Query Content in Sequential One-shot Multi-Agent Limited Inquiries when Communicating in Ad Hoc Teamwork

William Macke[1], **Reuth Mirsky**[1], **Peter Stone**[1, 2]
[1] The University of Texas at Austin, [2] Sony AI
{wmacke,reuth,pstone}@cs.utexas.edu

Abstract

Communication in Ad Hoc Teamwork (CAT) is a research area that investigates how communication can be leveraged by an agent that plans in a distributed, multi-agent collaborative environment, even if that agent does not have knowledge about its teammates or their plans a priori. This paper reports our progress in identifying three factors that can impact the complexity of CAT – environment, teammates, and communication protocol. Following the identification of these components, this paper investigates three extensions from existing work that affect each of these factors respectively – richer environments, complex teammate representations, and complex communication protocols. We present new algorithms to compute when to query under these new configurations, as well as preliminary results of their performance.

Introduction

Autonomous agents are becoming increasingly capable of solving complex tasks, but encounter many challenges when required to solve such tasks as a team. For example, service robots have been deployed to assist medical teams in the recent pandemic outbreak. Such robots' coordination strategy cannot be learned or decided a priori, as it interacts with previously unmet teammates (Cakmak and Thomaz 2012). This motivation is the basis for *ad hoc teamwork*, which is defined as collaborating with teammates without pre-coordination (Stone et al. 2010; Albrecht and Stone 2018). This terminology reflects that the *collaboration* is ad hoc – the ways in which the agents learn, act, and interact may be quite principled. Our previous work on CAT identified a specific variant of CAT, namely the Sequential One-shot MultiAgent Limited Inquiry CAT scenario, or SOMALI CAT (Mirsky et al. 2020). In SOMALI CAT, the agents execute *sequential plans* and only the ad hoc agent can inquire about a teammate's goal. SOMALI CAT was defined to be a broadly representative class of CAT problems. In such a SOMALI scenario, the robot can fetch different tools for a physician in a hospital. The physician would normally prefer to avoid the additional cognitive load of communicating with the robot, but will answer an occasional question from it so that the robot can be a better collaborator. The results from this work were evaluated on a simulated test-bed, namely *the tool fetching domain*. The algorithm presented in that work was a means to decide when to query in such a SOMALI scenario. This paper reports our progress investigating the different factors that can affect the complexity of a SOMALI scenario: environment, teammates, and communication protocol. An additional contribution is a set of heuristic algorithms for choosing when to query, that are shown to outperform previous work in these complex configurations.

Background

Communicating agents has been a fertile research area in the context of distributed multiagent systems (Singh 1998; Cohen, Levesque, and Smith 1997; Decker 1987). Goldman and Zilberstein (2004) formalized the problem of a decentralized POMDP with communication (DEC-POMDP-com). Communication in Ad-Hoc Teamwork (CAT) is a close problem that shares some similar assumptions: all teammates strive to be collaborative and the agents have a predefined communication protocol available. However, DEC-POMDP-com uses a single model that is collaboratively controlled by multiple agents, whereas CAT is set from the perspective of one agent that has no additional knowledge about its teammates' policies and that it cannot change the properties of these teammates (Stone et al. 2010).

Barrett et al. (2014) considered a scenario in which either teammates are assumed to share a common communication protocol, or else this assumption can be quickly tested on the fly (e.g. by probing). Their work was situated in a very restrictive multi-agent setting, namely a multi-arm bandit, where each task was a single choice of which arm to pull. A different type of CAT scenarios refers to tasks where a single agent reasons about the sequential plans of other agents, and can gain information by querying its teammates or by observing their actions (Mirsky et al. 2020). This Sequential One-shot Multi-Agent Limited Inquiry CAT scenario, or SOMALI CAT, was inspired by the use case of a service robot that is stationed in a hospital, who mainly have to retrieve supplies for physicians or nurses, and has two main goals to balance: understanding the task-specific goals of its human teammates, and understanding when it is appropri-

