Cost-awareness in Multi-Agent Active Search

Arundhati Banerjee[a;*], **Ramina Ghods**[a] **and Jeff Schneider**[a]

[a]School of Computer Science, Carnegie Mellon University

Abstract. Multi-agent active search requires autonomous agents to choose sensing actions that efficiently locate targets. In a realistic setting, agents also must consider the costs that their decisions incur. Previously proposed active search algorithms simplify the problem by ignoring uncertainty in the agent's environment, using myopic decision making, and/or overlooking costs. In this paper, we introduce an online active search algorithm to detect targets in an unknown environment by making adaptive cost-aware decisions regarding the agent's actions. Our algorithm proposes an online lookahead planner that combines priniciples from Monte Carlo Tree Search, Thompson sampling and pareto-optimal confidence bounds for decentralized multi-agent multi-objective optimization in an unknown environment. We analyze the algorithm's performance in simulation to show its efficacy in cost-aware active search.

1 Introduction

Active search [12, 13] refers to the task of efficiently discovering members of a desired class (targets) by making online sequential adaptive data collection decisions. For example, real world tasks like search and rescue or environmental monitoring using autonomous robots (agents) are best formulated as active search problems where the agents must not only detect targets accurately but should do so while minimizing the usage of resources like energy and time.

Previous studies have used various constraints and reductions to achieve resource efficient adaptive search. Such algorithms generally include parameters determining the trade-off between the informativeness of the collected data with the cost of such data collection [43]. Adaptive sensing applications in robotics typically reduce it to a planning problem assuming full observability of the environment [39, 25]. Imposing a cost budget in such applications is modeled as constrained path planning between the known start and goal locations. Unfortunately, this is in contrast with the real world where the agent's environment, the number of targets and their locations may be unknown and the agent may have access only to noisy observations from sensing actions. All these factors increase the difficulty of cost effective active search.

Besides cost efficiency, executing active search with multiple agents creates an additional challenge. Centralized planning in a multi-agent setting is often impractical due to communication constraints [53, 42]. Further, a real world system dependent on a central coordinator that expects synchronicity from all agents is susceptible to communication or agent failure. Instead, we assume an asynchronous inter-agent communication setting where each agent is able to individually plan and execute its next sensing action using whatever information it already has and happens to receive. Note that in our problem formulation, the agents are not entirely independent actors and therefore still share information with their peers in the team when possible. However, we do not require them to communicate synchronously and instead enable decentralized and asynchronous multi-agent active search (Figure 1b).

* Corresponding Author. Email: arundhat@andrew.cmu.edu.

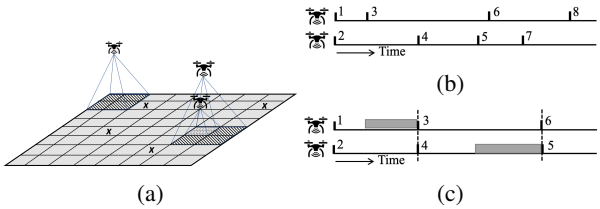


Figure 1: **Problem setup. (a)** Region sensing active search in an 8×8 grid with 3 agents looking for 4 targets, denoted $\mathbf{x}$. The agent sensing the 1×1 region is closer to the grid and receives a more accurate observation than agents sensing the 2×2 regions who are farther away and receive noisier observations. **(b, c)** Decentralized multi-agent asynchronous (b) vs synchronous (c) performance. The small vertical lines indicate the start of the q-th task. The shaded regions in (c) indicate idle time waiting for the other agent to complete its task.

Contributions. In this paper, we develop a novel cost-aware asynchronous multi-agent active search algorithm called CAST (Cost-Aware Active Search of Sparse Targets) to enable agents to detect sparsely distributed targets in an unknown environment using noisy observations from region sensing actions, without any central control or synchronous communication (Figure 1). CAST combines Thompson sampling with Monte Carlo Tree Search for lookahead planning and decentralized multi-agent decision making. CAST introduces Lower Confidence Bound (LCB) style pareto optimization for cost-aware action selection by trading-off the expected future reward from sensing actions with their associated costs. We demonstrate the efficacy of CAST with a set of simulation results across different team sizes, number of targets and size of the search space.

2 Related work

Informative Path Planning. Autonomous target search has diverse applications in environment monitoring [40], wildlife protection [33] as well as search and rescue operations [18]. In robotics, informative path planning (IPP) problems focus on adaptive decision making to reach a specified goal state or region. In contrast to our active search setting, common IPP algorithms consider a known environment [37], are myopic or use non-adaptive lookahead [49], and assume weakly coupled sensing and movement actions [8].

Bayesian optimization. Bayesian optimization and active learning methods are another approach to active search [36, 41, 22]. Unfortunately, they mostly address single-agent systems, or if multi-agent

they assume central coordination [2, 21] and except for [35] lack any realistic assumptions on sensing actions. Multi-agent asynchronous active search algorithms proposed in [14, 15] tackle several of these challenges but they are myopic in nature. Further, [20] introduced cost effective non-myopic active search but their setting is simplified assuming a known number of targets, constant cost for actions and excluding simultaneous evaluation of multiple search points.

POMDPs and Dec-POMDPs. Our active search formulation has close similarities with planning under uncertainty using a Partially Observable Markov Decision Process (POMDP) [23]. Monte Carlo Tree Search (MCTS) [26, 6] has found success as a generic online planning algorithm in large POMDPs [48] and recently, [11, 10] proposed single agent MCTS algorithms for (single objective) adaptive decision making using information theoretic reward in continuous state and observation domain POMDPs.

