Decentralized Decision Making with Anytime Algorithms

Alan Carlin and Shlomo Zilberstein
University of Massachusetts
Department of Computer Science
Amherst, MA 01003, USA
acarlin@cs.umass.edu, shlomo@cs.umass.edu

June 2010

Abstract

Anytime algorithms allow a system to trade solution quality for computation time. In previous work, monitoring techniques have been developed to allow agents to stop the computation at the "right" time so as to optimize a given time-dependent utility function. However, these results apply only to the single-agent case. In this paper we analyze the problems that arise when several agents solve components of a larger problem, each using an anytime algorithm. Monitoring is more challenging in this case because each agent is uncertain about the progress made so far by the others. We develop a formal framework for decentralized monitoring, establish the complexity of several interesting variants of the problem, and propose solution techniques for each variant. Finally, we show that the framework can be applied to decentralized flow and planning problems.

1 Introduction and Related Work

Algorithms take time to produce optimal solutions. Users, by contrast, may not have time to wait for completion. It is unfortunate when the user is forced into a binary decision, wait for algorithm completion, or do not run the algorithm at all. Fortunately there has been a considerable amount of work on a compromise, the "anytime algorithm". Anytime algorithms can produce partial or non-optimal solutions as well as optimal ones. If interrupted before completion, they will produce the non-optimal result. In this way, the user can decide on the best way to trade off compute time for solution quality.

In artificial intelligence, the trade-off is often quantified in a utility function $U(q, t)$, where t represents the time, q represents the quality, and $U(q, t)$ is thus the utility of achieving solution quality q at time t. If the algorithm senses that further deliberation will result in a favorable trade-off of time for solution quality, the algorithm continues. If not, the algorithm stops and produces a solution. One other area of study has been the ramifications of monitoring the algorithm's progress. Monitoring can often result in a "context switch", using up resources, that briefly hinders the efficiency of the algorithm being studies. Thus, the area of anytime algorithms and monitoring lends itself to a formal analysis.

The literature on anytime algorithms is rich in centralized settings. We refer to (Anderson, 1993; Cox and Raja, In Press) for good, recent overviews, and give a brief summary here. Dean and Boddy used the term “anytime algorithm” in the 1980’s to describe a class of algorithms that ”can be interrupted at any point during computation to return a result whose utility is a function of computation time” (Dean and Boddy, 1988). They used these algorithms in their work on time dependent planning, regarding how to plan when the time available may vary. Horvitz, during the 1980’s as well, used decision theory to analyze ”costs and benefits of applying alternative approximation procedures” to cases ”where it is clear that there are insufficient computational resources to perform an analysis deemed as complete” (Horvitz, 1987). (Zilberstein and Russell, 1996) used performance profiles of algorithms in order to inform future anytime decisions.

More recent work on decision theoretic frameworks include (Hansen and Zilberstein, 2001; Sandholm, 2003). In (Hansen and Zilberstein, 2001), a performance profile of the agent is formed offline. Based on this profile, a dynamic programming approach is used to make stopping decisions. The decisions use Bayesian reasoning based on the profiles in order to ascertain probability of future quality, as well as monitoring decisions, as to whether to pause and ascertain quality, or merely to continue. The calculations also include reasoning that an agent may choose to continue in the present step, but stop in the future step. The work in (Sandholm, 2003) is aimed at the decision as to when to optimally terminate an algorithm. Termination decisions are based on the prior probability of an answer, on a utility model based on the utility of quality and time, and on performance profiles. The concept is demonstrated on a 3-SAT model.

Further complexities arise when the system is decentralized. For instance, consider a decentralized setting, where multiple agents are each solving components of a larger problem by running multiple anytime algorithms concurrently. How does each individual agent know when is the best time to stop deliberating and start executing, when it does not necessarily know the status of computation of the other agents? Perhaps the solution to this problem is to communicate status among the agents, so that they can make a better joint decision. But such communication may not be free. How to weigh the value of this communication?