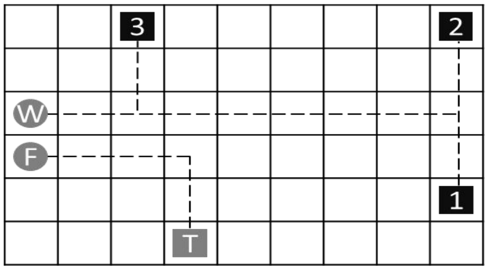

Figure 1: Example of the tool fetching domain. W, F, and T are the locations of the worker, fetcher, and toolbox respectively. The locations of the workstations are represented by the numbered squares.

ate to ask questions over acting autonomously. In such SOMALI CAT scenarios, we have additional assumptions: the task performed requires a sequence of actions; it is also an episodic, one-shot task; the environment is static, deterministic, and fully observable; the teammate is assumed to have perfect knowledge about the environment; the teammate is assumed to plan optimally, given that it is unaware of the ad hoc agent's plans or costs; and there is one communication channel, where the ad hoc agent can query as an action, and if it does, the teammate will forgo its action to reply truthfully (the communication channel is noiseless).

The Tool Fetching Domain Our previous work in SOMALI CAT introduced an experimental domain known as the tool fetching domain. This domain consists of an ad hoc agent, the *fetcher*, attempting to meet a teammate, the *worker*, at some workstation with a tool. The worker needs a specific tool depending on which station is its goal, and the worker's goal is unknown before hand to the fetcher. It is the job of the fetcher to deduce the goal of the worker based on its actions, and to bring the correct tool to the correct workstation. At each timestep, the agents can execute one action each. Additionally, the fetcher can query the worker with questions of the form "Is your goal one of the stations $g_1, g_2 ... g_N$?", where $g_1, \ldots, g_N \subseteq G$ is a subset of all workstations. All queries in the original setup are assumed to have a cost identical to moving one step, regardless of the content of the query. Figure 1 shows an example of this domain where the fetcher is following the path to the toolbox while the worker is following one of the paths to an unknown workstation. In this paper, the domain was implemented as a custom OpenAI Gym environment (Brockman et al. 2016).

When to Query To reason about when to act in the environment and when to query, three different reasoning zones were defined for each query that the ad-hoc agent can ask to disambiguate a subset of goals ($G' \subset G$) from ($G \setminus G'$):

Zone of Branching (Z_B) for a set of goals $G' \subseteq G$ is the set of timesteps from when the ad hoc agent (the fetcher) is required to commit to a specific goal and until the end of the episode, which means the timesteps in which it might take a different action from the one it would have taken if it had perfect knowledge about the teammate's true goal.

Zone of Information (Z_I) for a set of goals $G' \subseteq G$ is the set of timesteps from the beginning of the plan and until there is no longer any ambiguity in the domain between goals in G' and $G \setminus G'$.

Zone of Querying (Z_Q) for a subset of goals $G' \subseteq G$ is the intersection of these two sets of timesteps, where there may be a positive value in querying instead of acting.

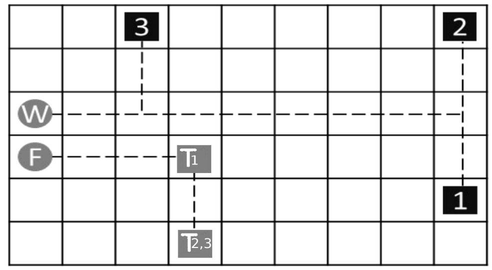

Figure 2: Example of a domain with multiple toolboxes. T_1 houses tool 1 while $T_{2,3}$ houses tools 2 and 3.

Given these zones for each subset of goals G', we can identify the **Critical Querying Point (CQP)** as the first timestep inside $Z_Q(G')$, and is the first timestep in which the ad hoc agent should consider whether to query "Is your goal one of the stations in G'?". If $Z_Q(G')$ is empty, then there is no time in which this query can be useful and $CQP(G') = -1$. In Figure 1, some of the critical querying points are $CQP(\{1\}) = CQP(\{2\}) = 6$, as it takes the fetcher 5 timesteps to reach the toolbox and only then it enters Z_B for goals 1 and 2.Notice that $CQP(\{1, 3\}) = 6$ as well, as in this case $G' = \{1, 3\}$ and $G \setminus G' = \{2\}$, which means that there is still a benefit from disambiguating a group of goals that contains goal 1 and a group of goals that contains goal 2. $CQP(\{3\}) = -1$, as by the time the fetcher reaches the toolbox, the worker has already reached or passed station 3.