Decentralized POMDP (Dec-POMDP) [4, 38] is another framework for decentralized active information gathering using multiple agents which is typically solved using offline, centralized planning followed by online, decentralized execution [28, 27]. Recently, [19] formulated multi-agent active search using a deep reinforcement learning framework that depends on centralized training followed by local inference at test time assuming a known team size. Decentralized MCTS (Dec-MCTS) algorithms have also been proposed for multi-robot active perception under a cost budget [50, 5] but they typically rely on each agent optimizing for a known global objective while maintaining a joint probability distribution over its own belief as well as those of the other agents, that helps ensure inter-agent coordination.

Cost-budgeted planning. Finally, cost aware active search is a multi-objective sequential decision making problem. [30] developed an MCTS algorithm for single agent cost budgeted POMDPs using a scalarized version of the reward-cost trade-off, whereas [52] introduced multi-objective MCTS (MO-MCTS) for discovering global pareto-optimal decision sequences in the search tree. Unfortunately, MO-MCTS is computationally expensive and unsuitable for online planning. [7] proposed the Pareto MCTS algorithm for multi-objective IPP but they ignore uncertainty due to partial observability in the search space.

3 Problem definition

Figure 1a depicts the idea behind cost-aware active search. Each agent is actively sensing regions of the search space looking for targets. To plan its next sensing action, the cost-aware agent has to trade-off the expected future reward of detecting a target with the overall costs it will incur in travelling to the appropriate location and executing the action. Given previous observations, it adaptively makes such data-collection decisions online while minimizing the associated costs as much as possible. Unfortunately, this problem is NP-hard [32].

Notation. Lowercase and uppercase boldface letters represent column vectors and matrices respectively. $\mathbf{A}^T$ is the transpose for $\mathbf{A}$. The ith entry of a vector $\mathbf{a}$ is $[\mathbf{a}]_i$.

Sensing setup. We first describe our setup for active search with region sensing. We consider a gridded search environment of size n described by a sparse matrix which we want to recover through multi-agent active search. $\boldsymbol{\beta} \in \{0,1\}^{n\times 1}$ denotes the flattened vector representation of the environment having k non-zero entries at the true locations of the k OOIs. $\boldsymbol{\beta}$ is the ground truth search vector. The sensing model for an agent j at time t is

$$y_t^j = \mathbf{x}_t^{j\mathrm{T}}\boldsymbol{\beta} + \epsilon_t^j, \text{ where } \epsilon_t^j \sim \mathcal{N}(0,\sigma^2). \tag{1}$$

$\mathbf{x}_t^j \in \mathbb{R}^n$ is the (flattened) sensing action at time t, with a non-zero support over the sensing region and zeros elsewhere. y_t^j is the agent's observation and ϵ_t^j is a random, i.i.d added noise. The tuple $(\mathbf{x}_t^j, y_t^j)$ is agent j's measurement at time t. We define our action space $\mathcal{A}$ ($\mathbf{x}_t^j \in \mathcal{A}$) to include only hierarchical spatial pyramid sensing actions [29]. For example in Figure 1a, the two agents in the bottom right half of the search space are sensing hierarchical 2×2 and 1×1 regions respectively. Note that the agents using this region sensing model must trade-off between sensing a wider area with lower accuracy versus a highly accurate sensing action over a smaller region. The support of the vector $\mathbf{x}_t$ is appropriately weighted so that $\|\mathbf{x}_t\|_2 = 1$ to ensure each sensing action has a constant power. This helps us in modeling observation noise as a function of the agent's distance from the region [14]. Since each action has a constant power and every observation has an i.i.d added noise with a constant variance, the signal to noise ratio in the unit squares comprising the rectangular sensing block reduces as the size of the sensing region is increased with increasing distance between the agent and that region.

Cost model. We introduce the additional realistic setting that sensing actions have different associated costs. First, we consider that the agent travelling from location a to location b incurs a travel time cost $c_d(a,b)$. [1] Second, we assume that executing each sensing action incurs a time cost c_s. Therefore, T time steps after starting from location x_0, an agent j has executed actions $\{\mathbf{x}_t^j\}_{t=1}^T$ and incurs a total cost defined by $C^j(T) = \sum_{t=1}^T (c_d(x_{t-1}^j, x_t^j) + c_s)$.

Communication setup. We assume that communication, although unreliable, will be available sometimes and the agents should take advantage of it when possible. In our setting, agents may communicate their own measurements asynchronously with other agents, without having to wait on such communications from their peers at any time. Figures 1b and 1c contrast the asynchronous vs synchronous communication setup, showing that in an asynchronous setting, each agent can continue its individual cost-aware decision making as soon as it has finished executing its previous action. We also do not require that the set of available past measurements remain consistent across agents since communication unreliability prevents it.

In the multi-agent setting, since agents share their observations asynchronously, an agent j decides its next cost-aware action at time t using all its past measurements as well as those received from its teammates till time t i.e. $\mathbf{D}_t^j = \{(\mathbf{x}_{t'}, y_{t'}) | \{t'\} \subseteq \{1,\dots,t-1\}\}$, $|\mathbf{D}_t^j| \leq t-1$. For example, the second agent ($j=2$) in the multi-agent example in Figure 1b starts task 5 before task 3 is completed with $\mathbf{D}_5^2 = \{(\mathbf{x}_{t'}, y_{t'}) | t' = \{1,2,4\}\}$.