In the multi-agent realm, Raja and Lesser explore a framework for coordinating agents Meta-Level control (Raja and Lesser, 2004, 2007). In these works, a single agent or multiple agents schedule a series of tasks. At various points in time, new tasks arrive, and each agent must decide whether to deliberate on the new information and whether to negotiate with other agents about the new schedule. The authors show that an MDP framework can be made to reason about such problems. These works explore the interaction of various tasks, with various deadlines and utility. We view our work as complimentary, as our work reasons about a finer granularity; we reason about progress within the individual tasks, when to terminate and communicate from tasks that have been partially completed.

In this paper, we formally analyze questions of decentralized anytime computation and monitoring. The paper concerns itself with the deliberation phase, with how a multi-agent system should monitor its progress in finding a solution, and when a multi-agent system should cease deliberation and begin execution. We develop a theoretical framework for this problem, prove complexity in a number of cases, and propose solution methods. We demonstrate the value of multi-agent techniques, as opposed to simple single-agent techniques, on small example problems.

2 Model

We formalize a decentralized monitoring problem (DMP) by defining it as a tuple $< Ag, Q, A, P, U, C_L, C_G, T >$.

- Ag is a set of agents.
- $Q =< Q_1, Q_2...Q_n >$ is a set of possible quality levels for agents $1..n$. At each step t, we denote the vector of agent qualities by $\vec{q}^{\,t}$, or usually more simply by $\vec{q}$. Components of $\vec{q}^{\,t}$ are qualities for individual agents. We denote the quality for agent i at time t by q_i^t (occasionally r_i^t and s_i^t is similarly used).
- $\vec{q}^{\,0}$, a joint quality at the initial step, known to all agents.
- A, A set of options, "continue", "stop", "monitorL", "monitorG" available to each agent. monitorL and monitorG represent the notions "monitor locally" and "monitor globally", respectively.
- P is the transition model for the "continue" action. For all $i, t \in \{1..T-1\}$, $q_i^t \in Q_i$, and $q_i^{t+1} \in Q_i$,
$$P(q_i^{t+1}|q_i^t) \in [0, 1]$$
Furthermore, $\Sigma_{q_i^{t+1} \in Q_i} P(q_i^{t+1}|q_i^t) = 1$. We also assume that the transition model of continuation for each agent is independent of the transition model of continuation for the other agents, that is, $P(q_i^{t+1}|q_i^t, q_j^t) = P(q_i^{t+1}|q_i^t)$
- A utility function , $U(\vec{q}, t)$, that assigns a utility to quality vector $\vec{q}$ at time t.
- C_L,C_G, is a cost assigned to local monitoring and global monitoring actions respectively.
- T, the horizon, is the number of time steps in the problem.

The agents in our model represent various agents, each running an anytime algorithm. Unless a "stop" action is taken by one of the agents, all agents will continue to deliberate for T time steps.

2.1 Quality

Quality represents the performance of each agent, as it deliberates. For instance, if each agent is running an online algorithm to compute the solution to a max flow problem, an example of quality is the flow of the current (perhaps non-optimal) solutions. The utility captures the fact that an agent considers utility as a function of quality of the individual agents. The time, t, is also considered in the utility function. In this paper we consider a class of utility functions.

Definition 1. *A time-dependent local utility function is called time separable if the utility of a solution of quality q at time t can be expressed as the difference between two positive functions,*
$$U(\vec{q}, t) = U_I(\vec{q}) - U_C(t)$$
where $U_I(\vec{q})$ is called the intrinsic value function and $U_C(t)$ is called the cost of time.

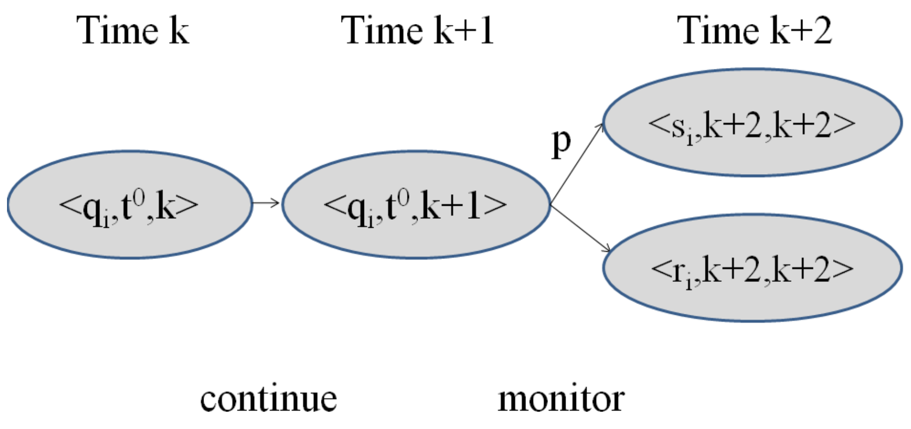

Figure 1: An example of the state space for one of the agents, while running a model $M1$ for 3 steps.

