quality(Window(T_i))$. The measure of Tightness-Quality(T_i) can then be used to quantify what potential overlap exists in the qualities distributions found within T_i's task window. This

allows the local agents to decide if additional information about their quality uncertainty may aid in their decision process, or if they can solve the constraints in this subtask without further information.

For example, assume task T_A can be fulfilled by basic subtasks A1 and A2. A1 has a quality distribution between 10 and 20, and A2 has a quality distribution of 15 and 25. Observe that a greedy scheduler should schedule A2 as its maximal utility (and average) is greater than that of the other basic subtasks. Also note that as per the tightness definition, Tightness-Quality(A2) = 25/10 = 2.5 as its maximal quality is 25 and the minimal quality value of the other subtasks is 10. However, it is possible that asking the user for input about the actual quality of A1 is worthwhile as the quality distributions of A1 and A2 overlap and A1 may have a higher quality than A2 (say 20 for A1 and 15 for A2). This is noted through observing that the tightness measure A1 is also greater than one as Tightness-Quality(A1) = 20/15 = 1.33. Thus, the ASA agents should decide that a greedy decision to select A2 is not necessarily correct as the tightness-quality of *another* basic subtask (here A1) within the task window is greater than 1.

Conversely, assuming ASA agents find no other basic subtask whose quality tightness is greater than one, it can definitively select the basic subtask with the highest maximal quality. Again, assume task T_A can be fulfilled by basic subtasks A1 and A2, but A1 has a quality distribution between 10 and 20 while A2 has a quality distribution of 25 to 35. In this case, basic subtask A2 should clearly be scheduled as even in the worse case this basic subtask will return a quality of 25 while A1 in the best case only returns a quality of 20. Similarly, Tightness-Quality(A1) = 20/25 = 0.8 indicating that basic subtask A2 should be selected regardless of any additional information. Thus, ASA agents can decide when additional information will *not* be useful based on this measure.

Similarly, we model an agent's duration *tightness* as:

Tightness-Duration(T_i) = $\frac{Duration_{max}(T_i)}{Length(Window(T_i))}$

where $Duration_{max}(T_i)$ returns the maximal duration from $\{D_{i1}, D_{i2}, \ldots, D_{ij}\}$, and $Length(Window(T_i))$ refers to the length of the time window allotted for completing task T_i. As per our previous definitions in Section 3.1, a window's length is defined as the time $\tau_{i,e}$ - $\tau_{i,s}$.

As was the case within the quality tightness measure, a value of more than one indicates that user information may be useful as an overlap between the quality distributions in basic subtasks exist. For example, assume task T_A may take either 10, 20, or 40 time units to complete within a time window, $Window(T_A)$ of 25 units. According to the tightness definition, tightness Tightness(T_A) is 40/25 of

1.6. Without any additional information, the ASA's scheduler must assume this subtask will take the maximal time, and cannot be scheduled in its time Window of 25 units. However, assuming information can be provided regarding the task's duration, say that T_A will only last for 10 or 20 units, the value of Tightness(T_A) drops below 1 and the ASA agent can schedule task T_A to be executed.

However, there may be a problem with defining W_i as per the above definitions. As can be seen from the definition of a schedule (Definition 1), there are many cases where the actual window in which a basic subtask T_i can be scheduled is only a subset of $Window(T_i)$, the defined window that is associated with that subtask. For example, if $T_i \prec^* T_j$ then the actual time point in which the performance of T_i can end is $min\{\tau_{i,e}, \tau_{j,e} - max\{D_j\}\}$. Similarly, if $T_j \prec^* T_i$ then the performance of T_i can start only at $max\{\tau_{i,s}, \tau_{j,s} + max\{D_j\}$. Furthermore, the actual window depends on the schedule that the team chooses, i.e., the possible times should satisfy the constraints of Definition 1. Thus, the definition of tightness should depend on the actual time window, not the defined one. However, computing the exact actual window requires finding a schedule, which is time consuming and may require global information. Additionally, as we assume there is computational and communication cost associated with communicating this information [4, 20] this approach is undesirable.

To overcome this problem, we extend the definition of tightness as follows: If the actual window of a subtask, T_i is different than $Window(T_i)$ then both the Tightness-Duration(T_i) and Tightness-Quality(T_i) are defined to be higher than one. Otherwise, the original definition is used. In this way, in cases where the actual actual window of T_i cannot be easily locally quantified, agents simply assume the worse case about this measure; that is, that the task windows tightness is more than one and that constraint cannot be locally solved efficiently without additional information.

The main question that remains is how the agent can decide whether the actual window of a subtask is equal to the defined window of this subtask. We propose that this decision will depend on the actual representation of the task information. In this paper we will focus on this decision in the context of the c-TAEMS Domain.

4 Analyzing Planning and Scheduling Constraints in c-TAEMS

In this section we present how the above general planning and scheduling formalization of Section 3.1 can be specifically applied to the c-TAEMS domain. Additionally, we provide illustrating examples as to what constraint information could potentially be added by the team's human users. Finally, we present how the generally defined tightness measure of Section 3.2 could be specifically applied to c-TAEMS.

4.1 c-TAEMS Domain Description and Examples

c-TAEMS is an instantiation of a general planning and scheduling problem defined in the previous section. Within c-TAEMS, the root task is referred to as the "task group". Time constraints are assigned to Windows and as such are inherited from their parents's nodes in the tree, ultimately terminating in the root node. In c-TAEMS, these basic subtasks are referred to as *methods* and are assigned to a specific agent within G.

Methods have associated with them a list of one or more potential outcomes. Values for these outcomes include: what that method's *Quality* will be, as the utility that the task will add to the team upon its successful completion, that method's *Duration* as the length of time the task requires to be completed, and *Cost* as the cost involved with executing that method. As the agents' cost directly impacts its utility, (e.g. the method's total utility is $Q - C$), we refer to Q alone as being calculated with any costs already being deducted. Uncertainty is modeled by defining a probabilistic distribution for the values of Q, D, and C. For example, a given task could have a duration of 10 with 50% probability, a duration of 4 with 25% probability, and a duration of 20 with 25% probability. As described above in Section 3.1, we model the possible quality outcomes of T_i as $\{Q_{i1}, Q_{i2}, \ldots, Q_{ik}\}$, and $\{D_{i1}, D_{i2}, \ldots, D_{il}\}$ as the possible duration lengths of T_i. Note that the time constraints for when task T_i can be executed, or its time Window W_i, is inherited from higher levels of the constraint tree as defined above.