4 Our proposed algorithm: CAST

4.1 Background

We first briefly describe the concepts essential to the planning and decision making components of our algorithm.

Monte Carlo Tree Search (MCTS) is an online algorithm that combines tree search with random sampling in a domain-independent manner. In our setup, a cost-aware agent would benefit from the ability to lookahead into the adaptive evolution of its belief about the target's distribution in the environment in response to possible observations from the actions it might execute. We therefore consider MCTS as the basis for developing our online planning method with finite horizon lookahead. But the presence of uncertainty about targets (number and location) in the unknown environment together with the

[1] We assume a constant travelling speed and compute the Euclidean distance between locations a and b.

noisy observations introduces additional challenges in our problem formulation that are not commonly addressed in the MCTS literature.

Pareto optimality: Our formulation of cost aware active search described in Section 3 can be viewed as a multi-objective sequential decision making problem where the agent trades-off the expected reward with the incurred cost. A common approach to solving such multi-objective optimization problems is scalarization i.e. considering a weighted combination of the different objectives resulting in a single-objective problem. However, tuning the weight attributed to each objective is challenging since they might be scaling quantities having different units and their relative importance might be context dependent. In contrast, pareto optimization builds on the idea that some solutions to the multi-objective optimization problem are categorically worse than others and are *dominated* by a set of pareto-optimal solution vectors forming a pareto front for the optimization objective. Considering a set of D-dimensional vectors $\mathbf{g} \in \mathcal{G}$, we define the following:

• $\mathbf{g}$ dominates $\mathbf{g}'$ (i.e. $\mathbf{g} \succ \mathbf{g}'$) iff: (1) $\forall d \in \{1, \dots, D\}, [\mathbf{g}]_d \geq [\mathbf{g}']_d$ (2) $\exists d \in \{1, \dots, D\}, [\mathbf{g}]_d > [\mathbf{g}']_d$

• $\mathbf{g}$ and $\mathbf{g}'$ are incomparable (i.e. $\mathbf{g} || \mathbf{g}'$) iff: $\exists d_1, d_2 \in \{1, \dots, D\}$, $[\mathbf{g}]_{d_1} > [\mathbf{g}']_{d_1}$ *and* $[\mathbf{g}]_{d_2} < [\mathbf{g}']_{d_2}$

• $\mathcal{G}^* \subseteq \mathcal{G}$ is the pareto-front of $\mathcal{G}$ iff: (1)$\forall \mathbf{g} \in \mathcal{G}$ and $\forall \mathbf{g}' \in \mathcal{G}^*$, $\mathbf{g} \not\succ \mathbf{g}'$ (2) $\forall \mathbf{g}, \mathbf{g}' \in \mathcal{G}^*, \mathbf{g} || \mathbf{g}'$.

In our algorithm, each agent estimates a reward-cost vector to evaluate its candidate actions and chooses the next sensing action from a pareto-optimal set of such vectors.

Thompson sampling (TS) [51] is an exploration - exploitation algorithm that has been studied in a number of bandit and reinforcement learning (RL) settings [17, 44, 16]. TS balances exploration with exploitation by choosing actions that maximize the expected reward assuming that a sample drawn from the posterior distribution is the true state of the world. Prior work in [3, 31] note that exploration in TS is reward oriented, leaning heavily on the drawn posterior sample. Moreover TS only explores on the seemingly optimal policies, which is a disadvantage in environments where knowledge-seeking actions having lower immediate reward are crucial for the agent's efficient long-term performance. This is especially relevant for cost-aware active search wherein actions which are not immediately rewarding in terms of having detected a target may still be informative, for example by reducing the uncertainty regarding the presence of targets in a certain part of the search space. In this work, we build upon these insights and combine TS with MCTS to develop a search tree building strategy for online lookahead planning in partially observable environments.

Additionally, posterior sample based action selection makes TS an excellent candidate for a decentralized multi-agent decision making algorithm [24] and it has been shown to be effective in multi-agent active search with myopic planning [14, 15]. In this work, we will show that our TS based lookahead planning algorithm enables decentralized multi-agent active search with no pre-coordination and minimum communication overhead among agents, unlike existing multi-agent algorithms that rely on pre-designed movement coordination or communication and update of joint probability distributions.

4.2 Belief representation and reward formulation

Next, we will describe two key components of our active search algorithm: the agent's posterior belief representation over the search space and the reward formulation for sensing actions. Note that the agents are unaware of the number of targets or their positions.

Belief representation: Following the setting described in Section 3, consider J agents searching for k sparsely located targets in an unknown environment and $\boldsymbol{\beta}$ is the search vector we want to recover. The agent's belief $b(\boldsymbol{\beta})$ over $\boldsymbol{\beta}$ is a continuous probability distribution over the search space. For any agent j, the prior belief is modeled by $b_0^j = \mathrm{P}(\boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ and the likelihood function following the sensing model (1) is given by $\mathrm{P}(y_t^j | \boldsymbol{\beta}, \mathbf{x}_t^j) = \mathcal{N}({\mathbf{x}_t^j}^T \boldsymbol{\beta}, \sigma^2)$. Therefore, at time step t, its posterior belief over $\boldsymbol{\beta}$ is denoted $b_t^j = \mathrm{P}(\boldsymbol{\beta} | \mathbf{D}_t^j \cup \{\mathbf{x}_t^j, y_t^j\}) = \mathcal{N}(\boldsymbol{\mu}_t^j, \boldsymbol{\Sigma}_t^j)$. In the multi-agent setting, each agent j maintains its own posterior belief $b_t^j(\boldsymbol{\beta})$ and MAP estimate $\hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}_t^j, y_t^j\})$.