The problems of interest are problems where both $|U_I|$ and $|U_C|$ monotonically increase with time. Thus, our agents must decide whether to accept the current solution quality, or whether to continue deliberation, which will result in a higher solution quality but also a higher time cost.

2.2 Actions

At each time step, agents decide which option to take, to "continue", "stop", or "monitor" globally or locally. If all agents choose to "continue", then the time step is incremented and solution quality transitions according to P. If any agent chooses "stop", then computation ceases for all agents and the utility $U(\vec{q}, t)$ of the current solution is taken as the final utility. If any agent chooses "monitorL", then for each agent that chooses monitorL, a cost of C_L is subtracted from the utility. If any agent chooses "monitorG", a single cost of C_G is subtracted from the utility. After an agent chooses to monitor, it must then choose whether to continue or stop, at the same time step. The next time step occurs only after all agents choose to "continue".

Agents are assumed to know the initial quality vector $\vec{q}^{\,0}$ only. An agent has no knowledge about quality in later time steps, unless a monitor action is taken. A "monitorL" action monitors the local quality; when agent i takes the "monitorL" action at time t it learns q_i^t. However, it still does not know any component of $\vec{q}^{\,t}_{-i}$. By contrast, a "monitorG" action forces **all** agents to communicate with each other, and all agents learn the global quality $\vec{q}^{\,t}$.

3 Local Monitoring

In this section, we examine the concept of local monitoring. That is, each agent must decide whether to continue its anytime computation, stop immediately, or to monitor its progress at a cost C_L, and then decide.

	q_1^t	r_1^t	s_1^t
q_2^t	$EU(q_1^{t+1}, q_2^{t+1} \| q_1^t, q_2^t)$	$EU(q_1^{t+1}, q_2^{t+1} \| r_1^t, q_2^t)$	$EU(q_1^{t+1}, q_2^{t+1} \| s_1^t, q_2^t)$
r_2^t	$EU(q_1^{t+1}, q_2^{t+1} \| q_1^t, r_2^t)$	$EU(q_1^{t+1}, q_2^{t+1} \| r_1^t, r_2^t)$	$EU(q_1^{t+1}, q_2^{t+1} \| s_1^t, r_2^t)$
s_2^t	$EU(q_1^{t+1}, q_2^{t+1} \| q_1^t, s_2^t)$	$EU(q_1^{t+1}, q_2^{t+1} \| r_1^t, s_2^t)$	$EU(q_1^{t+1}, q_2^{t+1} \| s_1^t, s_2^t)$

Table 1: Expected utility of continuing for two agents, when each agent has achieved quality q, r, or s at the current time step. The expected utility must be computed for the next time step. However, each agent does not know the quality of the other agent. Thus, agent 1 must decide on which columns it continues, agent 2 must decide on which row. The agents only jointly continue if both a column and row is selected for an entry.

3.1 Complexity of Local Monitoring

When $C_L = 0$, each agent should choose to monitor locally on every step, since doing so is free. When $C_G = \infty$, agents should never choose to monitor globally. The following theorem shows that even for the simple case where $C_L = 0$, $C_G = \infty$, and number of agents is fixed, the problem of finding a joint optimal policy is NP-complete.

Theorem 1. *The problem of finding an optimal solution for a DMP with $|Ag| = k$, $C_L = 0$ and $C_G = \infty$ is NP-hard.*

Proof. First we show that the problem is NP-hard. Table 1 illustrates a simple 2-step problem for 2-agents. Each agent has achieved a local quality, which it knows. Each entry in the matrix shows the value of continuing, given that agent 1's quality is specified by its column, and agent 2's quality is specified in the row. However, each agent does not know the quality of the other agents, it can only reason about the probabilities of the other agents' qualities based on the initial quality. Thus, agent 1 can only select to continue or not continue on each column, and agent 2 can only select to continue or not continue on each row. If either agent decides to stop, utility in future steps is always zero. Furthermore, the future utilities listed in the table must be weighted by the probability of inhabiting the corresponding joint state at the current step.