In c-TAEMS, interrelations between Tasks and Subtasks are also be quantified through Quality Accumulation Functions (QAFs). QAF's are assumed to be of three forms: min, max or sum. In a min QAF, the total added quality is taken to be the minimum of all subtasks – it could be thought of as a logical AND relation between tasks. In a max QAF, the quality is the maximum value (or the logical

OR), whereas in a sum QAF the quality is the sum of the expected quality of all subtasks. These subtasks can then be further subdivided into additional levels of subtasks, each level with an associated QAF. Explicit interrelations between Tasks and Subtasks not inherited from higher levels of the tree can also be quantified as a non-local effect (NLE). Hard constraints of this type are defined through the primitives *enable* or *disable*. For example, to capture the constraint that one task must occur before another, one would add an enables relation between the two tasks in c-TAEMS. To capture the requirement that two tasks (or subtasks) should not both be performed, a disables relation between the two tasks would be added. Finally, soft constraints are modeled through *facilitates* and *hinders* relationships. For example, if the team prefers, but does not require, that one task be executed before another a *facilitates* relationship would be introduced which would increase the resulting quality if the relation is respected.

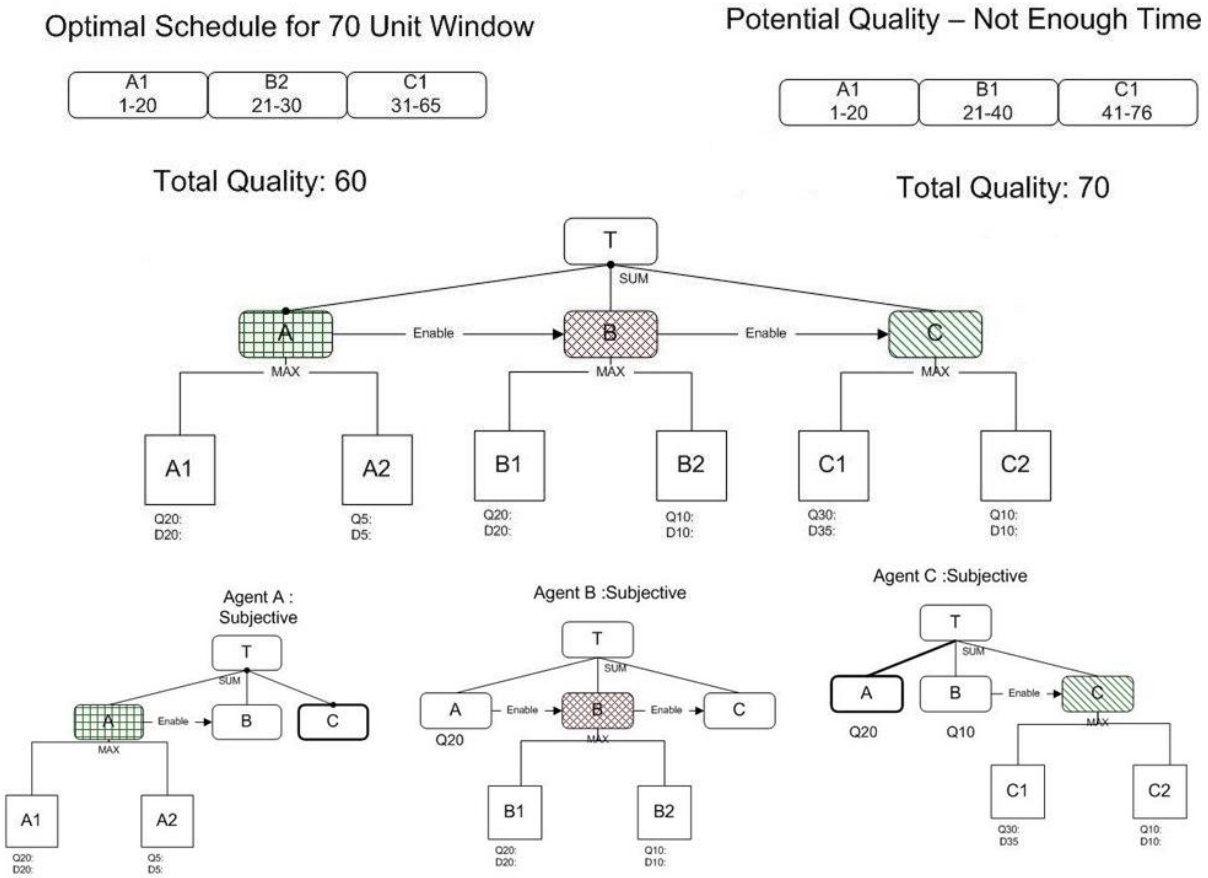

Figure 1: A sample c-TAEMS Scheduling Problem (global view, middle) with 3 agents (3 subjective views on bottom).

Figure 1 presents an example of a scheduling problem instance described in c-TAEMS. In this example, agents A, B, and C must coordinate their actions to find the optimal schedule for a global task T. Task T has three subtasks (A, B, and C) and these tasks are joined by a sum relationship. There are enable relationships between these tasks and thus they must be executed sequentially. In this example, an optimal schedule would be for A to schedule method A1, B to schedule B2, and C to schedule C1. However, assuming only 70 time units are available for all three tasks, there is insufficient time for that schedule. As such, one of the agents must sacrifice scheduling its method with the highest quality so the team's quality will be maximized. The team will lose 15 units of quality if A does not schedule A1, 10 units of quality if B does not schedule B1, and 20 units of quality if C does not schedule C1. Thus, B chooses B2 so that methods A1 and C1 can be scheduled.

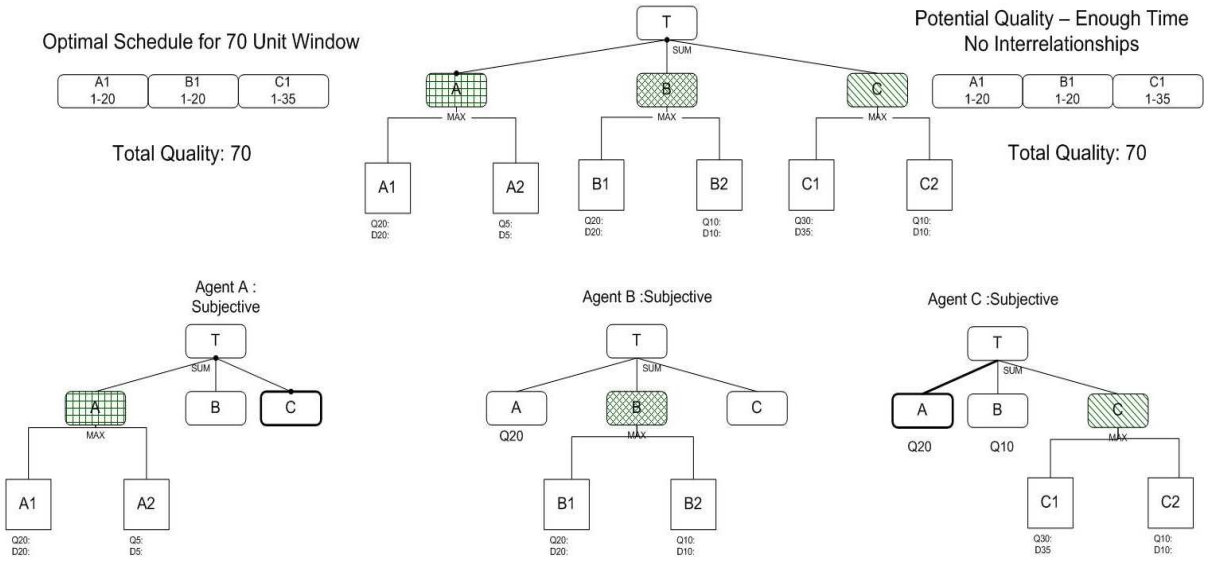

Figure 2: Modified c-TAEMS Scheduling Problem with NLE relationships removed. Note agents are now free to schedule tasks independently.

