Estimating Information Value in Collaborative Multi-Agent Planning Systems

David Sarne, Barbara J. Grosz
School of Engineering and Applied Sciences
Harvard University, Cambridge MA 02138 USA
{sarned,grosz}@eecs.harvard.edu

ABSTRACT

This paper addresses the problem of identifying the value of information held by a teammate on a distributed, multi-agent team. It focuses on a distributed scheduling task in which computer agents support people who are carrying out complex tasks in a dynamic environment. The paper presents a decision-theoretic algorithm for determining the value of information that is potentially relevant to schedule revisions, but is directly available only to the person and not the computer agent. The design of a "coordination autonomy" (CA) module within a coordination-manager system provided the empirical setting for this work. By design, the CA module depends on an external scheduler module to determine the specific effect of additional information on overall system performance. The paper describes two methods for reducing the number of queries the CA issues to the scheduler, enabling it to satisfy computational resource constraints placed on it. Experimental results indicate the algorithm improves system performance and establish the exceptional efficiency—measured in terms of the number of queries required for estimating the value of information—that can be achieved by the query-reducing methods.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*; H.1.1 [**Information Systems**]: Systems and Information Theory—*Value of information*

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Incorporating autonomous-agents capabilities into planning and scheduling systems can significantly increase their power, especially in highly dynamic environments and settings in which multiple agents' schedules are being coordinated [19; 22, inter alia]. Agents can suggest alternative courses of action as well as negotiate conflicts. They can help achieve team objectives more effectively in such highly dynamic environments as first-response situations [21, 13], exploration of remote planets, cleanup of hazardous sites, and military conflicts [11], because they have the ability to efficiently process the rich set of information affecting the problem. By taking responsibility for changes in task schedules, they can free people involved in doing tasks from being unnecessarily overburdened, especially in times of crisis when plans need to be adapted to new circumstances [13, 11]. Situations characterized by agents being separated geographically, coordinating groups having limited communication bandwidth, and changes in events and physical environment being complex, dynamic and stochastic in nature typically preclude a complete global view of the problem. They thus require localized, but coordinated, planning and scheduling decisions [20, 21]. In the remainder of this paper, we use the term "automated scheduling agent" (ASA) to refer to autonomous-agent systems that support multi-agent coordinated scheduling.

ASAs must take into account various external facts that may influence choice of scheduling changes significantly. These include changes in the environment in which tasks are being carried out and the state of the people or other agents working on a distributed, coordinated task. Information beyond the system's limited knowledge of the environment may affect constraints, either relaxing them or imposing new ones that better reflect the actual situation. For example, a driver will see changes in weather conditions that may affect route selection when they occur, while an automated navigation system does not. A scientist can induce that a colleague will be going to the school play and thus be unavailable to meet simply because their children go to the same school. Human-agent interaction must be managed appropriately for the system to obtain such information without overburdening other participants, especially people [13, 19]. Obtaining system-external information is associated with costs, including communication costs and the degradation in task performance of the person being interrupted [5, 8, 10, 22]. An ASA needs to plan its interactions to maximize the difference between the value of the information being obtained (taking into account the probability the person being queried has this information) and the costs, deploying appropriate decision-making mechanisms [9, 4].

This paper addresses the problem of identifying the value of information held by a teammate on a distributed, multi-agent, mixed human and computer team. We focus on a scheduling task in which the computer agents are ASAs supporting people who are carrying out complex tasks in a dynamic environment. The ASA is responsible for coordinating modifications to the task schedule as the environment changes with as few interruptions of the person as possible, thus enabling the person to focus on actual task performance.

Although coordination itself is domain independent, determining the value of information requires scheduling knowledge; information is valuable to the extent it influences schedule changes, but that influence cannot be determined without task and schedule knowledge. The design of a "coordination autonomy" (CA) module within a coordination-manager system (as part of the DARPA Coordinators project, which aims to construct intelligent cognitive software agents able to assist fielded military units adapt their mission plans and schedules as their situations change) provided the empirical setting for this work, including both constraints on system design and a testbed environment. The overall coordination-system design separates the CA module from the *scheduler* which encapsulates all the scheduling-related expertise. As a result, the CA must consider not only interruptions of the person, but also

resource demands it makes on the scheduler through querying for schedule-expertise. Thus, a key challenge to CA-module design was to develop methods that would obtain information from the (human) participants in a distributed, coordinated task without interrupting them too frequently or at the wrong times, while simultaneously not consuming too many scheduler, communication or other coordination system resources.

The paper makes two major contributions. First, it presents a decision-theoretic algorithm for determining the value of information in a multi-agent context in which a person rather than a computer agent has the information in question and in a systems context in which the calculation of information-value is complicated by the need to invoke a separate systems module (in this case, the scheduler).[1] Second, it describes two novel methods for dealing with the computational resource constraints on the CA. The first method filters queries depending on whether there has been a change in relevant tasks in the schedule. The second translates the problem of generating queries into a game and then attempts to minimize the score. Experimental results demonstrate the effectiveness of the algorithm and that these two methods increase efficiency by significantly decreasing the number of queries required for estimating the value of information.

To provide context for the empirical results as well as for the methods we designed for determining information value, the next section of the paper briefly describes the *Coordinators* application domain and the basic architecture of the CA module. Section 3 discusses the principles on which the process of estimating the value of information is based, and Section 4 presents mechanisms for improving the efficiency of the scheduler-querying process. Experimental results are given in Section 5. Section 6 discusses related work, and Section 7 contains our conclusions.

2. IMPLEMENTATION BACKGROUND

The CA module and algorithms we describe in this paper were developed and tested in the Coordinators' application domain [21]. In this domain, the ASAs, called "coordinators", operate in a rapidly changing environment and are intended to help maximize an overall team-objective which is measured using a hierarchical objective function. Each ASA operates on behalf of its *owner* (e.g., the team leader of a first-response team or a unit commander) whose schedule it manages. The actual tasks being scheduled are executed either by owners or by units they oversee, and the ASAs' responsibility is limited to scheduling these tasks.