Thus, the expected utility of the joint decision is the weighted sum of the entries where all agents have selected to "continue". However Tsitsiklis proved that solving this game is NP-hard through reduction from $3SAT$ (Tsitsiklis and Athans, 1985). □

To show that this DMP is in NP, we will reduce to a transition independent Decentralized MDP (Dec-MDP), a problem which was shown by Goldman and Zilberstein to be NP-complete (Goldman and Zilberstein, 2004). We will refer to this particular Dec-MDP model as **M1**.

In the Dec-MDP model, each agent has a local state space (denoted S_i) available, similar to a classic MDP. The vector of states, one for each agent, is referred to as the joint state. Each agent takes one action from a set of actions denoted A_i, and actions have stochastic effects which change the local state. The vector of actions, one per agent, is referred to as the joint action. Finally, agents receive a joint reward (denoted $R(\vec{s}, \vec{a})$) for taking a joint action from a joint state. Execution takes place sequentially, a joint action is taken from a joint state, a joint reward is received, and the process repeats. In a finite horizon problem, there are T repetitions, with T being the horizon of the problem. An agent's Dec-MDP policy is a mapping from its history of states and actions to a plan for future actions, and is denoted π_i.

Each agent is aware of its own local state and local action, but not necessarily the states and actions of the other agents at run-time. It is aware, however, of the other agent's policy which was formed at planning time. The Dec-MDP model enforces the rule that the joint state is jointly fully observable. That is, between the agents, the whole state can be observed at every step. In typical formulations of Dec-MDP, this means each agent is aware of its local state but not the state of the other agents. In a transition independent Dec-MDP, state transitions of each agent are fully independent of each other, no agent can have an effect another agent's local state. The only dependency between the agents is with respect to joint reward.

Theorem 2. *Let k be a constant. The problem of finding an optimal solution for a DMP with $C_L = k$ and $C_G = \infty$ is NP-complete.*

Proof. NP-hardness follows from above, with $k = 0$ as a special case. To show NP-completeness, we show that the problem can be reduced to a transition independent Dec-MDP. Policies and policy-values for the DMP will correspond to policies and policy-values for $M1$. The conversion to $M1$ is as follows:

The **state space** S^i for agent i is a tuple $< q_i, t_i^0, t_i >$, where q_i is a quality level (drawn from Q_i), t_i^0 is the time step at which that quality level was monitored, and t_i is the number of the current time step. We also define a terminal state for each agent.

The **action space** for all agents is {terminate, continue, monitor}.

The **transitions** consist of the following: when the action is to continue, t_i is merely incremented. When the action is to terminate, the agent transitions to the terminal state. Let $P_{MDP}(s'|s,a)$ be the transition function of the Dec-MDP $M1$. When the action is to monitor, we have:

$$\begin{aligned}
&P_{MDP}(< r_i, t_i'^0, t' > \mid < q_i, t_i^0, t >, \text{monitor}) = 0\\
&\quad if\ t' \neq t\ or\ t' \neq t'^0.\\
&P_{MDP}(< r_i, t'^0, t' > \mid < q_i, t_i^0, t >, \text{monitor}) =\\
&\quad P(r_i^{t'} | q_i^{t^0})\ if\ t' = t'^0 = t
\end{aligned}$$

The **Reward** is defined as zero if all actions choose to continue, as $-kC$ if k agents choose to monitor and none of the agents are in a terminal state, as $U(q_i, q_j ...)$ if one of the agents chooses to terminate and none of the agents are in a terminal state, and as zero if any agent is in a terminal state.

This reduction is polynomial, as the number of states in the $M1$ Dec-MDP is $|Q|T^2$ and number of actions is 3. The representation is transition-independent, as the state of each agent does not affect the state of the other agents. Note that when one agent terminates, the other agents do not enter a terminal state, such a specification would violate transition independence. Rather, this notion, that no reward is accumulated once any agent has terminated, is captured by the Reward function. No reward is received if any of the agents are in a terminal state. Since reward is only received when one of the agents enters the terminal state, reward is only received once, and the reward received by the Dec-MDP is the same as the utility received by the DMP.