The impact from adding information in cases with NLE relationships can often be easily quantified. As these constraints must be explicitly formulated, their potential impact can often be easily detected. Conversely, the removal of these constraints may signify that agents can safely make local decisions without impact on other agents. For example, Figure 2 contains the same basic c-TAEMS structure from Figure 1, but with the NLE enable statements removed. Note that all agents can independently schedule their methods, and every agent can greedily choose the subtask with the highest quality (A1, B1, and C1) at the beginning

of the time window.

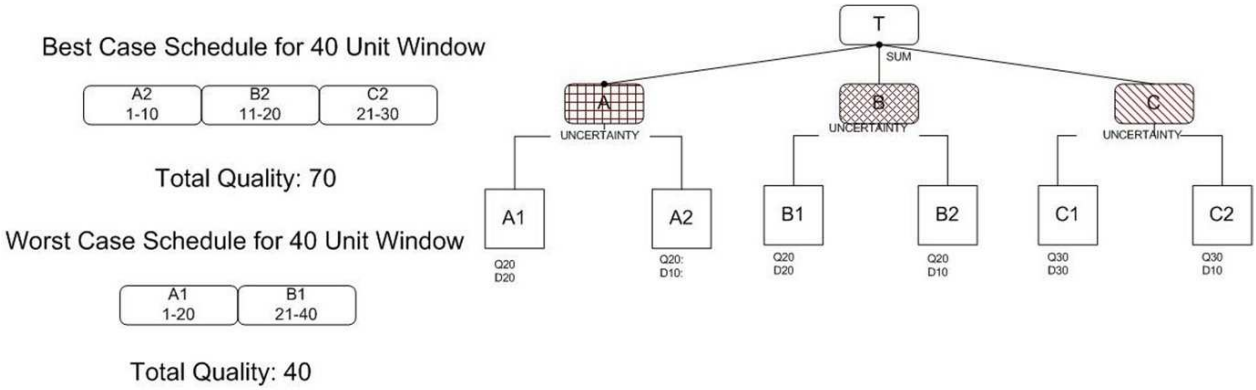

Figure 3: The impact of QAF constraints in the c-TAEMS structure (uncertainty found within only one agent).

Adding information can be equally significant in cases where task relationships are constrained based on their QAF relationships, especially in cases where uncertainty exists. This may even be true when considering the constraints within one agent. For example, Figure 3 depicts one agent's scheduling constraints as visible from its local view. Note that in this example, tasks A, B, and C contain only one possible method, but there is uncertainty in the duration of these methods. Furthermore, in this example, we assume no explicit constraints (NLE's) exist, and this agent is free to schedule from among tasks A, B, and C within a 40 time unit window. However, due to the task uncertainty, this one agent may not have enough time to schedule all of these tasks. For example, assuming task C lasts for 30 time units (option C1), only 10 additional time units remain to potentially execute methods A and B. Furthermore, if either of these methods requires 20 time units (options A1 and B1), they cannot be scheduled. In cases of this type of uncertainty, we assume agents schedule for the worst case, and the agent must only schedule methods A1 and B1. However, a human operator could provide information that tasks will take the shorter time, and potentially all three methods could be scheduled. For example, a person may have knowledge that tasks B and C will take the shorter times (options B2 and C2), allowing the agent to achieve a quality of 70 from executing all three methods. Thus, in this and similar problems the CA system must identify the importance of querying the human user for additional information.

Again, modifying the c-Taems structure, even slightly, can change the importance of these types of task uncertainty. Figure 4 contains the same c-Taems

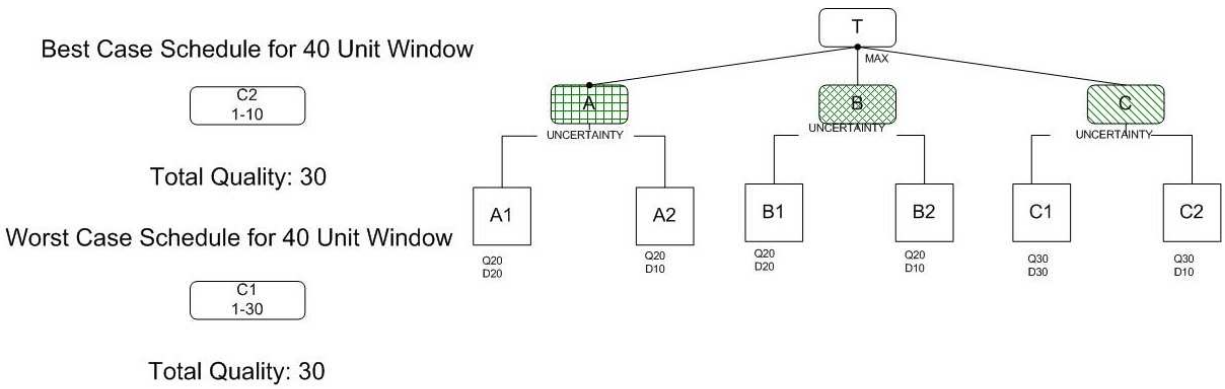

Figure 4: A modified c-TAEMS structure from Figure 3 where QAF constraints are relaxed.

problem, with a max QAF at the problem root instead of a sum. Here, the agent cannot execute all three tasks A, B, and C, and is free to greedily choose between the options. In this example, even assuming task C will take its maximal duration (30 units) sufficient time exists to schedule this method within the task's 40 unit time window. As a result, in this case the CA system should identify that there is no potential benefit from additional information, allowing the human operator to focus on other constraints.

4.2 Measuring Constraint Tightness in c-TAEMS

As the tightness measures described in Section 3.2 are based exclusively on local information, they have several significant limitations. First, the tightness definitions assume that a given task's window, $Window(T_i)$, can be measured. As previously stated, some localized communication may be needed to ascertain the quality and duration distributions of other agent's methods within W_i. Additionally, non-local constraints between task windows may exist, preventing measuring W_i from being able to be measured locally. An undesirable solution is to use communication between agents to exactly measure W_i. Specific to the c-TAEMS language, a task windows may also have a non-local event (NLE) that explicitly links several tasks, requiring the need to remeasure the actual window of T_i based on these constraints.