The quality of ASAs' schedules depends on knowledge of the environments in which their owners are operating, and in many cases the owners of ASAs may have more accurate information about the state of the environment. Owners in this domain may acquire relevant external information through various communication channels (formal and informal) they maintain with other owners. Typical sources of such information are occasional coordination meetings (e.g., used for reporting status of task execution), open communication the owner overhears (e.g. when leaving the radio open, one gets to listen to messages associated with other teams in the area) and direct communication used for coordination with other people acting on a joint task (throughout which an individual often learns informally about the status and existence of actions being executed by others). An additional source of task-outcome related information is an owner's accumulated experience.

In this domain, scheduling information and constraints are distributed. Each agent receives a different view of the tasks and structures that constitute the full multi-agent problem—typically only a partial, local one. Thus, the intended context of use of coordinators precludes central management. Scheduling problems must be solved distributively. As a result, each ASA needs to reason about changes in the timings of tasks with regard to their correlation with other ASAs' tasks and to adapt its owner's schedule accordingly. The time pressure of the intended applications (e.g., emergency response) requires that ASAs make decisions in real time, concurrently with their owner's (or their units) execution of tasks.

The ASA-owner relationship is a collaborative one, with the ASA needing to interact with its owner to obtain task- and environment information relevant to scheduling. The CA module is responsible for deciding intelligently when and how to interact with the owner for improving the ASA's distributed scheduling. It interacts primarily with a metacognition component of the ASA, which controls computational resources allocated to the CA and other ASA modules, and with the scheduler model of the ASA.[2]

The scheduler, which is the locus of the ASA's scheduling expertise, is responsible for task analysis, and thus is a resource whenever scheduling reasoning is required for evaluating the effect of changes in scheduling-problem settings on the system's expected performance. The separation of scheduling expertise in its own module is generally good system design, because many other ASA modules need to reason about similar aspects of the schedule. Furthermore, unlike such user-focused problems as estimating the probability a user knows relevant information, the user's confidence level in the information, and the costs associated user-interactions, which are naturally part of the CA, this scheduling expertise is system-focused. However, the mechanisms described in this paper are useful even in environments in which a CA would have scheduling expertise. In such settings, the CA could also deploy heuristics that take advantage of its knowledge of the problem structure.

The CA initiates interaction with the ASA when it requires scheduler resources or after interacting with the owner (for the purpose of transferring the information received to the ASA). Our basic CA architecture includes components that track the owner's interruptibility preferences and costs as well as managing interactions with the owner [17, 18]. The module is able to determine (future) times at which there is a high probability the owner will have new information that may affect system performance. For example, an owner is more likely to know the probability of success of a task to be done in the future shortly prior to the task's execution than further from its execution time. If the CA evaluates the expected benefit to be positive, then an interaction with the owner will be initiated. The mechanism and algorithms presented below enable the CA to efficiently activate the scheduler to estimate the value of a specific piece of task or environment information the owner has. The benefit of interrupting the owner is based on a comparison of this latter value multiplied by the (estimated) probability an owner has this specific information with the CA's estimation of the cost such an interaction will incur.

Coordinators ASAs represent planning and scheduling problems in cTAEMS structures [2] defining multi-agent hierarchical tasks with probabilistic expectations on their outcomes. For this paper the relevant elements of cTAEMS are methods, quality measures and inter-action dependency links, which we describe briefly.

Methods, which are actions executed by a single agent, are the atomic elements in cTAEMS. Figure 1 provides an example of a method definition. As shown, a method has multiple possible discrete outcomes that determine its possible durations and quality.

[1] The algorithm focuses on the "benefits" side of the decision making process that determines when to interact with a user. Specifically, we focus on estimating the expected value of the information the user may have. A complete decision-making mechanism would combine the work described in this paper with a method for determining the cost of interrupting the user and the probability the user has the necessary information [17, 4, 5].

[2] Several different architectures have been suggested for ASAs in the Coordinators domain [14, 12, 20]. All of them share the same core set of modules, including the scheduler and metacognition components.

Methods are usually constrained by "release times" (earliest possible start) and deadlines that are set by tasks higher in the cTAEMS hierarchy to which a method belongs, and inherited hierarchically. If a method violates its temporal constraint, it is considered to have failed and yields a zero quality. The probability of zero quality and the duration distributions are given in the "failure" portion of the method definition, as shown in Figure 1. The quality of execution of a method contributes to the quality of its parent task by means of a quality accumulation function (QAF), which defines how the qualities are aggregated up the hierarchy. For example, a q_{min} QAF for a task results in a quality equal to the minimum achieved by its subtasks. The functions q_{max} and q_{sum} are defined analogously. Only methods that are executed within their constraints accrue quality, and the overall objective is to maximize quality. The non-local effects links (NLEs) in cTAEMS indicate inter-node relationships such as enablement and facilitation. These abstract cTAEMS structures are used to represent real-world situations. For example, the situation in Figure 1 might represent a plan to go the store to buy wine as part of a larger plan for a dinner party. The time required for this activity might be 16, 20 or 24 minutes (depending, for instance, on the probability of running into a neighbor who wants to talk or the conditions of the sidewalk after a snowstorm). The quality of the schedule (12 or 15) depends on the brand of wine obtained. The zero quality might result if the shopper forgets his or her wallet, which may have a low probability, but will affect the rest of the dinner plan.

The hierarchial outcomes of cTAEMS structures may be translated into a simple tabular representation of outcomes and their probability, as given in the table to the right of Figure 1. Each cell in the table contains the probability associated with the outcome described by its row quality and column duration. This tabular presentation is particularly useful when we investigate methods for increasing the CA mechanism's efficiency in Section 4.

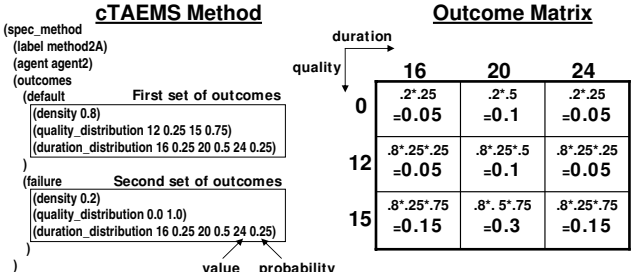

Figure 1: A typical method in cTAEMS

cTAEMS problems are known to be difficult scheduling problems and are not easily modeled and solved by traditional solvers for planning and scheduling [6]. In Coordinators, a variety of techniques for scheduler implementations have been used [6, 12, 20]. The value-of-information methods we present do not depend on the particular scheduler technology. The only functionality they require is the ability to produce a feasible schedule given changes in the cTAEMS problem and to be consistent (i.e., if the same problem arrives again then the same solution is supplied).