An optimal policy for the Dec-MDP produces an optimal policy for the corresponding multi-agent anytime problem. Note that the uncertainty of quality present when an agent does not monitor is simulated in the MDP. Even though, in an MDP, an agent always knows its state, in this reduction the transition is not executed until the monitoring action is taken. Thus, even though an MDP has no local uncertainty, an agent does not "know" its quality until the monitor action is executed, and thus the local uncertainty of the multi-agent anytime problem is represented. □

3.2 Solution Methods with Local Monitoring

3.2.1 Greedy Solution

We first show a naive, greedy polynomial solution method to the multi-agent anytime problem, and then we show its flaws and improve on it. The solution method follows from directly adapting (Hansen and Zilberstein, 2001), and considering the other agents to be a part of the environment. The definitions in this subsection thus are adapted from the single-agent definitions of (Hansen and Zilberstein, 2001), with the addition of multi-agent vectors where appropriate. Greedy computation does not take into account the actions of the other agents, we will initiate a greedy computation by assuming that the other agents always continue, and that they will never monitor or terminate. For ease of explanation, we will describe the algorithm from a single agent's point of view. It should be assumed that each agent is executing this algorithm simultaneously.

Each agent begins by forming a performance profile for the other agents. We will use the term Pr as a probability function assuming only "continue" actions are taken, extending the transition model P over multiple steps. Furthermore we can derive performance profiles of multiple agents from the individual agents, using the independence of agent transitions. For example, in the two agent case we use $Pr(\vec{q})$ as shorthand for $Pr(q_i)Pr(q_j)$.

Definition 2. *A dynamic local performance profile of an anytime algorithm, $Pr_i(r_i|q_i, \Delta t)$, denotes the probability of agent i getting a solution of quality r by continuing the algorithm for time interval Δt when the currently available solution has quality q.*

Definition 3. *Let $t' = t + \Delta t$. A greedy estimate of expected value of computation (MEVC) for agent i at time t is:*

$$MEVC(q_i^t, t, \Delta t) = \sum_{\vec{q}^{\,t} \in Q_i} \sum_{\vec{r}^{\,t'} \in Q_i} Pr(\vec{q}^{\,t}|q_i^t, t) Pr(\vec{r}^{\,t'}|\vec{q}^{\,t}, \Delta t)(U(\vec{r}^{\,t'}, t') - U(\vec{q}^{\,t}, t))$$

The first probability is the expectation of the current global state, given the local state, and the second probability is the chance of transition. Thus, MEVC is merely the difference between the expected utility level after continuing for Δt more steps, versus the expected utility level at present. Both of these terms must be computed based on the performance profiles of the other agents, and thus the utilities are summed over all possible qualities achieved by the other agents. Cost of monitoring, C_L, is not included in the above definition. An agent making a decision must subtract this quantity outside the MEVC term.

For type-dependent utility functions, the agent faces a choice as to whether to continue and achieve higher quality in a longer time, or to halt and receive the current quality with no additional time spent. We call a policy that makes such a decision, a monitoring policy.

	q_1^t	r_1^t	s_1^t
q_2^t	**-2**	0	-1
r_2^t	**5**	**-3**	**-1**
s_2^t	**-2**	-1	1

Table 2: An example of a case where greedy termination policy produces a poor solution

Definition 4. *A monitoring policy $\Pi(q_i, t)$ for agent i is a mapping from time step t and local quality level q_i to a decision whether to continue the algorithm and act on the currently available solution.*

The above definition excludes the costs C_L and C_G, the choices are merely whether to continue or act. Thus, we must define a cost-sensitive monitoring policy, which accounts for C_L and C_G.

Definition 5. *A cost-sensitive monitoring policy, $\Pi_{i,c}(q_i, t)$, is a mapping from time step t and quality level q_i into a monitoring decision $(\Delta t, m)$ such that Δt represents the additional amount of time to allocate to the anytime algorithm, and m is a binary variable that represents whether to monitor at the end of this time allocation or to stop without monitoring.*

Thus, a cost-sensitive monitoring policy at each step chooses to either blindly continue, monitor, or terminate.