Reward formulation: Prior work in multi-agent active search [14, 15] focuses on myopic decision making using TS and proposes choosing the action for agent j that maximizes its one-step lookahead reward assuming that a sample drawn from the posterior $\boldsymbol{\beta}_t^j \sim b_t^j(\boldsymbol{\beta})$ is the true state of the world. Such algorithms select $\mathbf{x}_t^j$ that maximizes $\mathbb{E}_{y_t^j | \mathbf{x}_t^j, \boldsymbol{\beta}_t^j}\left[\lambda(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+1}^j)\right]$ where $\lambda(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+1}^j) = -\|\boldsymbol{\beta}_t^j - \hat{\boldsymbol{\beta}}(\mathbf{D}_{t+1}^j)\|_2^2$ and $\mathbf{D}_{t+1}^j = \mathbf{D}_t^j \cup \{\mathbf{x}_t^j, y_t^j\}$. This ensures that the agents will keep exploring the search space as long as there is uncertainty in the posterior samples $\boldsymbol{\beta}_t^j$, while the posterior belief distribution will contain uncertainty as long as there are unexplored or less explored locations in the search space.

In contrast, the cost-aware active search agents reason about the reward obtained by identifying targets, per unit cost incurred from executing such sensing actions over a finite horizon lookahead. But we note that $\lambda(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+1}^j) \leq 0$, therefore maximizing $\frac{\lambda}{c_d + c_s}$ would erroneously favour costlier actions for the same reward. Instead, we propose using the difference $\lambda^-(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+1}^j) = \max\{0, \|\boldsymbol{\beta}_t^j - \hat{\boldsymbol{\beta}}(\mathbf{D}_t^j)\|_2^2 - \|\boldsymbol{\beta}_t^j - \hat{\boldsymbol{\beta}}(\mathbf{D}_{t+1}^j)\|_2^2\}$ as the one-step lookahead reward. We design λ^- to encourage information gathering by favoring actions $\mathbf{x}_t$ that reduce the uncertainty in the posterior sample $\boldsymbol{\beta}_t^j$ over consecutive time steps. Additionally, $\lambda^-(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+1}^j) \geq 0$. Now, we can compute the u-step lookahead reward $R^u(\mathbf{x}_t, \boldsymbol{\beta}_t^j)$ over the action sequence $\mathbf{x}_{t:t+u}$ as the γ-discounted expected sum of λ^- over u steps.

$$R^u(\mathbf{x}_t, \boldsymbol{\beta}_t^j) = \mathbb{E}_{y_{t:t+u}}\Big[\sum_{\Delta t=1}^{u} \gamma^{\Delta t - 1} \lambda^-(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+\Delta t}^j)\Big] \qquad (2)$$

Following the discussion in Section 4.1 about TS, we observe that the reward computation in (2) is dependent on the posterior sample $\boldsymbol{\beta}_t^j$. Particularly, $\lambda^-(\boldsymbol{\beta}_t^j, \mathbf{D}_{t+1}^j)$ is higher for sensing actions $\mathbf{x}_t$ that identify the non-zero support elements of the vector $\boldsymbol{\beta}_t^j$. Further, maximizing $R^u(\mathbf{x}_t, \boldsymbol{\beta}_t^j)$ over all sequences $\mathbf{x}_{t:t+u}$ for a sampled $\boldsymbol{\beta}_t^j$ would exacerbate this problem by choosing a series of point sensing actions that identify the non-zero support of the particular sample. As a result, the agent might overlook possible region sensing actions that would help reduce the uncertainty in its posterior belief, but that do not immediately identify the support of $\boldsymbol{\beta}_t^j$. Section 8.2 of [44] also highlights this drawback of employing TS based exploration in active learning problems that require a careful assessment of the information gained from actions. In order to overcome these challenges, we propose generalizing the posterior sampling step to a sample size greater than one and combine the information from these samples using confidence bounds over λ^- to evaluate the corresponding sensing actions. To further clarify these design details, we now describe our new algorithm CAST, outlined in Algorithm 1.

4.3 CAST: Cost-Aware Active Search of Sparse Targets

At each time step t, on the basis of the set $\mathbf{D}_t^j$ of past measurements (its own measurements as well as those communicated by other agents), the agent j decides its next region sensing action $\mathbf{x}_t^j$ using the SEARCH procedure of Algorithm 1. It starts with an empty tree

$\mathcal{T}_t^j$ having just a root node and gradually builds it up over m episodes. We assume a maximum tree depth d_{max}. Our search tree has two types of nodes - belief nodes and action nodes. A belief node h is identified by the history of actions and observations accumulated in reaching that node. An action node (h, a) is identified by the action $\mathbf{a}$ taken at the immediately preceding belief node h in the search tree. The root as well as the leaves are belief nodes.