3. INFORMATION VALUE

In cTAEMS, the outcome of a method after it has been executed (or has failed) represents the contribution of performance of that action (its "quality") to the overall team effort weighted according to the method's NLEs and QAFs. Because execution outcome is probabilistic, if the CA can obtain more accurate information about the probability distribution of a method's outcomes before method execution is completed, it can help increase solution quality significantly. Environmental information relevant to method outcomes may be helpful in deciding among possible schedules. For instance, if an owner can tell ahead of time that the method its ASA has scheduled to be executed next will definitely fail (e.g., because the weather has changed and will prevent success), early notification of the ASA will allow it to change the schedule before the owner even begins executing this method. If the ASA notifies other owners' ASAs of this change, then the schedules of any other owners whose tasks are affected by this method's outcome (because their methods either are part of common QAFs or are enabled by NLE links originating from this method) also can be changed earlier, rather than waiting for the failure to occur. Without such external information, the ASAs may lose flexibility in choosing alternative schedules as well as valuable task-execution time, because they will only know method outcome at the end of execution.

In this section, we describe a Bayesian-decision-theoretic [16, 1, 3] algorithm for estimating the value of information an owner may have. The algorithm calculates possible method outcomes and associated consequences resulting from getting this information at different times; it weighs each possibility's contribution by the probability it will occur. The algorithm queries the scheduler iteratively to determine the value of information an owner may have. The CA does not know what information the owner has, only that the owner has (or may have) relevant information. Thus, unlike prior approaches that use decision theory to calculate the value of information [8, 10, 22], the algorithm must compute the value of information without knowing the actual piece of information (i.e., without knowing a priori which value the owner will provide). Furthermore, the algorithm handles the overall changes in schedule that may result when new information is obtained (essentially emulating the re-scheduling that would result), and it considers all possible outcomes of a method, not simply success or failure. This functionality is not included in the scheduler and is essential for updating the value-of-information appropriately.

3.1 Querying Principles

Throughout this section we will consider a scenario in which the ASA has scheduled method M to be executed next, and the potential outcomes of this M are given by the duration vector $D^M = \{d_1^M, \ldots, d_k^M\}$ and the quality vector $Q^M = \{q_1^M, \ldots, q_l^M\}$. The owner may be able to reveal (with some probability) that the actual outcome of M is (d_i^M, q_j^M), information not directly available to the CA. That is, prior to interacting with the owner, the CA has only the a priori probability $P(d^M, q^M)$ for each potential outcome as given in the method's definition (as illustrated in Figure 1). The value of the information (d_i^M, q_j^M) is the marginal improvement in the quality of the schedule after it is changed as a result of obtaining this information (at query time). That is, the value of information (d_i^M, q_j^M) is the difference between performance of the ASA with this information "a priori" (i.e., when it queries the owner), denoted $E[Q/access\_owner]$, and its performance when it obtains this information only at the end of execution of a method, denoted $E[Q/\neg access\_owner]$. The value is positive only if the system changes its action depending on (d_i^M, q_j^M), but is always non-negative.

In the ASA setting, to estimate the value of an interaction with the owner (i.e., to calculate $E[Q/access\_owner]$), the CA needs to weigh the value of each possible specific outcome according to the probability of this outcome, because (as noted above) even when the CA knows that the owner has certain information about a method's outcome, it does not know the outcome-value the owner knows. The algorithm presented below uses hypothetical ("what-if") queries, which include the original cTAEMS problem settings and the potential new information from the owner, to the scheduler to determine the value of a specific piece of information.

We will use the notation $S_t(T, I)$ to denote the best schedule the scheduler can produce at time t, given a cTAEMS problem

T and the additional information I (known at time t). The structure I may be empty, indicate a specific outcome (d_i^M, q_j^M) of the method M, or eliminate a specific outcome from the distribution of possible outcomes. We denote the value (quality) of a schedule $S_t(T, I)$ by $S_t(T, I).quality$. The expected quality of a schedule given updated-method-status information from an interaction with the owner is given by

$$E[Q/access\_owner] = \sum_{i=1}^{k}\sum_{j=1}^{l} S(T, (d_i^M, q_j^M)).quality P(d_i^M, q_j^M). \quad (1)$$

Calculating the value $E[Q/\neg access\_owner]$ is a non-trivial task. Although the owner is operating according to some schedule, the quality associated with this schedule cannot be used as a baseline, because it does not reflect the re-scheduling that can occur after a method's actual outcomes are known (or other outcomes are eliminated). This rescheduling may alter significantly the actual quality that is achieved. As an example, we consider a method M planned to be executed at time t, that a priori has four duration outcomes $\{d_1^M, \ldots, d_4^M\}$, with probability $P(d_i^M) = 0.25\ \forall 1 \leq i \leq 4$, and its actual duration outcome is d_3^M. In Coordinators, each agent receives an execution completion method at the time the method successfully completed its execution.[3]. At time $t + d_1^M$ the ASA will realize that the method did not complete execution yet because it did not receive an execution completion (EC) message; thus, outcome d_1^M is no longer valid for this method. As a result, the ASA and all other relevant ASAs can update their expectations for method M using Bayesian update; the method will now have three possible duration outcomes $\{d_2^M, \ldots, d_4^M\}$ with probability $P(d_i^M) = 0.33\ \forall 2 \leq i \leq 4$. The ASAs can then re-analyze the problem and re-schedule methods that do not start execution prior to time $t + d_1^M$. Similar re-scheduling can be performed at time $t + d_2^M$. Finally, at time $t + d_3^M$ the ASA receives the completion message and all agents can re-schedule any methods that have not yet (i.e., prior to time $t + d_3^M$) started execution. This final rescheduling is based on the actual duration and quality outcome of method M. The quality achieved at the end of this "re-scheduling" process may be very different from the expected quality of the schedule the ASAs had at the beginning of the process.