Generalizing SOMALI CAT

While previous work successfully demonstrated that each set of possible queries had a unique optimal time to ask, namely the CQP, it used a naive approach of choosing half the relevant goals at random for deciding the *content* of a query when multiple queries share the same CQP. Such an approach also misses the potential in more complicated scenarios, such as when the worker chooses its goal with a non-uniform probability. or when different queries cost different amounts based on their content. In this section we modify some of the assumptions from previous work: having multiple tool stations instead of just one (hence having multiple zones of branching); having a non-uniform distribution over the possible goals the worker might reach; and having different query cost models rather than a unit cost per query.

Multiple Zones of Branching In previous work, domain setups only contained one toolbox that housed all the tools. This meant that effectively all queries had the same CQP regardless of their content, so an efficient strategy was to query about half of the goals randomly at the CQP. However, when there is more than one tool location present, this strategy no longer holds. Figure 2 shows an example when this strategy falls short. In this example, the tool for station 1 is in the top toolbox (T_1) and the tools for goals 2 and 3 are in the bottom toolbox (T_2, T_3). The first point in time when the fetcher might want to query is after it arrives at the toolbox T_1. If it were to query about half the goals randomly, it may ask "Is your goal station 1?" This action is effectively a wasted query, since it does not add new information, which means that the fetcher cannot act upon its current knowledge.

Non-Uniform Goal Distribution Another assumption made in previous work was that the worker is always assigned a goal according to a uniform probability distribution. This assumption may not hold in practice. For instance, if a human were to take the part of the worker, they may be more likely to choose goals that are closer to them. Knowledge of this distribution may allow the fetcher to construct more informative queries. For instance, if there are three goals in a domain, and the worker goal distribution is $g_1 = 0.98, g_2 = 0.01, g_3 = 0.01$ then it is likely better to query about $\{g_1\}$ or $\{g_2, g_3\}$ than about $\{g_2\}$ or $\{g_3\}$ (the fetcher is more likely to learn the worker's true goal with the former queries than with the latter ones).

Query Cost An assumption that was used in previous work is that the cost of querying is uniform, relatively small, and is not affected by the content of the query or its timing. However, it is very likely that a different cost model would result in different performance. If the cost of a query is larger than the value of the information gained, then there will be no benefit from asking such a query. We investigate how effective various query strategies are with different cost models. We consider two variables for a cost model in particular: *base cost (bc)*, or the initial cost of asking any query, and *station cost (sc)*, or the additional cost of including another station in a query. The total cost of asking a query q that asks "Is your goal one of the stations $g_1, g_2, ...g_N$?" is $bc+sc*N$. Importantly, in such a cost model, querying about many goals is more costly than querying about just one goal. This means that in the first running example, querying about $\{g_2, g_3\}$ or $\{g_1\}$ are no longer equal in their potential benefit, and the latter, smaller query becomes preferable.

Query Algorithms

We present a basic objective for determining what to query without any of the new assumptions presented earlier in this section. Previous work would query about half the relevant goals to try and maximize information gain. However reasoning more thoroughly regarding which goals to ask about can give even better performance. Consider the pairs of goals $G_B = \{(g_i, g_j) | t \in Z_B(g_i, g_j)\}$ where t is the current time and $g_i, g_j \in G_B$, a set of all N possible goals that might still be the true goal of the worker. We can construct a binary vector $\vec{x}$ of length N such that if x_i is the i-th value in that vector, then goal g_i is included in the query if and only if $x_i = 1$. A query that asks about a subset of goals $G' \subseteq G$ will disambiguate between the sets G' and $G \backslash G'$. Therefore, to increase the information gained from the query, we want to split the pairs of goals (g_i, g_j) as evenly as possible between G' and $G \setminus G'$. We write the following maximization goal

$$\max \sum_{(g_i,g_j) \in G_B} (x_i \oplus x_j) \quad (1)$$

The term in the objective is 1 only when one x is 0 and the other is 1. This objective ensures that the query disambiguates between as many of the pairs of goals as possible. Consider the example above when the fetcher arrives on toolbox T_1. This approach is guaranteed to now ask either "Is your goal in $\{g_1\}$?" or "Is your goal in $\{g_2, g_3\}$?", both of which are guaranteed to allow the fetcher to act in the next timestep regardless of the worker's answer.