To address the second challenge, agents share a minimal amount of information to determine *if* these non-local constraints exist that may affect the actual window of T_i. We detail this check in Algorithm 1 as follows:

First, c-TAEMS agents check if their task T_i has any NLE constraints (line

Algorithm 1 Checking for Explicit and Partial Order Constraints (C-TAEMS File T, Task T_i)

```
1:  if Explicit(T_i) then
2:      return constraint-flag ⇐ set
3:  else
4:      for j = Level_1 to Level_{T_i} do
5:          if Tree-Constrained(T_i) then
6:              return constraint-flag ⇐ set
7:          end if
8:      end for
9:  end if
10: return constraint-flag ⇐ unset
```

1). As these constraints are explicitly defined, they can be easily detected. If this type of constraint is found, the agent must immediately flag this task as having non-local constraints that may impact its tightness measure (line 2). Otherwise, the agent must check for the existence of a partial order between the subtasks, $\prec$, higher up in the constraint tree that may impact T_i. This check must be done from the current level in the constraint tree ($Level_{T_i}$) up until the root node ($Level_1$). We assume that some minimal communication can be done to ascertain this information even if it is not locally available. This can generally be done within constraint trees that are aware of partial orders that may be affected by other portions of the tree. Specific to c-TAEM's representation, this is done by iteratively checking for any higher level QAF constraints such as a sum QAF that could affect its actual window. Assuming this type of constraint is found, the agent sets a constraint flag (line 6) for this task. Otherwise, the flag remains *unset* until it is finally returned at the end of the Algorithm in line 10. Note that in lines 2 and 6, the algorithm immediately ends (returns) once an explicit or higher level constraint is found that *may* affect the task's actual window.

As previously described at the end of Section 3.2, the locally measured value for W_i may not correspond to its actual value. The task's constraint flag described here is critical for making this determination. When this flag is left *unset*, the locally measured value W_i represents the actual value. In these cases, if the task's tightness value is less than one, agents can definitively assume their task is underconstrained and cannot benefit from additional information. However, in cases where the tightness measure is greater than one, including cases where the value is assumed to be greater than one because the constraint flag is set, agents cannot definitively identify whether additional information will be beneficial. Nonetheless, in these cases, agents use machine learning approaches based

on local problem attributes to predict the value of information. We present the process of this second stage in the next section.

5 Learning the Value of Information

As the tightness measure only addresses which tasks will definitely *not* benefit from additional information, the next step in our approach is a machine learning model to suggest which remaining tasks *will* show the largest potential gain from added user information. In addressing this challenge, we built two machine learning models: a regression based model to predict a numeric value for added information from the human operator, and a classification model to classify a given constraint as potentially benefiting or not benefiting from additional information. Alternatively, within the classification model, quantitative categories can be created, such as High, Low, and Zero impact categories instead of binary Yes and No categories.

Each of these machine models has advantages and disadvantages in terms of building the final human-agent application. As our goal is to present the user those methods worthy of her attention, we need to best estimate the utility gain that the user can potentially add in relaxing a given constraint. In general, regression models predict the expected value of a certain parameter based on the learned model. Here, the regression model returns a value that predicts the potential gain, or *quantifies* the gain from adding information about that subtask. This numeric value could potentially be used to formally model the cost versus gain analysis for every constraint[3]. The final list of constraints could be ordered based on this tradeoff and presented to the user. While this may sound ideal, regression models in general often have some model error, and we report on the margin of error in our model in Section 6. Thus, any order presented to the user is unlikely to be precise.

The advantage of decision tree learning approaches is their ability to classify a problem instance as belonging to qualitative categories. Instead of *quantifying* the value of added information, this model can *classify* if information will either help for a given constraint in the Yes/No two category decision, or belongs to High, Low, or Zero categories. In many cases, it may be impossible to measure the exact utility gain for a given constraint, especially as people typically operate with bounded rationality [21]. In these cases, instead of presenting a model that

[3]Note that the cost component of the application is not addressed in this paper. We assume work by Sarne and Grosz [23] can be used model this component.

explicitly weighs between costs and gains, a model may be preferred that generally indicates certain constraints are important. For example, referring back to Figure 1, constraints would be colored green if information was predicted to be less important while high importance constraints would be colored red. The advantage to the 3 category problem is that it provides a middle-ground between the regression and 2 category models. Here, we have some relative quantity of information worth (High versus Low versus None) which may be useful based on the specifics of a human-agent application under consideration. Specifically, High categories could be trained for which the estimated gain is estimated to be at least the cost of interacting with the user. For this reason, we also trained two types of decision tree classification models and report on the success of these approaches in Section 6 as well.

The general methodology for creating the machine learning models was to break each of the problems in the c-TAEMS training set into two variations – the base problem with constraint uncertainty in the problem's methods (or its basic subtasks as generally defined in Section 3.1), and the relaxed constraint after the user communicated information and the uncertainty was removed. We generated offline "what if" queries to obtain what the potential gain from information was in the latter case. It is important to stress that these "what if" queries were only used for training this machine learning model and not to be used during task executing as were done in previous approaches [22, 23, 25, 27]. The goal of this methodology was to train a model that can estimate what types of problem instances involving quality and duration uncertainty are most likely to benefit from the user's attention. During task execution, no resource intensive "what if" queries are used as the model has already been created to predict the expected utility for all types of constraints. Furthermore, as agents do not send constraint information during execution, no performance deterioration is possible because too many constraints were sent [4, 20].

The procedure we adopted for training the machine learning model is outlined in Algorithm 2. For each of the X training problems (line 1 in the algorithm), we first measure the schedule quality as generated without any additional information (line 2). In the experiments reported in the next section, we used a previously tested c-TAEMS centralized scheduler [30] under the assumption that uncertainty in problems would result in the worst case outcome. Thus, in the first set of problems, we assumed method's constraints would yield the lowest quality, and in the second problem set the methods would take the maximal time. We then computed what the team's utility would be if we could relax that assumption, and the user

could provide information about each of the basic subtasks (called methods in c-TAEMS) with uncertainty. As such, we assumed the user could definitively provide information that the method would yield the highest possible quality, or take the shortest time (line 4). Next, we stored this information into a table along with a vector of the problem's specific parameters (e.g. problem parameters such as tightness, number of agents, number of task, methods, local NLE's, maximal duration, and quality) and entered this information into the table, Table (line 5).