Thus, to calculate the expected quality, it is necessary to capture the effect of the re-scheduling of events that occurs whenever a method's execution time exceeds one of the possible durations defined for the method and no EC message has been sent, or when the method completes execution. Generally, if a method is still executing at time $t + d_j^M$, then its outcome distribution can be updated to $\{d_{j+1}^M, \ldots, d_{|D^M|}^M\}$ where $P'(d_i^M) = P(d_i^M)/(1 - \sum_{k=1}^{j} P(d_k^M))\ \forall j < |D^M|, j < i \leq |D^M|$.

Thus, the schedule that will be used at future time t' $(t' > t)$ should be constructed incrementally, based on the last re-scheduling event that would have occurred. For exposition purposes, we assume that method M is supposed to start execution at time t. Hence,

$$S_{t'}(T, (d_i^M, q_j^M)) = S_{t'}(T_c(S_{t+d_{i'}^M}), \neg d_{i'}^M) \quad , t' < t + d_i^M \quad (2)$$

where T_c is the constrained problem generated by the re-scheduling process that occurs at the time $t + d_{i'}^M$ $(i' = argmax_i\{d_i^M < t'\})$ when the ASA learns that duration $d_{i'}^M$ is no longer a valid outcome.

Finally, the expected value of the information the owner may have can be calculated as:

$$V = \sum_{i=1}^{k}\sum_{j=1}^{l} \big(S_t(T, (d_i^M, q_j^M)).quality - S_{t+d_i^M}(T, (d_i^M, q_j^M)).quality\big) P(d_i^M, q_j^M) \quad (3)$$

[3]This has many equivalents in other domains, e.g., a UPS scheduling assistant can be updated based on events recorded in the standard tracking system.

Thus, the estimation of the improvement in quality that results from learning each possible outcome requires two types of queries of the scheduler, (1) a non-constrained query using the new information, and (2) a constrained query that incorporates the results of re-scheduling processes that occur until an EC message is received.

3.2 The Querying Algorithm

Algorithm 1, given in pseudo-code below, embodies the principles just described to estimate the value of getting the actual outcome of a method from the owner (before the method completes execution). The core functionality it supports is: (a) the calculation of the differences in the expected quality with and without the owner's information for the outcomes associated with the first (shortest) duration; and, (b) if execution exceeds the first possible duration outcome, recursive execution of the algorithm on the new schedule to obtain the revised quality for the reduced problem.

Algorithm 1 $GetValue(T, M)$ - Evaluating the benefit in interacting with the owner

Input: T - a cTAEMS problem; M - The method we are working on; t - the time when method M starts executing;
Output: V - the expected additional utility if receiving the actual outcome from the owner.

1: **if** $D^M.length = 0$ **then**
2: **return** 0
3: **end if**
4: Set $S = GetSch(T)$; $Value = 0$;
5: Remove schedule S from cTAEMS T
6: **for** $j = 1$ to $Q^M.length$ **do**
7: Set $T_c = T.clone()$; $T_{nc} = T.clone()$; $M' = M.clone()$
8: Set $D^{M'} = (d_1^M)$; $Q^{M'} = (q_j^M)$; $P(d_1^M) = P(q_j^M) = 1.0$
9: Replace method M in T_c and T_{nc} with method M'
10: **for** $i = 1$ to $S.length$ **do**
11: **if** $S[i].start\_time < t + d_1^M$ **then**
12: Add $S[i]$ to T_c
13: **end if**
14: **end for**
15: Set $Value = Value + P(d_1^M)P(q_j^M)(GetSch(T_{nc}).quality - GetSch(T_c).quality)$
16: **end for**
17: **for** $j = 2$ to $D^M.length$ **do**
18: Set $P(d_j^M) = P(d_j^M)/(1 - P(d_1^M))$
19: **end for**
20: Set $P_{remain} = 1 - P(d_1^M)$
21: Remove d_1^M from D^M
22: Replace method M' from T_c with method M
23: **return** $Value + P_{remain} * GetValue(T_c, M)$

The input to the algorithm is a cTAEMS problem, T, which includes a partial schedule and a method M (with an updated outcome distribution). M's actual outcome is what the CA is considering asking the user about. The recursive implementation (step 23) allows the algorithm to emulate the re-scheduling process used by the ASA, as described above. The algorithm first queries the scheduler for the best schedule that can be produced given the initial problem settings (represented by S (in step 4)). The variable $value$ accumulates value over the recursion. The main loop (steps 6-16) weighs the differences in expected quality of the schedules for the first possible duration outcome, of the method M, by creating two new cTAEMS problems (step 7) in which method M has that duration outcome and one of the quality possible outcomes with probability 1.0 (steps 8-9). In the first problem, T_{nc}, no scheduling constraints are imposed, whereas in the second problem, T_c all the ASAs' methods that were scheduled to start executing before the ASA learns the actual outcome of method M are included in the problem settings as an initial schedule that cannot be changed (steps 10-14). By removing any re-scheduling flexibility up to time $t + d_1^M$, the cTAEMS problem T_c emulates the scenario in which the ASAs adjust to information about method M's outcomes only

when the actual outcome is determined by the environment (i.e., when the method has completed execution or failed). Similarly, the cTAEMS problem T_{nc} emulates the scenario in which the information is received a priori and the ASAs are free to re-plan at this earlier time using this information.

Step 15 uses the function $GetSch(T)$ to query the scheduler so that it returns the optimal schedule that can be generated for the problem T (taking the schedule elements it contains as a constraint for scheduling). A comparison of the quality of the schedule received for the non-constrained problem T_{nc} and that for the constrained problem T_c yields the incremental quality improvement that can be obtained by knowing the outcome (d_i^M, q_j^M) a priori. The result is multiplied by the probability of getting this outcome from the owner (step 15).