A greedy monitoring policy can then be derived by applying dynamic programming over one agent. Working backwards, such an algorithm assigns each quality level on the final step a value, based on its expected utility over possible qualities of the other agents. Then, working backwards, it finds the value of the previous step, which is the max over: (1) the current expected utility over the possible qualities of the other agents (if it chooses to stop). (2) The expected utility of continuing (if it chooses to continue). An algorithm to find a cost-sensitive monitoring policy can similarly find the expectation over each time step with and without monitoring, and compare the difference to the cost of monitoring.

3.2.2 Solution Methods: Modeling the Other Agents

The greedy solution may not be optimal when a following conditions is false: (1) The other agents will never terminate. (2) The other agents will never observe their local state.

To illustrate the first point, examine Table 2, as an instantiation of the expected utility values shown in Table 1. Assume all entries have equal probability and monitoring cost is zero, that the values shown merely represent the value of continuing. Agent 1 would greedily decide to continue if it is in state q_1^t only, as that is the only column whose summation is positive. Agent 2 would greedily continue if it has achieved quality r_2^t, as that is the only row whose summation is positive. However, this would mean that the agents continue from all joint quality levels which are bolded. The sum of these levels is negative, and the agents would do better by selecting to terminate all the time!

Rather than the greedy approach, we propose to solve the DMP with $C_G = \infty$ by leveraging the bilinear program approach of Petrik and Zilberstein to solving transition independent Dec-MDPs (Petrik and Zilberstein, 2009). We first convert the problem to the $M1$ model described above. We prune "impossible" state-actions, for example we prune states where $t_i^0 > t_i$, as an agent can not have last monitored in the future. Then we convert the resulting problem into a bilinear program. A bilinear program can be described by the following inequalities for the two-agent case (the framework is extensible beyond two agents if more agent-vectors are added).

$$\begin{aligned} maximize_{x,y}\ & r_1^T x + x^T R y + r_2 y \\ subject\ to\ & A_1 x = \alpha_1 \\ & A_2 y = \alpha_2 \end{aligned}$$

Each component of the vector x is a joint state-action pair. The vectors r_1 and r_2 are non-zero for entries corresponding to state-actions that have non-zero reward, for agents 1 and 2 respectively. α_1 and α_2 represent the initial distributions.

In the case of the $M1$ model, r_1 and r_2 are $-C_L$ in entries corresponding to state-actions where the agent monitors. The matrix R specifies joint rewards for joint actions, its entries correspond to the joint utility of the row and column state, when either agent terminates. The constraints enforce the transition probabilities. A_1 and A_2 have a row for each state, and a column for each state-action. α_1 and α_2 are 1 in the first row and -1 for the last row. Intuitively, the constraints are very similar to the classic linear program formulation of maximum flow (Cormen et al., 2009).

Bilinear programs, like their linear counterparts, can be solved through methods in the literature (Petrik and Zilberstein, 2009). These techniques are beyond the scope of this paper, one technique is to alternatingly fix x and y policies and solve for the other as a linear program. Although bilinear problems are NP-complete in general, in practice performance depends on the number of non-zero entries in R.

4 Global Monitoring

Next, we examine the case where agents can communicate with each other. We will analyze the case where $C_L = 0$ and $C_G = k$, where k is a constant. This problem is NP-complete as well. We will show this by reducing to a transition independent Dec-MDP-Comm-Sync (Becker et al., 2009). A Dec-MDP-Comm-Sync is a transition independent Dec-MDP with an additional property: After each step, agents can decide whether to communicate or not to communicate. If they do not communicate, agents continue onto the next step as with a typical transition independent Dec-MDP. If any agent selects to communicate, then all agents learn the global state. However, a joint cost of C_G is assessed for performing the communication. Agents form joint plans at each time of communication. The portion of the joint plan formed by agent i after step t is denoted π_i^t.

Theorem 3. *The DMP problem with $C_L = 0$ and C_G is a constant, is NP-complete.*

Proof. NP-hardness is the same as the theorem above.