Algorithm 2 Training the Information Model(Problem Set of Size X)

```
1: for k = 1 to X do
2:    Without ⇐ Utility(Problem_k)
3:    for j = 1 to Num(Constraints(Problem_k)) do
4:       With ⇐ Utility(Problem_k,j)
5:       Table[k][j] ⇐ (With) - (Without)
6:    end for
7: end for
```

In computing the value of added information, we adopted a highly optimistic approach for the value of Utility($Problem_k$,j) that makes several assumptions: a) the person being contacted actually has information about the subtask j, b) the person will have information that will help the team in the maximal possible way by informing the agents that the task will have the highest possible quality or take the shortest duration, and c) all different types of constraint information can be provided be equal (constant) cost. Despite this oversimplification, this approach was useful for identifying the maximal theoretical gain (upper bound) for the utility gain that could be gained through added information. Additionally, it is important to stress that the "what if" training queries are sent one at a time. This was done to prevent sending the scheduler we used [30] too many queries which might prevent it from finding the optimal schedule [4, 20].

6 Experimental Results

In order to implement the machine learning models in Section 5, we used a problem generator created by Global Infotech Inc. (GITI) for the purpose of generating c-TAEMS problems within the framework of the COORDINATORS DARPA program[4]. We created two sets of 50 problems (or a total of X=100 as the number of problems in line 1 of Algorithm 2) where c-TAEMS parameters such as

[4]http://www.darpa.mil/ipto/programs/coordinators/

the number of agents (between 5 and 10), the number of tasks to be scheduled (between 25 and 100), the number of methods (between 50 and 500), the hierarchical structure of the tasks (randomized QAF values), the number of NLE relationships between tasks (between 0 and 15), the range of values for task duration (between 2 and 60) and quality (between 2 and 20) were randomly generated. In the first set of problems, we generated problems with uncertainty in quality distributions, while keeping the duration parameter deterministic. In the second set of problems, we created problems with uncertain durations, while keeping the quality parameter deterministic. Note that these 100 total problems represent a small fraction of the total number of the thousands of problem permutations the GITI scenario generator could create. A person could potentially communicate information about any given method within a problem subtask. We found that the 100 c-TAEMS problems used in this problem set contained over 5000 methods with uncertainty where the user's information might be helpful.

We found that only a small percentage of subtasks had any benefit from adding this type of constraint information. While each of the 100 training problems did benefit from added information, less than 30% of the basic subtasks (c-TAEMS methods) with uncertainty within these problems benefited from additional user information. For example, in the duration problem set, only 722 of the 2808 methods benefited from any additional information. Similarly, in the quality problem set 711 of 2399 methods benefited from this information. Thus, in this domain, finding the constraints where user information is beneficial is akin to "finding a needle in a haystack". Clearly, naive methods that query every constraint are not appropriate, especially if there is a cost associated with interacting with the human operator.

Nonetheless, we found strong support for the usefulness of the quality and duration tightness measures in identifying the cases where information definitely did *not* help. For example, of the 1360 quality tasks with a tightness less than one, only 7 (about 0.5%) benefited from additional information[5]. As nearly half of the 5000 constraints with uncertainty had tightness measures less than one, this measure was effective in allowing agents to filter away many constraints.

We used the Weka machine learning package [31] to train and evaluate the value of information in the remaining cases. Recall from Section 5 that we pro-

[5]It is not clear what significance, if any, these 7 cases have about the tightness measure we present. Despite the general effectiveness of the solver we used [30] it was not always able to find the optimal solution as these problems are still NP-complete. It seems that in these 7 cases, the added information reordered the constraints sent to the solver in such a way that it was able to find a better solution. Thus, this anomaly seems to be more connected to the inner workings of the particular solver than the tightness measure presented.

posed creating three machine learning models: a regression model, a classification model based on 2 information categories (Yes/No impact of information) and a 3 category information classification task (High, Low, and Zero impact). In implementing these models we used the M5P algorithm to form the regression model, and C45 decision trees (J48 within the Weka implementation) for the classification models. While we present results from these learning algorithms, other learning algorithms could have been used. We did, in fact, train models based on Bayes Networks, Neural Networks, and SVM models. However, we found the results from these algorithms to be nearly identical with those we present. In all cases, we performed 10-fold cross validation to evaluate the results.

We considered four different attribute sets for creating the machine learning models to predict the value of added constraint information. These attribute sets range from one that requires all local and general problem information to one requiring no information as follows:

- **All Information** - The attributes in this model were based on all information agents have about their problem. This included the local tightness measure we presented, in addition to c-TAEMS information about tasks and methods, their potential durations and quality values, and general problem information such as the number of agents and the total number of tasks and methods to schedule. Note that this model requires the most communication as we assume local agents do not have access to this global information. Still, no information about the exact values of constraint uncertainty needs to be communicated in this model.

- **Tightness with Flag** - This model used the tightness measure as described in Section 4.2. As described at the end of Section 4.2, a constraint flag was also needed to check if any explicit (NLE) constraints existed, or if higher level constraints, such as a sum QAF, existed that could impact that task. Note that this model required communicating only a very limited amount of local information to set this flag.

- **Tightness without Flag** - This model used the tightness measure as described in Section 4.2 but without the constraint flag described in that section. Our hypothesis was that the tightness measure will be accurate only in cases where this flag was set to false. As a result, we expected the model's accuracy to drop once this flag was omitted.

- **Naive Model** - This naive model simply classifies a problem based on the

Table 1: Comparing the accuracy (the mathematical correlation) in the regression trained model for duration and quality categories.

	Duration	Quality
All Information	0.60	0.73
Tightness with Flag	0.55	0.59
Tightness without Flag	0.22	0.50
Naive Model	-0.06	-0.04

majority of instances in the training set. As nearly 70% of the constraints in the training set never benefited from information, this model always predicts the value of information was 0 in the regression model, and in the decision tree models classifies all constraints as belonging to the category where information does not help.

6.1 Quantifying the Value of Information

We first focused on using a regression based model to attempt to quantify the value of added user information in the remaining constraints. We began with this model as it can ideally quantify the value of added information, something that is typically needed in human-agent theoretical decision frameworks [5, 9, 10, 8, 22, 23]. Using Weka [31], we trained and evaluated the regression based model and present the results of this model in Table 1. In this regression model, the results represent the correlation between the value of information predicted by this model versus the actual utility gain from adding this information as logged during the training period (see Algorithm 2). Ideal results would be a 1.0 statistical correlation, while a correlation of 0 indicates no success in the model. The first column presents the results from the problem set with duration uncertainty, and the second column presents the results from the corresponding problem set with quality uncertainty. Note that this model yielded an average correlation between the predicted and actual values of 0.60 and 0.73 when all local information was transferred. This value dropped to 0.55 and 0.59 with only the tightness and constraint flag information (as defined at the end of Section 4.2). As expected, the model's accuracy dropped significantly, most noticeably in the duration category, once this flag information was removed. Nonetheless, in all cases, the learned model produced significantly improved prediction accuracy above the zero correlation level found within the no communication category. Thus, while these results do leave some room for improvement, they do show the success of the tightness measure even without other local information.