Therefore, upon the completion of steps 6-16 the algorithm has the weighted expected utility of outcomes associated with duration d_1^M. To complete the process (i.e., for adding the weighted utility for methods with $d_i^M > d_1^M$), the algorithm creates a new problem in which the outcome d_1^M is removed from M, and it updates the probabilities of the remaining outcomes (steps 17-22). The new problem (T_c) contains the scheduling constraints imposed by the inability to change any of the methods that are scheduled to start their execution prior to time $t + d_1^M$. This problem is the input for the recursive execution of the algorithm, which stops when there are no more duration outcomes for M.

Once the CA module calculates this gain, it can multiply it by the probability representing its estimation that the owner knows the actual outcome (as detailed by Sarne and Grosz [17]) and if the result is greater than its estimated cost associated with the interruption then the interaction is being initiated.

The total number of queries sent to the scheduler throughout the recursive executions of the algorithm is $(2 * |Q^M| + 1) * |D^M|$, which is derived as follows: (a) two queries (constrained and non-constrained) for each outcome (making a total of $2*|Q^M|*|D^M|$), used for calculating the outcome's marginal utility (see step 15); and (b) one query for generating the schedule evolvement due to realizing the elimination of each duration outcome, as time goes by (step 4) (i.e., a total of $|D^M|$).

4. IMPROVING EFFICIENCY

The $(2 * |Q^M| + 1) * |D^M|$ scheduler queries required by Algorithm 1 is a significant overhead, because querying the scheduler is a resource-intensive task. As a result, techniques that enable minimizing the number of queries the CA uses for its benefit estimation process can significantly improve ASA performance. In this section, we show how to reduce significantly the number of queries used for estimating the information value without compromising the accuracy of the obtained result. This decrease is achieved by an intelligent selection of the sequence of outcomes for which queries are generated.

As shown in Equation 3, the expected value of information can be represented as a weighted sum of the differences between the quality associated with the non-constrained problem, T_{nc}, and the constrained one, T_c. By identifying outcomes for which the difference is zero, it is possible to eliminate the need for generating the two queries in such cases. Information has value at a specific time only if its receipt at that time leads an agent to change its owner's schedule in a way that it could not have if it only received this information in an EC message or induced it from not receiving an EC message at the end of a possible method duration. Thus a zero difference occurs when no agent in the environment changes its owner's schedule in comparison to EC-messages based schedule. We use the term "relevant change" to refer to a scheduling change the existence of which indicates that the ASA's learning about a method's outcome before that method finishes executing will yield positive value for the ASA. Formally, the definition of a relevant change is as follows:

DEFINITION 1. *Consider an initial problem T and a specific outcome (d_i^M, q_j^M) of method M. Any change between the schedules $S_t(T, (d_i^M, q_j^M))$ and $S_{t+d_i^M}(T)$ over the interval $(t, t+d_i^M)$ is a "relevant" change associated with receiving the true outcome (d_i^M, q_j^M) at time t.*

The basic idea of the first query-reduction method is that the existence of relevant changes is the only thing that matters for identifying zero value differences, because these changes determine whether knowing the outcome (d_i^M, q_j^M) before the completion message occurs contributes value or not. The difference in quality due knowing this information prior to time $t + d_i^M$ is necessarily positive if relevant changes between the non-constrained schedule and the EC-based schedule are found at time prior to $t + d_i^M$. Otherwise, the difference is necessarily zero.

Thus, the CA should first send the scheduler all its non-constrained queries ((T_{nc})-based) and should initiate a constrained query ((T_c)-based) only for those outcomes (q_i^M, d_j^M), $q_i^M \in Q^M, d_j^M \in D^M$, for which a relevant change was identified by inspecting the schedule sent back for the constrained query. If the number of outcomes that result with relevant changes is N ($N < |Q^M| * |D^M|$), then the number of queries required using this technique is $(|Q^M|+1) * |D^M| + N$, derived as follows: (a) $|D^M|$ queries for generating the schedule changes resulting from the elimination of duration outcomes as in the original Algorithm 1; (b) $|Q^M|*|D^M|$ for obtaining the non-constrained schedules; and (c) N constrained queries for calculating the quality differences for those scenarios which yield relevant changes. Although this approach significantly lessens the number of queries, it is still far from the theoretical minimum number of queries that need to be used, which is $2 * N + |D^M|$, where the $|D^M|$ additional constrained queries are required for following the re-planning procedure and the $2*N$ queries (N constrained and N non-constrained) are required for calculating the quality differences for those scenarios which yield relevant changes.

A further significant reduction in the number of queries sent to the scheduler may be obtained by analyzing the possible-outcomes space. Figure 2 illustrates this space of outcomes (and is an analogue of the table given in Figure 1) for a method M with 5 possible duration outcomes and 4 quality outcomes. We begin our analysis with an arbitrary outcome $o=(d_i, q_j)$ that is not associated with relevant changes. Though this outcome is not associated with relevant changes, its neighboring outcomes in the outcomes space (i.e., (d_i, q_{j+k}) or (d_{i+k}, q_j), $k \in \{+1, -1\}$) may be associated with relevant schedule changes. Table 1 gives examples of ways an increase or a decrease in the duration or quality results in relevant schedule changes (both in a particular ASA's and other ASAs' schedules).

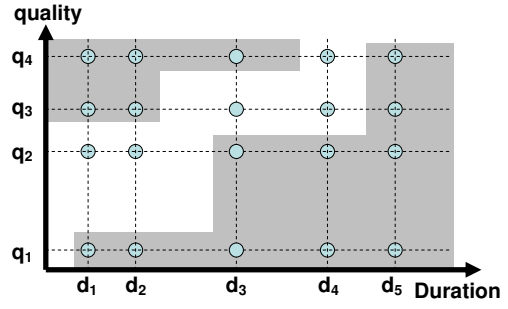

Figure 2: A method's outcome space

If a method o' which is a neighbor of method o and has relevant changes can be found, then relevant changes appear in any other outcome in that direction of movement in the outcomes space. For example, if no relevant changes are found for outcome $o=(d_i, q_j)$,