Algorithm 1 Cost-Aware Active Search of Sparse Targets

1: **procedure** MAIN ▷ Executed on each agent j
2: **for** t in $\{1, 2, \ldots\}$ **do**
3: $\mathbf{x}_t^j$ = SEARCH($\mathbf{D}_t^j, x_{t-1}^j, b_t^j$)
4: Execute action $\mathbf{x}_t^j$. Observe y_t^j.
5: Update $\mathbf{D}_{t+1}^j = \mathbf{D}_t^j \cup \{\mathbf{x}_t^j, y_t^j\}$
6: Share $\{\mathbf{x}_t^j, y_t^j\}$ asynchronously with teammates.
7: Update belief b_{t+1}^j and estimate $\hat{\boldsymbol{\beta}}(\mathbf{D}_{t+1}^j)$.
8: **procedure** SEARCH($\mathbf{D}$, x, b)
9: Search tree $\mathcal{T} = \phi$
10: **for** each episode $m' \in \{1 \ldots m\}$ **do**
11: Sample $\boldsymbol{\beta} \sim b$. Discretize $\boldsymbol{\beta}$ to get $\boldsymbol{\beta}_{m'}$.
12: SIMULATE($\boldsymbol{\beta}_{m'}, \mathbf{D}, x, 0$)
13: $\mathcal{A}^*$ =ParetoOptimalActionSet($\mathcal{T}$)
14: $\mathbf{a}^* = \text{argmax}_{\mathbf{a}}\{\frac{\mathbf{a}.r^{LCB}}{\mathbf{a}.cost} | \mathbf{a} \in \mathcal{A}^*\}$
15: **return** $\mathbf{a}^*$
16: **procedure** SIMULATE($\boldsymbol{\beta}, \mathbf{D}, x, depth$)
17: $n(h) \leftarrow n(h) + 1$ ▷ Denote root (belief) node as h
18: **if** $depth = d_{\text{max}}$ **then return** $0, 0$ ▷ Reached leaf node
19: **if** $\lfloor n(h)^{\alpha_s} \rfloor > \lfloor (n(h)-1)^{\alpha_s} \rfloor$ **then**
20: add new child action node (h, a)
21: **else** select action node (h, a) using (3)
22: $n(h, a) \leftarrow n(h, a) + 1$
23: $o \leftarrow \mathbf{a}^{\mathrm{T}}\boldsymbol{\beta}$, $\mathbf{D}' \coloneqq \mathbf{D} \cup \{\mathbf{a}, o\}$
24: **if** o was not previously observed at (h, a) **then**
25: append new node h' due to o in branch hah'
26: $r_{h'} = \lambda^-(\boldsymbol{\beta}, \mathbf{D}'), c_{h'}(x, a) = c_d(x, a) + c_s$
27: Update $r_{h'}^{LCB}$ and $\mathbf{g}_{h'} = \begin{bmatrix} r_{h'}^{LCB} & -c_{h'}(x, a) \end{bmatrix}^{\mathrm{T}}$
28: r', c' = SIMULATE($\boldsymbol{\beta}, \mathbf{D}', a, depth + 1$)
29: $r'' = r_{h'} + \gamma \times r', c'' = c_{h'} + c'$
30: $\bar{Q}^{UCT}(h, a) = \frac{\bar{Q}^{UCT}(h,a) \times (n(h,a)-1) + \frac{r''}{c''}}{n(h,a)}$
31: LCBParetoFrontUpdate(h')
32: LCBParetoFrontUpdate((h, a))
33: **return** r'', c''

Each episode $m' \in \{1, \ldots, m\}$ consists of the following steps.

1. *Sampling*: First, a posterior sample is drawn at the root node from the belief $b_t^j = \mathrm{P}(\boldsymbol{\beta}|\mathbf{D}_t^j)$ and discretized into a binary vector $\boldsymbol{\beta}_{m',t}^j \in \{0, 1\}^n$ (Line 11).
2. *Selection and Expansion*: Starting at the root node, a child action node selection policy (tree policy) is applied recursively at every belief node h in a top-down depth-first traversal till a leaf node is reached. In order to prevent tree width explosion with a larger action space, the progressive widening parameter α_s (Line 19) [9] determines when a new action node is added to the tree. For adding a new child action node at h (Line 20), we select the action $\mathbf{a}$ corresponding to the largest change in posterior belief entropy per unit execution cost $c(x, a)$ assuming $\boldsymbol{\beta}_{m',t}^j$ is the true state of the world. Note that x indicates the agent's location corresponding to belief node h. Arriving at any action node (h, a), the corresponding maximum likelihood observation $o = \mathbf{a}^T\boldsymbol{\beta}_{m',t}^j$ is computed (Line 23) which helps transition to its child belief node h'. The 1-step reward $\lambda_{m'}^-$ for $\boldsymbol{\beta} = \boldsymbol{\beta}_{m',t}^j$ and associated execution cost is computed at each belief node visited in m' (Line 26). Every belief and action node in m' also updates the count of times it has been visited so far ($n(h)$, $n(h, a)$ respectively).
3. *Backpropagation*: Once d_{max} is reached, the lookahead rewards and associated costs are backpropagated up from the leaf to each belief and action node visited in m'. Each action node stores the discounted reward per unit cost averaged over $n(h, a)$ simulations in the subtree rooted at that node (Line 30). Further, each belief and action node builds a reward-cost pareto front (Lines 31 and 32) using the backed up values from their respective subtrees which is utilized in deciding $\mathbf{x}_t$ after m episodes (Line 13).