6.2 Classifying the Value of Information

As the regression model did, in fact, not successfully predict the value of information in all cases, we also considered a two category classification model (Yes/No categories) to more generally classify constraints as instances where information does or does not help. This model is more appropriate for this type of task. When presenting the results from the classification model, we first present results about the total accuracy and recall of the model. We refer to the total percentage of problems correctly classified as the model's total accuracy. In this dataset, over 70% of the constraints do not benefit from additional information. Thus, even naively categorizing all constraints within this category results in a relatively high total accuracy of the model. However, as the goal is to effectively find all constraints where information will help, this approach will find none of the desired instances. As a result, we must study what percentage of the desired problems, where information *will* help, were correctly returned. We use the recall measure to quantify how many of these instances were found out of the total number of these instances in the dataset.

Table 2 presents the accuracy and recall results from this two category model. In both the problem sets with duration and quality uncertainty, the model using all local information had the highest recall in finding the constraints where information would help (a recall of 0.61 of these constraint in the duration set, and 0.7 in the quality set). Note that the tightness measure alone was sufficient to achieve similar results (and even had a slightly higher overall accuracy in the duration problem set) so long as the constraint flag was also sent. As previously described in Section 4.2, agents need to identify whether their localized task window has any other constraints that may need to be considered. Having even a binary flag (do these constraints exist or not) allows agents to definitively use the tightness measure in cases where these constraints do not exist. However, agents that lacked this information were not able to guarantee that their local tightness window was in fact underconstrained. Thus, the accuracy of the model, and the recall of the desired information dropped precipitously. Note that the model accuracy without this flag dropped from 79.95% to 70.36% in the duration problem set, and from 83.83% to 74.15% in the quality problem set. Similarly the system's recall dropped from 0.54 to 0 (no cases found!) in the duration set, and from 0.56 to 0.34 in the quality set.

Finally, we studied the three category classification task. The advantage to this model is that it incorporates the qualifying aspects of the previous two category

Table 2: Comparing the overall accuracy and number of high information instances found within a 2 category decision tree model.

	Duration Accuracy	Duration Recall	Quality Accuracy	Quality Recall
All Information	79.16%	0.61	85.29%	0.70
Tightness with Flag	79.95%	0.54	83.83%	0.56
Tightness w/o Flag	70.36%	0	74.15%	0.34
Naive Model	70.36%	0	74.29%	0

Table 3: Comparing the accuracy and number of high information instances found in a 3 category decision tree model with *duration* uncertainty.

All information	Classified High	Classified Low	Classified Zero	Accuracy	Recall
High	175	49	118	73.11%	0.512
Low	80	85	204	73.11%	0.23
Zero	92	102	1494	73.11%	0.885
Tightness with Flag					
High	222	4	116	74.57%	0.649
Low	113	0	256	74.57%	0
Zero	120	1	1567	74.57%	0.928
Tightness w/o Flag Naive Model					
High	0	0	342	70.36%	0
Low	0	0	369	70.36%	0
Zero	0	0	1688	70.36%	1

decision model, while still having various utility thresholds that will be likely needed by some human-agent models [5, 9, 10, 8, 22, 23]. Here, we divided the training data into a High category where information about that constraint improved the team's schedule quality by 10 or more units, a Low category where constraint information helped less than 10 units, and a Zero category where information did not help. Note that the threshold of 10 units for the High category is not fundamental to the model, and can be modified based on the needs of a given application. For example, say using previous approaches [23] we can measure that the cost of obtaining information from the user is 7 units. We can change this model such that the High category represents constraints that have an added utility gain of *at least* this amount and in these cases the cost of interrupting the user is outweighed by the gain. Thus, despite the qualifying nature of this model, it could be used to definitely decide when to query the user for information.

The results of this experiments from the duration set are found in Table 3, and those from the quality experiments are in Table 4. Within these tables, we present the classification confusion matrix, often presented in multi-category classifica-

Table 4: Comparing the accuracy and number of high information instances found in a 3 category decision tree model with *quality* uncertainty.

All information	Classified High	Classified Low	Classified Zero	Accuracy	Recall
High	135	34	50	81.37%	0.62
Low	34	250	219	81.37%	0.5
Zero	45	141	1900	81.37%	0.91
Tightness with Flag					
High	129	13	77	79.77%	0.59
Low	104	157	242	79.77%	0.31
Zero	66	66	1954	79.77%	0.94
Tightness w/o Flag Naive Model					
High	0	0	219	74.29%	0
Low	0	0	503	74.29%	0
Zero	0	0	2086	74.29%	1

tion problems. We plot the number of instances found within a given category (the diagonal of table) as well as the number of instances misclassified per category. Not all misclassification errors are necessarily equally important. For example, misclassifying a High problem as Low, or a Low problem as High is likely to be less problematic than classifying a High problem as Zero. Note that within the quality experiments involving the tightness information with the constraint flag (the middle portion of table 4), only 157 of 503 Low instances were classified as Low, but another 104 of these instances were classified as High. This distinction can be quite significant. The recall of the this category alone (Low classified as Low) is only 0.31, however, if we view classifying High either as High or Low as being acceptable, the recall jumps to 0.52. Similarly, in the duration experiments, the recall of the same category (Low classified as Low) was zero (none properly classified as Low). However, 113 of the cases were classified as High.

6.3 Exploring the Tradeoff between Precision and Recall

As the machine learning models never achieved a recall near 100%, we considered creating models which were biased towards categorizing a task as benefiting from information. While this bias will result in a higher recall for this category, it will generate false positives that will lower the overall accuracy of the model. This type of approach is a classical tradeoff between a model's recall and precision level. In general, one can improve any model's recall by biasing the model

to accept a certain problem as belonging to that category. In the trivial case, any model's recall can be made 1.0 by having it always classify that instance as belonging to the desired category. However, in this case, the model's precision is likely to be quite low. In this problem, recall refers to how many constraints were found where information helped and precision refers to what percentage of these constraints was correctly classified.

In this domain, this tradeoff would likely be useful if the human user is able to be prompted Q times to add information, when Q is greater than the actual number of tasks that can benefit from added information. Alternatively, this approach may also be useful if the known cost from interrupting the user is relatively low. For example, if the cost of prompting the user is 1 unit, we should be willing to ask several queries for information so additional High instances (each worth 10 utility units) can be found. To train this model, we followed the previously developed MetaCost algorithm [3]. Metacost is one method for creating cost-sensitive classification which allows for easily setting a cost matrix for biasing the classification of a data instance as belong to one category versus another. We used the Metacost implementation found within the Weka [31] learning package.

We did find that the weight biases created through the MetaCost algorithm were extremely effective in increasing the system's ability to find the tasks worthy of prompting the user for information, albeit at a cost of false positives that reduced the overall accuracy. To explore this point, we used the MetaCost function to apply different weights for falsely classifying a constraint where information was useful (Yes) as belonging to the non-useful category (No). The base weights, or unbiased classification, will have equal weighting for these categories (1 to 1 weight). We found that as we increased these weights, we obtained a progressively higher recall from the desired Yes category, but at an expense in overall reduction of model accuracy, and thus a lower precision value. Table 5 presents the results of this approach from the quality problem set with a two category classification model with agents sharing all local information (compare these results to the right side of the top line of Table 2).