	Self changes	Changes in other ASAs
$D \uparrow$	Method M exceeds deadline thus replaced by method $\overline{M'}$	Method M_1 (enabled by M) cannot start on time, and thus is replaced with $\overline{M_2}$
$D \downarrow$	The execution of method M can now be delayed and a new method $\overline{M'}$ can be scheduled before it	Method $\overline{M_1}$ enabled by M is added to the schedule, and consequently method $\overline{M_2}$ that facilitates $\overline{M_1}$ is added at time $t' < d^M + t$
$Q \uparrow$	Not applicable	Method $\overline{M_1}$ facilitated by M is added to the schedule, and consequently method $\overline{M_2}$ that enables $\overline{M_1}$ is added at time $t' < d^M + t$
$Q \downarrow$	Method M is replaced by method $\overline{M'}$ (which has a better quality)	A method M_1 under a q_{min} QAF along with M is replaced by method $\overline{M_2}$

Table 1: Relevant schedule changes due to a new information concerning method M's quality and duration ($\overline{M_j}$ denotes a method that is currently non-scheduled)

and a relevant change is found for outcome $o' = (d_{i-1}, q_j)$, then relevant changes exist also in any outcome $o* = (d_{i-k}, q_j), \forall 0 < k < i-1$. This property may be explained by the fact that the same arguments used for changing the schedule when moving from outcome o to a neighbor outcome o', remain valid in the transition to $o*$.[4] Therefore, typically, the area representing the outcomes for which relevant schedule changes occur partially wrap around the area representing the outcomes for which no relevant schedule changes occur. This characteristic is illustrated in Figure 2 where those outcomes in which relevant schedule changes occur are marked with a gray background. An immediate implication of this result is that for any fixed duration d_i, if there are outcomes $(d_i, q_j), 1 \leq j \leq |Q^M|$ not associated with relevant schedule changes, then these outcomes appear in a sequence along the quality axis (and similarly when setting a fixed quality).

This regularity suggests an algorithm for solving the problem as a "game" defined by the following rules: (a) given a matrix of size $|Q^M| * |D^M|$ representing the space of possible outcomes, in which each of the elements can be either associated with relevant changes or not; (b) the goal is to identify the matrix elements that are associated with relevant changes (marked in gray in Figure 2); (c) at each game stage, determine which element of the matrix to query next; (d) for each element queried, receive an indication of whether it is associated with relevant schedule changes; (e) if two matrix elements within the same row (alternatively, column) are found to be not associated with relevant changes, then all outcomes between them at that row (alternatively, column) are also not associated with schedule changes. The goal is to query all the elements associated with relevant changes while querying as few elements as possible not associated with relevant changes.

A simple implementation of an efficient algorithm for playing this "game" would start with the lowest duration and quality outcome (i.e., the lowest left corner in Figure 2) and would check the matrix elements by moving upward. Upon hitting an outcome that does not involve relevant schedule changes (or scanning all elements of a column in this direction), the process moves to the next column (next duration value), moving upward again in the same manner. This process will continue until it either has covered all duration columns or the first outcome of a scanned column reveals no relevant schedule changes. In this latter case, the process will continue in the same manner, but starting with the rightmost column and switching to lower duration columns whenever it hits an outcome that does not involve relevant schedule changes (or it scans all columns in this direction). When the process reaches a column that has already been scanned, or if the first method in the scanned column reveals no relevant schedule changes, then the process is repeated from the top of the columns down the quality scale. (Depending on the dimensions of the problem, it may be better to scan rows instead of columns, but that process is analogous.) This method guarantees a maximum overhead of $2 * min(|D|, |Q|)$ unnecessary queries to the scheduler, since at most it will go over all columns and will stop once while moving upward and once while moving downward in each column. Therefore, the number of queries generated using this method is bounded by $2 * N + 2 * min(|D^M|, |Q^M|) + |D^M|$.

5. EXPERIMENTS AND RESULTS

Our CA module implementation was successfully integrated in a Coordinators ASA and evaluated in a real-time simulation environment created for the *Coordinators* project. In a series of tests comparing performance of a Coordinators ASA, with and without the CA module described in this paper, for cTAEMS problems that were part of the Coordinators test suite, the CA module was shown to lead to significant performance improvements.[5] For those scenarios in which the information supplied by the (simulated) owner benefitted the ASAs, the quality improved tremendously. For those scenarios for which the CA initiated an interaction with the (simulated) owner, but the information provided by the owner left the plan unchanged, there was a decrease in quality as a result of the interaction cost. This result was common, because the decision whether to interrupt the user or not is based on the expected benefit which may differ from the actual benefit. However, despite the fact that such scenarios resulted in a quality decrease, the overall improvement was positive, because of the far larger benefit obtained for those cases in which the owner's information led to schedule changes.

We also tested the CA module with a constraint-based scheduler [6] which allowed us to execute extensive tests without the overhead of the full *Coordinators* simulation system. It thus enabled us to test query-reduction performance (as described in Section 4) more thoroughly. These experiments used the test suite that the *Coordinators* project used to evaluate different ASA architectures.[6]

For each test case, the CA picked a random snapshot of the schedule being executed and activated its utility estimation mechanism for the specific method being executed at that time, with the following different techniques (from Section 4): (a) evaluating all outcomes (denoted "Naive"); (b) sending a non-constrained query for each outcome and then sending constrained queries as necessary (denoted "Improved"); and, (c) using the game-based algorithm described in Section 4 (denoted "Improved+"). In addition, we calculated the theoretical minimum number of queries needed $(2 * N + |D^M|)$ for each problem. The graphs below depict the average number of queries used by the CA for estimating the value of information using each of these methods, and the average theoretical minimum (denoted "Theoretical"). The results are based on 2093 problems from the test suite. These problems were divided into types following a classification based on parameters such as the scale of the required scheduling task, the number of agents, and the interdependencies between tasks. Each type represents different problem characteristics and complexity.

The left graph in Figure 3 depicts performance by problem type in terms of number of queries (y-axis). For this set of experiments, there were 5 quality outcomes and 3 duration outcomes for each

[4]This is based on the existence of *enables* and *facilitates* NLEs in the current version of TAEMS in Coordinators (both influence the same way). If future implementations add conflicting NLEs (such as *disables* and *hinders*), then theoretically there could exist extremely rare scenarios where a change in a method's outcome will create several conflicting effects. However, it is very difficult to find an example of such a scenario.