While the above can easily handle multiple tool locations, it can fail under circumstances where there's a non-uniform probability distribution over the worker's possible goals. To reason about such circumstances, we modify the integer program's objective from the previous section to weigh the goals by the ad-hoc agent's current belief state:

$$\max \sum_{(g_i,g_j) \in G_B} (P(g_i) + P(g_j)) * (x_i \oplus x_j) \quad (2)$$

where $P(g_i)$ refers to the probability that the worker's goal is g_i. Intuitively, this new equation will prioritize goals that are more likely. If the worker's goal has the same probability to be either g_i or g_j, than it is most informative to disambiguate between the two. On the other hand, if one goal has a probability of 0.99, as in the example in section , it would be advantageous just to query about that one goal. Incorporating the probabilities of goals in the objective as shown above results in the method prioritizing disambiguating this higher probability goal from others.

Finally, a complete model that is able to reason about all of the extensions presented in the previous section is still required to incorporate different query cost models. Since we want to minimize the needed query cost as part of our objective within the integer program, we add the negative cost of the query to our objective. Consequently, the final integer program objective becomes

$$\max \sum_{(g_i,g_j) \in G_B} (x_i \oplus x_j) * (P(g_i) + P(g_j)) - \sum_i (x_i * sc) \quad (3)$$

where *sc* is the cost of including a goal in a query. This objective now simultaneously attempts to maximize the probability that the ad hoc agent will be able to act in the next timestep and minimize the cost of the query. All objectives shown above were solved with the Coin-Or Integer Program Solver using PuLP as the front end (Forrest et al. 2018; Mitchell, Consulting, and Dunning 2011).

Table 1: The different algorithms used in the experiments.

Never Query	Never Queries but waits until it knows an action is optimal
Random Query	Randomly asks about half the remaining potential goals when in a Z_Q
Max Binary Policy	Optimizes the query according to Equation 1 when in a Z_Q
Goal Prob Policy	Optimizes the query according to Equation 2 when in a Z_Q
Weighted Cost Policy	Optimizes the query according to Equation 3 when in a Z_Q

Results

We hypothesized that our new methods should be able to significantly outperform the previous approach regardless of the cost model used or the worker's probability of choosing goals. The experiments compare 5 different query algorithms, as presented in Table 1. Additionally, if the fetcher is going to query in a given timestep, it may be advantageous to query about goals that are not critical for acting. That is, goals g such that $(g, g') \notin G_B \forall g' \in G$. While it is possible to modify the objective to consider these goals, the size of the integer program quickly increases beyond what is reasonably tractable. Therefore, as a heuristic, we include half of these stations in the query in order to increase information gain. Each of the following experiments has a grid size of 50×50 with 100 stations located randomly. Each station has a required tool that is in one of five random toolbox locations. We assume the cost of all non-querying actions is 1. All results are averaged over 100 random instances where each instance consists of a station and tool locations, the initial fetcher and worker positions and a specific workstation assigned as the worker's goal.

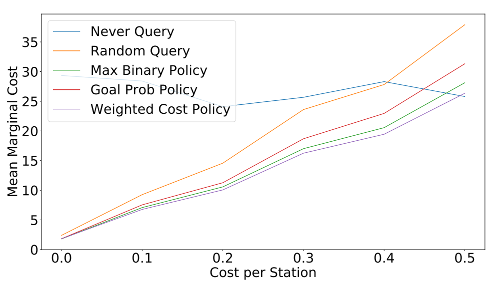

Figure 3: Marginal cost of the fetcher's plan execution given a varying cost per station in a query, under *uniform* distribution of goals for the worker.

Figure 3 shows the marginal plan execution costs over the optimal plan, assuming an oracle that lets the fetcher know what's the worker's true goal. The x-axis shows different additional cost per workstation (sc) in a query, given an initial query cost (bc) of 0.5. The probability of a worker being assigned a goal is *uniform across all 100 goals*. As shown, all query methods decrease in performance as the cost per station increases, however Weighted Cost Policy decreases at a much lower rate compared to other methods and never performs significantly worse than the Never Query method.