In some simple classifiers, the tradeoff between precision and recall is extreme with a high value of recall resulting in a very low value in precision, and a high value in precision resulting in a very low recall. Here, however, even with this biased model, the recall and precision values were still moderately high in the extreme cases. For example, a 5 to 1 bias towards the Yes category found 609 of the 722 instances (or 0.84 recall), and a 50 to 1 bias found nearly all instances (0.99 recall). However the 50 to 1 bias yielded the lowest overall accuracy due

Table 5: Exploring the tradeoff between higher recall of desired results (Yes instances found), and false positives and negatives within a 2 category decision tree model with quality uncertainty.

Weight	Total Accuracy	Found	Not Found	Recall	Precision
1 to 1	85.29%	502	220	0.70	0.72
2 to 1	84.54%	538	184	0.75	0.68
5 to 1	82.66%	609	113	0.84	0.62
10 to 1	79.84%	655	67	0.91	0.57
50 to 1	70.26%	717	5	0.99	0.46

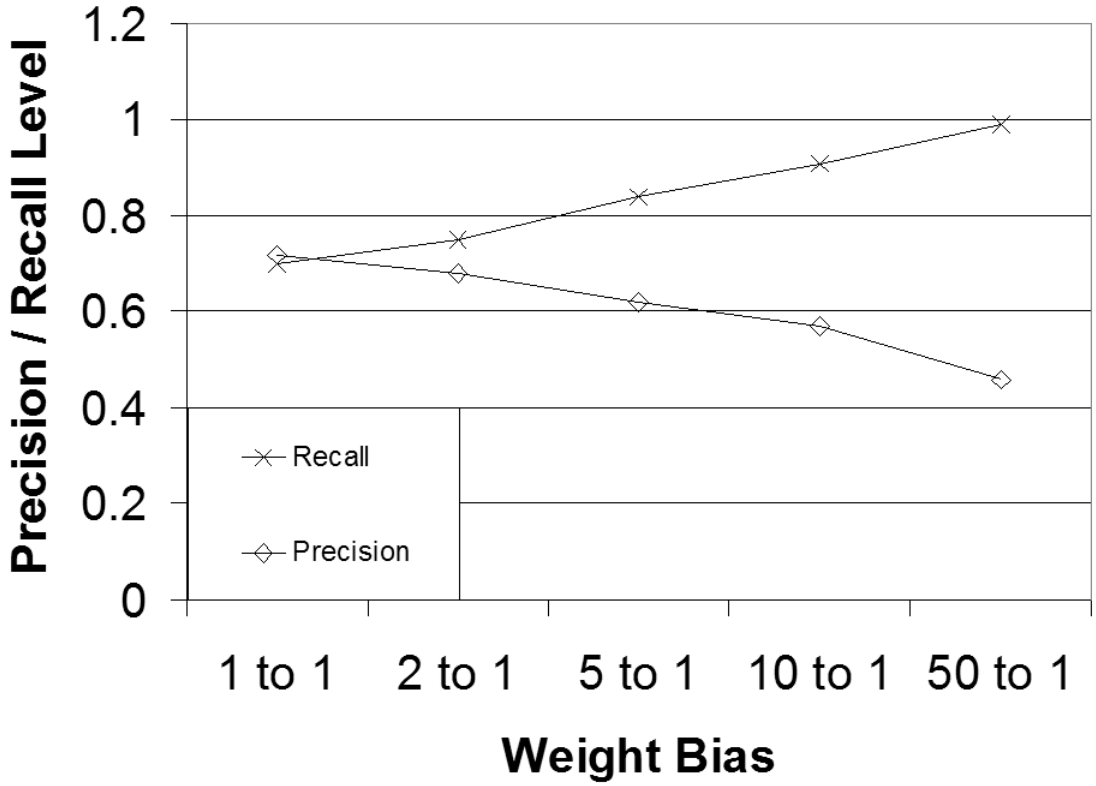

Figure 5: A graphical tradeoff between the recall and precision for finding constraints where user information will help with varying weight biases.

to false positives (0.46 precision compared to 0.62 precision with a 5 to 1 bias and 0.72 with no bias). Nonetheless, even with this strong bias towards categorizing a constraint as benefiting from information (and thus achieving a nearly 1.0 value in recall), the precision still was a moderate 0.46. This indicates that the model was still able to definitively identify nearly half of the constraints as still not benefiting from additional information. At the other extreme we present the unbiased classification model, or a cost bias of 1-1. Note that in this case

both the recall and precision values were above 0.7. This demonstrates that even without any bias, the recall of the system was quite high, as one typically finds a recall value near 0 at this extreme. Figure 5 helps to visualize these results.

We also applied this approach to the two category duration problem set, as well as the quality and duration 3 category classification models. As expected, in all cases the cost biases were effective in increasing the recall of the categories where information helped, albeit at a cost of lower model accuracy. Thus, a system designer will likely choose some cost bias based on the actual costs for querying a specific user. This issue is explored in other work [23].

7 Discussion and Future Directions

In general, we found that the tightness measure presented in this paper was extremely effective in finding which constraints would *not* benefit from adding information. This was done by having local agents identify which constraints could be locally solved. While we applied this measure to c-TAEMS problems, this domain represents only one possible instance of the the generally defined the scheduling task in Section 3.1. Furthermore, the tightness measure in Section 3.2 was generally defined, allowing it to be applied to other domains as well. It reasons that other scheduling and planning problems have the same underconstrained and constrained groupings of problems typically found within constraint problems [1, 2, 16, 25] and many constraints in these problems can also be solved without further information. Additionally, c-TAEMS problems can be directly mapped to the more general Distributed Constraint Optimization Problem (DCOP) formalization [28]. This suggests that the presented approach should also be applicable to scheduling problems based on the DCOP formalization [13]. Thus, future work should focus on directly applying the tightness measure to identify which constraints in other domains can be locally resolved without additional information.

We believe several other research directions are worthy of study. First, we found that decision trees were, overall, very effective in quantifying the expected impact of adding information, and thus were helpful in recommending if the user should be contacted for additional information. In contrast, previous work on adjustable autonomy [24] found decision trees were ineffective in enabling agents to make autonomous decisions. It seems that the difference of results stems from the very different tasks considered. The previous work used the learned policy

from decision trees to enable agents to act independently of people within their team. As a result, their scheduler system made several critical errors (such as canceling group meetings and volunteering people against their will for group activities) by overgeneralizing decision tree rules. In contrast, our support system never tries to make autonomous decisions, and instead takes the support role of recommending what constraint(s) a person should focus on. This distinction may suggest the need to create different types of learning models for different agent-human tasks.