[5]We do not give quantitative results, because space limitations preclude providing the test environment parameters that would make those meaningful.

[6]The only change we made was to augment the outcome distribution in order to increase the number of outcome values for experimental purposes (but without changing the original mean and edges values).

evaluated method. As shown in the graph, the Improved+ approach provides the greatest improvement and is very close to the theoretical minimum. For problem types 5 and 6, both the theoretical minimum and the Improved+ approach produce very few queries. This result is correlated with the relatively low level of dependencies between methods in these problem types, leading to a relatively small effect of external information on the optimal schedule.

The right graph in Figure 3 depicts the additional overhead (in percentages) generated by each of the different methods in comparison to the theoretical minimum number of queries. In this graph, each group (represented on the horizontal axis) is distinguished by the number of quality outcomes used for each method (the number of duration outcomes was set to be 3) and includes all the problems, regardless of their type (i.e., each group includes all the problems used in the left graph). The theoretical minimum number of queries is the baseline (100%) against which the overhead of all other methods is calculated. This graph shows that while the overhead of the "Naive" and "Improved" methods modestly increases as the number of possible outcomes increases, the Improved+ algorithm exhibits a constant improvement as a function of the number of outcomes; it has only a 23% average overhead in comparison to the theoretical minimum number of queries for 9 quality outcomes.

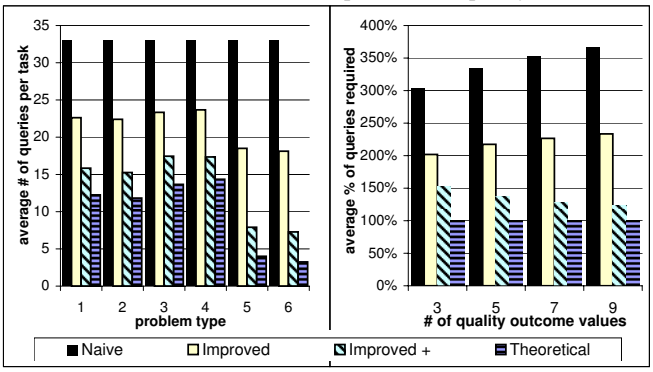

Figure 3: # of interactions initiated

6. RELATED WORK

The research reported in this paper is within the broad category of AI research that uses decision-theoretic techniques to address problems of agent decision making in dynamic environments with incomplete information [9, 5]. It relates most directly to work on interruption management, autonomous agents to support planning, and information gathering.

Most work on interruption management is concerned with the question of determining when to interrupt users to provide help or information relevant to their work [8, 10]. For instance, the Notification Platform system [8] modulates the flow of messages from multiple sources to devices by performing ongoing decision analysis. As in the CA problem, the agent is uncertain about a user's workload and focus of attention as well as about the value of the alternative actions a user will undertake as a result of receiving new information. The primary goal of such work is to balance the expected value of information with the attention-related costs of disruption. It typically uses probabilistic models (often Bayesian network models) of a user's future actions to infer the probability of alternative feasible actions as a function of actions the agent has observed, using that context to determine the cost of interruption. Although this problem domain has much in common with the ASA domain, work in this area differs from our work in three ways. First, it focuses almost entirely on modeling the user's attentional state and avoiding unnecessary or unhelpful interruptions and the ensuing frustration they cause [10, inter alia]. Second, it commonly makes two assumptions that do not hold in the ASA domain: (1) that users will change their actions in a pre-determined manner when they receive the new information, and (2) that it is straightforward to calculate the benefit from performing the alternative action. Third, in prior work on interruption management, it is the agent that has information needed by the person rather than the reverse. As a result, the question is determining the effect of having this information on the user's performance, and there is no need to deal with uncertainty about the information (i.e., to consider different types of information the user has). One exception is the work of Fleming and Cohen [4, 5] who use a utility-based quantitative approach to designing systems that are capable of making decisions about when to ask a user to provide additional information to improve problem solving. However, their work emphasizes modeling the cost of "bothering" the user repeatedly. A related difference between our work and prior work is that in prior work, the information is not principally probabilistic (as in the CA's case), so systems do not need to reason about how users will change their actions in situations analogous to not receiving an EC message.

Research on autonomous agents providing assistance to human planners [15; 22, interalia] by providing information alerts at appropriate times also focuses mainly on the person side of the human-computer interaction, considering such human aspects as the cognitive load created on the person providing information or alerting the person to important events. In this domain, the value of alerting users with new information derives from its usefulness for their decision making.

Research on value-driven information gathering systems [7] focuses on the development of autonomous agents operating in an information rich domain under time and monetary resource restrictions. It considers decisions about the information to collect based on an explicit representation of the user's decision model and a database of information sources. Queries are activated based on the marginal query value, i.e., by considering the current information known by the system and the set of queries that have been sent but have yielded answers. Query-values are derived from a utility function based on the user's preferences in combination with information provided by the expert who constructed the decision model.

Our work differs from all this prior work in two ways. First, it considers the problem of efficiently obtaining the value of information, which prior efforts have not needed to do, because the value of information either was assumed known or could be calculated simply. Furthermore, even in cases where the information value-estimation process required more complex calculation, the system's goal is assisting the user so the systems-resource problems that we faced do not arise. Second, because schedule-expertise is external to CA, it needs to interact effectively with another ASA module to calculate information value whereas prior work has presumed all systems-available information related to the information-value problem is internal to the component making this calculation.

7. DISCUSSION AND CONCLUSIONS

This paper addresses the problem of computing the value of information for multi-agent coordinated scheduling in situations in which the (human) users of an autonomous agent system (ASA) have access to information that is not directly available to the ASA. It considers this problem in the context of developing a coordination autonomy module (CA) within a coordination-manager system. Unlike other problem domains in which value-of-information must be determined, this setting presents the challenge of computing the value of information when the expertise needed for reasoning about alternative schedules is not contained within the CA itself but rather in a separate scheduler module. Furthermore, the fact that scheduling problems require the value of information be computed in an environment in which the schedule is dynamically (and often,

continuously) being revised, poses a great challenge with respect to the computation of the effect of not getting this information a priori. As earlier sections of this paper elaborate, even when the ASAs do not receive method-related information from the user at the current time, they will sometime later be able to improve their schedules based on the elimination of different duration outcomes of the executed method (as time goes by) or the receipt, eventually, of the execution completion message of this method. The CA thus must emulate those dynamics in order to accurately infer the system performance for the case in which it does not receive this information from the user. These demands combined with strong constraints on ASA resources raise significant efficiency issues, issues which are normally not addressed in applications for estimating the value of information in other domains. The need for systems to have adequate planning and scheduling capabilities permeates autonomous agents and multi-agent systems applications, making mechanisms such as those described in this paper important for a wide-range of multi-agent systems operating in uncertain dynamic environments.