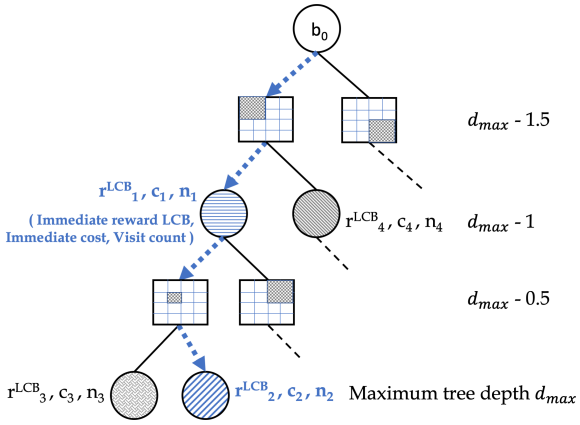
Tree policy. UCT (Upper Confidence Bound applied to trees) [26] is the tree policy used in most MCTS implementations to balance exploration-exploitation in building the search tree. UCT exploits action nodes based on their lookahead reward estimates averaged over past episodes but does not account for the inter-episode variance in such rewards. In our setting, the lookahead reward at any action node in an episode m' depends on the posterior sample $\boldsymbol{\beta}_{m',t}^j$ drawn at the root node and this stochasticity leads to sample variance especially when the particular action node has been visited in only a few episodes. We can account for this variance using the UCB-tuned policy [1] to guide action node selection. Besides, [45] formalized a correction to the UCT formula in an MDP framework replacing its logarithmic exploration term with an appropriate polynomial. We extend it to our tree policy in CAST, called CAST-UCT (3), by combining it with UCB-tuned to balance exploration with exploitation while building the search tree in our partially observable state space. Specifically, CAST-UCT chooses

$$\mathbf{a}^* = \underset{\mathbf{a}}{argmax}\, Q(h, a) + \sqrt{\frac{2\sigma_{h,a}\sqrt{n(h)}}{n(h, a)}} + \frac{16\sqrt{n(h)}}{3n(h, a)} \quad (3)$$

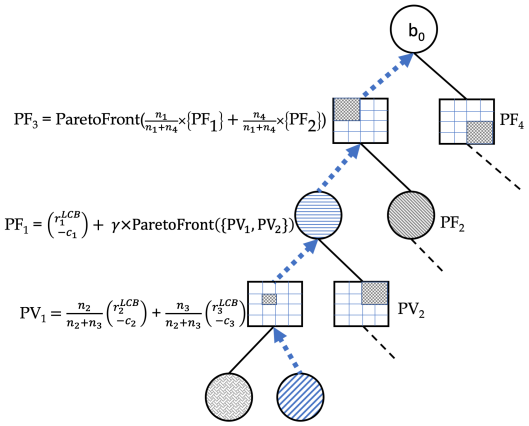
where $\sigma_{h,a}^2$ is the variance of the episode wise terms averaged in $Q(h, a)$ (Line 30).

Pareto front construction with confidence bounds. During the selection and expansion phase in any episode m', the one-step lookahead reward $\lambda_{m'}^-$ is computed at each visited belief node h (Line 26). We note that $\lambda_{m'}^-$ depends on the posterior sample $\boldsymbol{\beta}_{m'}$ drawn for that episode. Assuming that a belief node h is visited in $n(h)$ episodes so far in the search tree $\mathcal{T}_t$, we account for the stochasticity in the computed λ^- by maintaining the Lower Confidence Bound (LCB) of these rewards (denoted r_h^{LCB}) using the Student's t-distribution to estimate a 95% confidence interval (Line 27). Denoting the cost of executing the action that transitions into the belief node h as c_h (Line 26), we define a LCB based immediate (one-step lookahead) reward-cost vector at h, $\mathbf{g}_h = \begin{bmatrix} r_h^{LCB} & -c_h \end{bmatrix}^{\mathrm{T}}$ which is essential to our multi-objective decision making as described next. Figure 2a highlights these variables updated during the selection and expansion phase in one episode.

Next, we compute the pareto front over the lookahead reward-cost vectors at tree nodes visited during the backpropagation phase in episode m'. Figure 2b illustrates this process. Note that the search tree depth at the leaf nodes is d_{max} and consecutive action and belief nodes differ in depth by 0.5. The lookahead reward-cost vector at the action node at depth $d_{\text{max}} - 0.5$ is the weighted average of the reward-cost vectors $\mathbf{g}_{h_\ell}$ of all its leaves h_ℓ, weights being in proportion of



(a)



(b)

Figure 2: **Illustration of a search tree $\mathcal{T}_t$ with $d_{\max} = 2$.** b_0 is the belief at the root node. The action nodes (rectangles) indicate region sensing actions. The belief nodes (circles) are shaded to indicate the evolving posterior belief in the search tree. (a) Illustration of top-down traversal for episode m'. r_1^{LCB}, c_1, n_1, r_2^{LCB}, c_2 and n_2 are updated. (b) Illustration of bottom-up traversal for episode m'. $PV_{\{1,2\}}$ are vectors, $PF_{\{1,2,3,4\}}$ are the pareto fronts at the respective belief and action nodes. γ is the discount factor. ParetoFront() obtains the pareto-optimal vectors from an input set. During backpropagation in episode m', only PV_1, PF_1 and PF_3 are updated corresponding to nodes encountered during top-down traversal.

their visits (eg. PV_1 in Figure 2b). Next, the belief node at depth $d_{\text{max}} - 1$ builds a pareto front from the lookahead reward-cost vectors of all its children action nodes. It then takes a discounted sum of its immediate reward-cost vector with this pareto front to build its lookahead reward-cost vector *set* (eg. PF_1 in Figure 2b).