Also, further work is necessary to identify general attributes where information definitively *does* add utility. This paper's conclusion is that local information is sufficient for guaranteeing that a given problem is not constrained, and thus information will *not* help. However, the disadvantage to the exclusively local approach we present is that agents are less able to consider the full extent of all constraints within the problem. Because of this, we believe this approach was less effective in finding the cases where information would *definitely* help.

We have begun to study several of these directions in parallel to the work we present here. Along these lines we have studied how the tightness measure can guide agents if they should communicate all of their constraints to a centralized Constraint Optimization Problem (COP) solver [20]. The COP solver would then attempt to centrally solve the constraint problem after receiving all constraints from all agents. To address what agents should communicate, each agent viewed all of its constraints as belonging to only one task window, with all subtasks falling within this window. We found that the resulting tightness measure created three classic clusters of constraint interactions: under-constrained, constrained, and over-constrained with a clear communication policy emerging based on this measure. Under-constrained problems had low tightness and could locally be solved without sending any constraints to the COP solver. Constrained problems had a medium tightness value, and most benefited from having every agent send all of its constraints. Problems with the highest tightness value were the most constrained. In fact, these problems had so many constraints that agents sending all constraints flooded the COP solver, which was not able to find the optimal solution. In these problems, agents were again best selecting communication approaches that sent fewer constraints.

Finally, it is important to note that these research directions are complementary. We foresee applications where different tightness measures are applied to filter and predict different characteristics. Say, for example, a domain exists where agents could send constraint information freely. As we have previously

found, sending too much information can prevent centralized problem solvers from finding the optimal solution [20]. One solution might be to apply the local tightness measure we present here to filter cases where information definitely will not help, and then have agents send all remaining constraints. This more limited set of constraints might be most manageable than the original set. We are hopeful that this work will lead to additional advances in this challenging field.

8 Conclusion

In this paper we presented an approach to estimate the expected utility gain from adding user information to agents within distributed scheduling problems. Agents used information about their constraints to predict whether adding constraint information will help the team increase its utility. The significance of this work is its ability to successfully enable agents to find which types of constraints will most likely benefit from additional human information, without sharing significant information about other agent's constraints or through resource intensive queries required in other approaches [22, 23, 25, 27, 29]. In our two-stage approach, agents first initially assess which constraints will *not* benefit from any additional information by using our general *tightness* measure. After agents shared a very small amount of information regarding the problem's structure, they were able to locally resolve a large percentage of the problem's constraints without any additional information from other agents' or their human operators. In the second stage, agents use machine learning models to estimate the value of a user's constraint information on the remaining constraints. These models were trained offline, allowing agents to predict the value of information during task execution without cost. We present the results of a regression based model which quantifies the estimated utility gain for added information to each of these constraints, and decision tree approaches that classify if a constraint should be categorized as benefiting from information or not.

We studied the effectiveness of this two-stage approach within the c-TAEMS distributed scheduling domain. We found that the non problem-specific tightness measure in the first stage was extremely effective in finding where additional information about constraints would *not* be helpful. Both of the machine learning models in the second stage were moderately useful in identifying where information *would* be helpful. Finally, we presented several possible future directions of study, including discussions regarding the generality and possible applications

of this approach in other domains.

References

[1] S. A. Brueckner and H. V. D. Parunak. Resource-aware exploration of the emergent dynamics of simulated systems. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multi-agent systems*, pages 781–788, 2003.

[2] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.

[3] P. Domingos. Metacost: a general method for making classifiers cost-sensitive. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 155–164, 1999.

[4] E. H. Durfee. Practically coordinating. *AI Magazine*, 20(1):99–116, 1999.

[5] M. Fleming and R. Cohen. User modeling in the design of interactive interface agents. In *Proceedings of the seventh international conference on User modeling*, pages 67–76, 1999.

[6] M. S. Fox, N. Sadeh-Koniecpol, and C. Baykan. Constrained heuristic search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 309 – 316, 1989.

[7] B. J. Grosz and S. Kraus. Collaborative plans for complex group activities. *Artificial Intelligence Journal*, 86(2):269–357, 1996.

[8] E. Horvitz, C. Kadie, T. Paek, and D. Hovel. Models of attention in computing and communication: From principles to applications. *Communications of the ACM*, 46(3):52 – 59, 2003.

[9] E. J. Horvitz, J. S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2:247–302, 1988.

[10] E. J. Horvitz and A. C. Klein. Utility-based abstraction and categorization. In *UAI*, pages 128–135, 1993.

[11] S. Kambhampati. A comparative analysis of partial order planning and task reduction planning. *SIGART Bulletin*, 6(1):16–25, 1995.

[12] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, 2004.

[13] R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 310–317, 2004.

[14] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 438–445, 2004.

[15] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[16] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic "phase transitions". *Nature*, 400(6740):133–137, 1999.

[17] K. L. Myers and D. E. Wilkins. Reasoning about locations in theory and practice. *Computational Intelligence*, 14(2):151 – 187, 1998.

[18] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR)*, 20:379–404, 2003.

[19] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *IJCAI*, pages 968–975, 1999.

[20] A. Rosenfeld. *A study of dynamic coordination mechanisms*. PhD thesis, Bar Ilan University, 2007.

[21] S. J. Russell. Rationality and intelligence. In C. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 950–957, 1995.

[22] D. Sarne and B. Grosz. Estimating information value in collaborative multi-agent planning systems. In *AAMAS'07*, pages 227–234, 2007.

[23] D. Sarne and B. Grosz. Sharing experiences to learn user characteristics in dynamic environments with sparse data. In *AAMAS'07*, pages 202–209, 2007.

[24] P. Scerri, D. V. Pynadath, and M. Tambe. Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research (JAIR)*, 17:171–228, 2002.

[25] J. Shen, R. Becker, and V. Lesser. Agent Interaction in Distributed MDPs and its Implications on Complexity. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 529–536, 2006.

[26] D. Smith, J. Frank, and A. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):61–94, 2000.

[27] S. Smith, A. T. Gallagher, T. L. Zimmerman, L. Barbulescu, and Z. Rubinstein. Distributed management of flexible times schedules. In *2007 Intl conf on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2007.

[28] E. Sultanik, P. J. Modi, and W. C. Regli. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *IJCAI*, pages 1531–1536, 2007.

[29] K. Sycara, S. F. Roth, N. Sadeh-Koniecpol, and M. S. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[30] W. J. van Hoeve, C. P. Gomes, B. Selman, and M. Lombardi. Optimal multi-agent scheduling with constraint programming. In *AAAI/IAAI*, pages 1813–1818, 2007.

[31] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, June 2005.

[32] F. Yaman and D. S. Nau. Timeline: An htn planner that can reason about time. In *AIPS Workshop on Planning for Temporal Domains*, pages 75–81, 2002.

[33] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.