To show that the problem is in NP, we can reduce the problem to finding the solution of a Dec-MDP-Comm-Sync (Becker et al., 2009). We create the following Dec-MDP-Comm-Sync from a DMP with $C_L = 0$.

- S^i = the possible (q_i, t) levels for agent i.

- F^i = A "terminal" state for each time step for agent i.
- **Joint States**: $\prod_i (S^i + F^i)$
- A^i = "Continue", "terminate"
- **Joint Actions**: $\prod_i A^i$
- **P**: $P(s, \text{continue}, s') = P_i(s' \in Q_i | s \in Q_i)$
- **P**: $P(s, \text{terminate}, f_i) = 1$.
- **R**: R($\vec{s}$, terminate) = 0 if any s_i in F^i.
- **R**: Otherwise, R($\vec{s}$, terminate) = U($\vec{s}$)
- **T**: The same as T from the DMP.
- **C**: C_G

The reduction is polynomial as the number of states added is equal to T, and only one action is added. □

Having represented the problem as a Dec-MDP-Comm-Sync, we use solution techniques from the literature (Becker et al., 2009). To make communication decisions, agents compute Value of Information (VoI). Agents compute

$$VoI = V^* - V - C_G$$

where V^* represents the expected utility after monitoring, V represents expected utility without monitoring, and C_G is cost of monitoring. Computation of each of these terms requires summation over the possible states of the other agent. Non-myopic policies require each agent to make a communicate/not-communicate decision at each step, resulting in the construction of a table resembling tables 1 and 2.

5 Profiles

We generated performance profiles of two different anytime algorithms. The first algorithm was the Ford Fulkerson maximum flow solution method. Our motivating scenario involved a decentralized maximum flow problem where two entities must each solve a maximum flow problem in order to supply disparate goods to the customer. To estimate the transition model P in the DMP, we profiled performance of Ford Fulkerson through Monte Carlo simulation. The flow network was constructed randomly on each trial, with each edge capacity in the network drawn from a uniform distribution. Quality levels corresponded to regions containing equal-sized ranges of the current flow. From the simulation, a 3-dimensional probability table was created, with each layer of the table corresponding to the time, each row corresponding to a quality at that time, each column representing the quality at the next time step, and the entry representing the transition probability. Utility was defined as $min(q_i, q_j) - t$ (the lesser of the flows minus the timestep, which was the time in seconds modulo 5) there were 10 quality levels and 10 time steps for each agent and $C_{L,G} = 1$. We created software to compile a Decentralized MDP from the probability matrix, as described in the previous sections, and solved the resulting problem using a bilinear program.

Error Bound	> 5	$4 - 5$	$3 - 4$	$2.5 - 3$	< 2.5
Solution Quality	0	1	2	3	4

Table 3: Assignment of quality levels for Rock Sampling.

Problem (Local/Global)	**Compile Time**	**Solve Time**
Max Flow Local	3.5	11.4
Rock Sample Local	.13	2.8
Max Flow Global	.04	370
Rock Sample Global	.01	129

Table 4: Timing results (s). Compile time represents time to compile performance profile into bilinear problem, solve time measures time taken by the bilinear solver.

The second example was the Rock Sampling domain, borrowed from the POMDP planning literature. In this planning problem, two rovers must each form a plan to sample rocks, maximizing the interesting samples. However, the location of the rocks are not known until runtime, and thus the plans can not be constructed until the rovers are deployed. We selected the HSVI algorithm for POMDPs as the planning tool (Smith and Simmons, 2002). HSVI is an anytime algorithm, the performance improves with time, its error bound is constructed and reported at runtime. Prior to runtime, the algorithm was simulated $10,000$ times on randomized Rock Sampling problems, in order to find the performance profile. Quality of performance was assigned according to Table 3, lower error bound meant higher quality. The utility function was $q_i + q_j - t$, where t was time in seconds modulo 5. Since the online HSVI algorithm took 2 seconds to construct its error bound, cost of monitoring was assigned as $.4t$.

For each of these examples, we conducted separate experiments for Local Monitoring (LM) and Global Monitoring (GM). In Local Monitoring, $C_G = \infty$ while for Global Monitoring, $C_L = 0$. For Local Monitoring, the decentralized anytime problem was converted to a Dec-MDP and solved using the bilinear program. For Global Monitoring, the problem was converted to a Dec-MDP-Comm-Sync and solved. We refer to these as Non-Myopic strategies (NM) in the table.