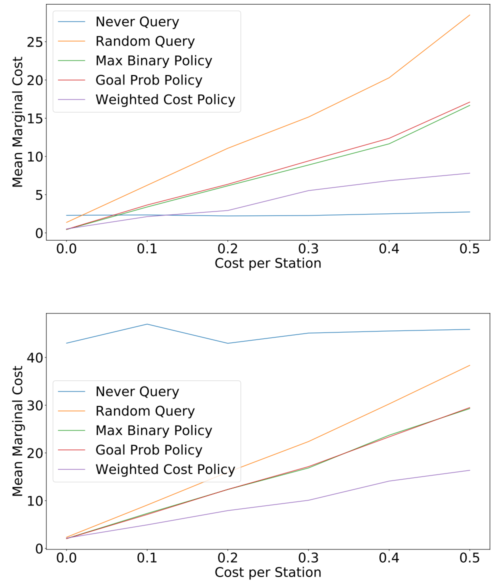

Figure 4: Marginal cost of the fetcher's plan execution when the probability of a worker being assigned to a goal is a softmax of the worker's negative (top) and positive (bottom) distances to a goal.

Figure 4 show marginal costs with various additional cost per workstation in a query, but with a *non-uniform worker goal distribution*. The probability of a worker being assigned a goal as the softmax of the negative and of the positive worker's distances to goals respectively. Intuitively, it respectively defines workers that are most likely to prefer workstations that are closer or farther from their initial location. These graphs present similar results, with the primary difference being the relative performance of the Never Query Strategy. In Figure 4 (top), the worker is much more likely to move to a close workstation, which reduces the maximum time before the fetcher knows the worker's goal. Similarly in Figure 4 (bottom), the worker is more likely to move to a distant station, increasing this maximum time for the fetcher to know its goal, which causes the Never Query strategy to perform better or worse respectively.

Conclusion

We presented several extensions to SOMALI CAT and several novel algorithms for determining what and when to query, and demonstrated their performance in the Tool Fetching Domain. Our new algorithms were able to outperform previous techniques in multiple scenarios. For future work we plan to further generalize our query algorithms to perform well in any SOMALI CAT domain, regardless of the query cost model, probability over goals, or other domain-specific details.

ACKNOWLEDGMENTS

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CPS-1739964, IIS-1724157, NRI-1925082), ONR (N00014-18-2243), FLI (RFP2-000), ARO (W911NF-19-2-0333), DARPA, Lockheed Martin, GM, and Bosch. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research. Studies in this work were approved under University of Texas at Austin IRB study numbers 2015-06-0058 and 2019-03-0139.

References

Albrecht, S. V., and Stone, P. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258:66–95.

Barrett, S.; Agmon, N.; Hazon, N.; Kraus, S.; and Stone, P. 2014. Communicating with unknown teammates. In *AAMAS*, 1433–1434.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym.

Cakmak, M., and Thomaz, A. L. 2012. Designing robot learners that ask good questions. In *ACM/IEEE international conference on Human-Robot Interaction*.

Cohen, P. R.; Levesque, H. J.; and Smith, I. A. 1997. On team formation. *Synthese Library* 87–114.

Decker, K. S. 1987. Distributed problem-solving techniques: A survey. *IEEE transactions on systems, man, and cybernetics* 17(5):729–740.

Forrest, J.; Ralphs, T.; Vigerske, S.; LouHafer; Kristjansson, B.; jpfasano; EdwinStraver; Lubin, M.; Santos, H. G.; rlougee; and Saltzman, M. 2018. coin-or/cbc: Version 2.9.9.

Goldman, C. V., and Zilberstein, S. 2004. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of artificial intelligence research* 22:143–174.

Mirsky, R.; Macke, W.; Wang, A.; Yedidsion, H.; and Stone, P. 2020. A penny for your thoughts: The value of communication in ad hoc teamwork. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Mitchell, S.; Consulting, S. M.; and Dunning, I. 2011. Pulp: A linear programming toolkit for python.

Singh, M. P. 1998. Agent communication languages: Rethinking the principles. *Computer* 31(12):40–47.

Stone, P.; Kaminka, G. A.; Kraus, S.; and Rosenschein, J. S. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*.