Repeating these steps in episode m' all the way up to the root node, we alternate between the following: 1) every action node builds its lookahead reward-cost vector set as the pareto front computed from the weighted average of the lookahead vectors of its children belief nodes 2) every belief node builds its lookahead reward-cost vector set by taking the discounted sum of its immediate reward-cost vector with the pareto front obtained from its children action nodes. Note that all reward-cost vectors use the LCB of the rewards. Therefore, at the end of m episodes, each child action node at the root has an LCB based pareto-optimal set of lookahead reward-cost vectors. $\mathcal{A}_t^*$ (Line 13) is the pareto front over such action nodes at the root.[2] Finally, the agent selects the action node having the maximum value of reward per unit cost among its vectors in $\mathcal{A}_t^*$ (Line 14). This completes the agent's decision making step at time t.

Remark 1. In summary, posterior sampling based lookahead planning enables *decentralized* and *asynchronous* multi-agent decision making in CAST so that each agent can select and execute region sensing actions using its current posterior belief, which is updated with its own previous measurements *and* those received from other agents. Additionally, LCB based pareto front construction helps select actions taking into account the sample variability in multi-step reward-cost trade-off computation.

5 Results

We evaluate CAST by comparing in simulation the total cost incurred during multi-agent active search using cost-aware agents against the cost-agnostic active search algorithms SPATS [14] and RSI [35]. SPATS is a TS based algorithm for multi-agent active search, whereas RSI chooses sensing actions that maximize its information gain. We consider sequential point sensing (PS) for exhaustive coverage.

Remark 2. Our purpose in contrasting these baselines with CAST is to demonstrate the benefit of an explicit cost-aware approach to multi-agent active search. To the best of our knowledge, there are no other *cost-aware active search* baselines that we could compare to CAST. Moreover, pareto-optimality in multi-agent multi-objective decision making under uncertainty is not a well studied setting, so comparison to such algorithms is not considered in our problem setup.

In our experiments, we focus on 2-dimensional (2D) search spaces discretized into square grid cells of width 10m. An agent can move horizontally or vertically at a constant speed of 5m/s. Each sensing action incurs a fixed cost of c_s seconds (s), in addition to the travel time between sensing locations. We note that the cost-aware active search strategy may differ depending on the relative per unit magnitudes of sensing cost c_s and travel cost c_d. Hence, for each setting, we will vary $c_s \in \{0\text{s}, 50\text{s}\}$ to simulate high travel cost and high sensing cost respectively. Throughout, any agent is allowed to consider only hierarchical spatial pyramid sensing actions. Our goal is to estimate the k-sparse signal $\boldsymbol{\beta}$ by detecting all k targets with a team of J agents.

The search vector $\boldsymbol{\beta}$ is generated as a randomly uniform k-sparse vector. All the algorithms are unaware of k and the generative prior. We set the signal-to-noise ratio to 16. For CAST, we set $\gamma = 0.97$ and $\alpha_s = 0.5$. The hyperparameters in SPATS and RSI follow [14, 35].

In all our experiments we allow the agents to continue searching the space until all targets have been recovered. Then, across multiple random trials we measure the mean and standard error (s.e.) of the *total cost incurred by the team* (not to be confused with cost per agent) in recovering all k targets. We also plot the mean and s.e. of the full recovery rate achieved as a function of the total cost incurred. The *full recovery rate* is defined as the fraction of targets in $\boldsymbol{\beta}$ that are correctly identified. [3] All agents start from the same location at one corner of the search space, fixed across trials. But the exact instantiation of the search space varies across trials in terms of the position of the targets.

Remark 3. The size of the action space in our experiments is larger than what MCTS algorithms commonly deal with, unless they are

[2] With increasing tree width, the computational complexity of building the pareto front at each action node increases, but the size of the pareto front does not explode. Note that the reward function is adaptive submodular and the cost is monotonic increasing with number of actions, so we do not get a dense pareto-front.

[3] The full recovery rate is 1 when the agents have no false positives or false negatives in their estimated target locations based on their posterior beliefs.

augmented with a neural policy network [46, 47]. Having a continuous state vector gives rise to additional challenges of exploding width at the belief nodes, making the tree too shallow to be useful and may cause collapse of belief representations resulting in overconfidence in the estimated policy. Further, the added observation noise would exacerbate these challenges.

5.1 2D search space discretized into 16×16 *grid cells*

Figure 3 shows the mean and s.e. of the full recovery rate versus total cost incurred over 10 trials with J agents looking for $k = 5$ targets in a 16×16 search space. We vary the team size as $J \in \{4, 8, 12\}$. Table 1 indicates the corresponding mean and s.e. of the total cost to correctly detect all targets. CAST simulates $m = 25000$ episodes with a lookahead horizon of 2 actions ($d_{\text{max}} = 2$). Each agent can choose from 341 region sensing actions over successive time steps.[4]

Table 1: Total cost (mean and s.e. over 10 trials) to achieve full recovery in a 16×16 grid with J agents, $k = 5$ targets. CAST outperforms all other baselines for different team sizes and different relative costs for travelling and sensing.

Algorithm	J	$c_s = 0$s	$c_s = 50$s
CAST	4	**655.9 (39.4)**	**6852.3 (314.1)**
SPATS		2988.8 (285.6)	12563.8 (1132.7)
RSI		797.4 (37.2)	6862.4 (252.8)
PS		1654.1 (64.4)	42753.3 (1742.6)
CAST	8	**827.0 (48.4)**	**9529.7 (350.6)**
SPATS		2482.3 (255.7)	10242.3 (1033.6)
RSI		1455.5 (59.8)	12815.5 (513.6)
PS		3414.9 (143.7)	88839.9 (3735.3)
CAST	12	**991.6 (39.6)**	**7647.6 (445.4)**
SPATS		2699.2 (240.1)	10764.2 (948.8)
RSI		2118.9 (71.0)	19023.9 (551.1)
PS		4827.0 (167.4)	125582.0 (4352.2)