Mean running time is shown in Table 4. The Max Flow problem was larger than the Rock Sample problem (containing more quality levels), thus consumed more time. The global formulations, as opposed to the local formulations, required a subproblem formulation to compute V^* at each communication point, and thus more time elapsed. As expected, solutions with Global Monitoring outperformed their counterparts with merely Local Monitoring, due to the fact that for the GM experiments, C_L was assumed to be zero, and thus local monitoring was free.

6 Conclusions and Future Work

Anytime algorithms effectively gauge the trade-off between time and quality. Monitoring is an essential part of the process. Existing techniques from the literature weigh

Problem	NM Local	NM Global
Max Flow	6.35	6.75
Rock Sample	2.8	3.6

Table 5: Expected utility of joint policies on two profiles.

the trade-off between time, quality, and monitoring for the single-agent case. The complexity of the monitoring problem is known, and dynamic programming methods provide an efficient solution method.

However, this paper shows that these techniques do not scale to the multi-agent case. In this paper, we took a decision-theoretic approach to the monitoring problem. We formalized the problem for the multi-agent case, and proved that there exist problems for both local and global monitoring which are NP-complete. We showed how the multi-agent monitoring problems can be compiled as special cases of Decentralized Markov Decision Processes, and thus solvers from the literature can produce efficient solutions.

Future work lies in several directions. First, we will analyze and produce solutions for monitoring problems that are partially observable. We will also examine items like varying the monitoring cost. Second, we would like to examine cases with non-cooperative utility functions. Third, we will apply the methods to cases involving more than two agents. The latter will require modifications to the bilinear solver.

References

Anderson, M. 2007. A review of recent research in metareasoning and metalearning. *AI Magazine* 28(1):7–16.

Becker, R.; Carlin, A.; Lesser, V.; and Zilberstein, S. 2009. Analyzing myopic approaches for multi-agent communication. *Computational Intelligence* 25(1):31–50.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2001. *Introduction to Algorithms*. Chapter 29.

Cox, M., and Raja, A. In Press. *Metareasoning: Thinking about thinking*. Cambridge, MA: MIT Press.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 49–54. AAAI.

Goldman, C., and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *JAIR* 22:143–174.

Hansen, E., and Zilberstein, S. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence* 126(1-2):139–157.

Horvitz, E. 1987. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of Third Workshop on Uncertainty in Artificial Intelligence*, 429–444. AAAI.

Lesser, V.; Decker, K.; Wagner, T.; Carver, N.; Garvey, A.; Horling, B.; Neiman, D.; Podorozhny, R.; NagendraPrasad, M.; Raja, A.; Vincent, R.; Xuan, P.; and Zhang, X. 2004. Evolution of the gpgp/taems domain-independent coordination framework. In *Autonomous Agents and Multi-Agent Systems*, 87–143.

Petrik, M., and Zilberstein, S. 2009. A bilinear approach for multiagent planning. *JAIR* 35:235–274.

Puterman, M. 2005. *Markov decision processes, Discrete stochastic dynamic programming*. John Wiley And Sons, Inc.

Raja, A., and Lesser, V. 2004. Meta-level reasoning in deliberative agents. In *Proceedings of the International Conference on Intelligent Agent Technology (IAT 2004)*, 141–147.

Raja, A., and Lesser, V. 2007. A framework for meta-level control in multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 15:147–196.

Sandholm, T. 2003. Terminating decision algorithms optimally. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, poster paper. Springer.

Smith, P., and Geddes, N. 2002. *A Cognitive Systems Engineering Approach to the Design* *of Decision Support Systems, The Human Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Earlbaum Associates.

Smith, T., and Simmons, R. 2004. Heuristic Search Value Iteration for POMDPs. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI 2004)*.

Tsitsiklis, J., and Athans, M. 1985. On the complexity of decentralized decision making and detection problems. *IEEE Transactions on Automatic Control* 30(5):440–446.

Zilberstein, S., and Russell, S. 1996. Optimal composition of real-time systems. *Artificial Intelligence* 82(1-2):181–213.