The paper presents an algorithm that determines the value of information that is directly available only to the person being supported by the ASA, but is potentially relevant to scheduling revisions. The empirical results described in Section 5 provide strong support for the usefulness of the algorithm. In addition, the paper presents two methods for reducing queries to the scheduler to address the strong resource constraints on the ASA. These methods were extensively tested, and the results of these experiments establish that a significant decrease can be achieved in terms of the number of queries the CA module sends by adopting these methods. The implementation of the algorithm and query-reduction methods within an ASA developed in the context of the *Coordinators* provided an environment for validating assumptions in system design and enabled testing of the mechanisms we developed based on an externally developed test suite of experiments. Furthermore, these mechanisms are scheduler-independent and thus can be used for any ASA implementation.

Although the algorithm and query reduction methods developed in this paper were motivated in part by the separation of CA from the scheduler, these mechanisms are also applicable in systems in which scheduling and value-of-information expertise are contained within a single module. Although such a unified module has the potential for using heuristics to filter queries based on knowledge of scheduling constraints, the use of our approach guarantees obtaining the actual value of information, while such heuristic approaches can provide only estimations.

Possible extensions of this work include addressing the problem of determining the methods to focus on in a given schedule, assuming the CA is constrained with respect to the total number of scheduler queries it can initiate.

8. ACKNOWLEDGEMENT

The research reported in this paper was supported in part by contract number 55-000720, a subcontract to SRI International's DARPA Contract No. FA8750-05-C-0033. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or the U.S. Government. We are grateful to Monira Sarne for developing the experimental infrastructure, Willem-Jan van Hoeve for providing the constraint-based scheduler and support in its use, and Ben Lubin for insightful discussions and assistance in building an initial CA.

9. REFERENCES

[1] A. Ang and W. Tang, editors. *Probability Concepts in Engineering Planning and Design, Volume II - Decision, Risk, and Reliability*. John Wiley & Sons, New York.

[2] M. Boddy, B. Horling, J. Phelps, R. Goldman, and R. Vincent. ctaems language specification. Unpublished; available from this paper's authors, 2005.

[3] R. Clemen, editor. *Making Hard Decisions: An Introduction to Decision Analysis*. Duxbury Press, Belmon, CA.

[4] M. Fleming and R. Cohen. A user modeling approach to determining system initiative in mixed-initiative AI systems. *Lecture Notes in Computer Science*, 2109:54–63, 2001.

[5] M. Fleming and R. Cohen. A decision procedure for autonomous agents to reason about interaction with humans. In *AAAI Spring Symp. on Interaction between Humans and Autonomous Systems over Extended Operation*, 2004.

[6] C. Gomes, W. van Hoeve, and B. Selman. Constraint programming for distributed planning and scheduling. In *AAAI Spring Symp. on Distributed Plan and Schedule Management*, 2006.

[7] J. Grass and S. Zilberstein. A value-driven system for autonomous information gathering. *J. Intell. Inf. Syst.*, 14(1):5–27, 2000.

[8] E. Horvitz, C. Kadie, T. Paek, and D. Hovel. Models of attention in computing and communication: from principles to applications. *Commun. 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] B. Hui and C. Boutilier. Who's asking for help?: a bayesian approach to intelligent assistance. In *IUI '06*, 2006.

[11] W. McClure. Technology and command: Implications for military operations in the twenty-first century. Maxwell Air Force Base, Center for Strategy and Technology, 2000.

[12] D. Musliner, E. Durfee, J. Wu, D. Dolgov, R. Goldman, and M. Boddy. Coordinated plan management using multiagent mdps. In *AAAI Spring Symp. on Distributed Plan and Schedule Management*, 2006.

[13] K. Myers, P. Jarvis, and T. Lee. Active coordination of distributed human planners. In *AIPS'02*, 2002.

[14] J. Phelps and J. Rye. Gpgp a domain-independent implementation. In *AAAI Spring Symp. on Distributed Plan and Schedule Management*, 2006.

[15] M. Pollack. Intelligent technology for an aging population: The use of ai to assist elders with cognitive impairment. *AI Magazine*, 26(2):9–24, 2005.

[16] H. Raiffa and R. Schlaifer, editors. *Applied Statistical Decision Theory*. Harvard University Press, Cambridge, MA.

[17] D. Sarne and B. J. Grosz. Sharing experiences to learn user characteristics in dynamic environments with sparse data. In *AAMAS'07*, (to appear), 2007.

[18] D. Sarne and B. J. Grosz. Timing interruptions for better human-computer coordinated planning. In *AAAI Spring Symp. on Distributed Plan and Schedule Management*, 2006.

[19] P. Scerri, D. Pynadath, W. Johnson, P. Rosenbloom, M. Si, N. Schurr, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *AAMAS*, pages 433–440, 2003.

[20] S. Smith, A. Gallagher, T. Zimmerman, L. Barbulescu1, and Z. Rubinstein. Multi-agent management of joint schedules. In *AAAI Spring Symp. on Distributed Plan and Schedule Management*, 2006.

[21] T. Wagner, J. Phelps, V. Guralnik, and R. VanRiper. An application view of coordinators: Coordination managers for first responders. In *AAAI*, pages 908–915, 2004.

[22] D. Wilkins, S. Smith, L. Kramer, T. Lee, and T. Rauenbusch. Execution monitoring and replanning with incremental and collaborative scheduling. In *Workshop on Multiagent Planning and Scheduling, ICAPS'05*, June 2005.