We observe that CAST outperforms SPATS, RSI and PS by incurring a lower cost and a higher full recovery rate across different team sizes and cost scenarios. RSI is information greedy and outperforms SPATS and PS when there are fewer agents searching the region. Unfortunately, RSI is deterministic in its decision making and as a result all agents choose the same actions, which leads to an increasing cumulative cost with larger team sizes. On the other hand, the stochastic nature of TS based active search in SPATS is suited to the asynchronous and decentralized multi-agent setup and becomes competitive especially in the setting where sensing actions are more expensive than travelling ($c_s = 50s$) which aligns more with the objective of (cost-agnostic) active information gathering. Exhaustive coverage in PS is comparable only with a smaller team size in the case when travelling is expensive (J=4,c_s=0s) but outperforms SPATS in that setting, thereby showing the need for cost-awareness in active search. Unfortunately in cases that do not match their most favorable scenarios, all of these algorithms exhibit poor cost efficiency. In contrast, the cost-aware agents using CAST's posterior sampling based lookahead pareto-optimal planning and stochastic decision making are able to achieve cost efficiency across different cost scenarios with teams of varying sizes.

We also evaluate the robustness of CAST by comparing the total cost incurred to correctly identify all targets as the number of targets increases in the search space. Table 2 shows the mean and s.e. over 5 trials of the total cost incurred to reach full recovery with $J = 8$ agents and varying number of targets k in a 16×16 search space. We observe that CAST not only outperforms all others across multi-target and cost scenarios, additionally the total cost incurred is hardly affected by k since CAST enables each agent to be cost-aware through decentralized decision making independent of team size J and sparsity rate k. In contrast, SPATS being myopic in nature exhibits more randomness in the actions selected, whereas RSI approximates its mutual information objective assuming $k = 1$, thereby requiring more sensing actions to

[4] $|\mathcal{A}| = 341$ for a hierarchical region sensing action space in a 16×16 grid.

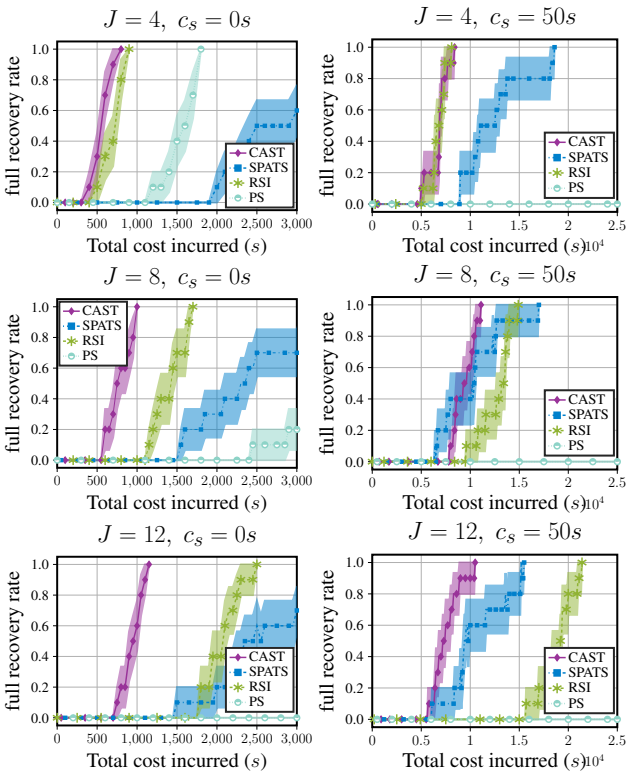


Figure 3: Full recovery rate versus total cost incurred in seconds in a 16×16 grid with J agents, $k = 5$ targets. Shaded regions indicate standard error over 10 trials. Compared to baselines, CAST achieves a full recovery rate of 1 at a lower total cost, both when traveling is costlier than sensing ($c_s = 0$s) and when sensing is more expensive ($c_s = 50$s).

Table 2: Total cost (mean and s.e. over 5 trials) to achieve full recovery in a 16×16 grid with $J = 8$ agents, k targets. For different number of targets, CAST incurs similar total cost for the same grid size and teamsize. CAST also incurs a lower cost compared to baselines.

Algorithm	k	$c_s = 0$s	$c_s = 50$s
CAST	4	**740.7 (26.5)**	**8130.4 (293.1)**
SPATS		3404.4 (432.9)	13574.4 (1792.2)
RSI		1262.8 (73.4)	10242.8 (685.8)
PS		2698.3 (438.6)	70208.3 (11404.6)
CAST	8	**735.3 (57.9)**	**9157.0 (476.7)**
SPATS		3217.2 (461.5)	13267.2 (1737.3)
RSI		1968.4 (72.2)	18398.4 (500.3)
PS		3339.9 (151.7)	86889.9 (3945.2)
CAST	16	**843.9 (29.9)**	**8880.8 (316.2)**
SPATS		3032.7 (54.3)	13212.7 (321.0)
RSI		2734.4 (58.9)	30524.4 (871.3)
PS		3559.1 (83.7)	92589.1 (2176.8)

recover all targets as k increases.

5.2 Comparison with myopic cost-aware variations of SPATS

As discussed earlier in Section 5, to the best of our knowledge, there are no existing cost-aware active search baselines to compare against CAST. We therefore modify the cost-agnostic myopic active search baseline SPATS [14] to incorporate cost-awareness in two different ways.

SPATS-scalarize. First, we consider a scalarized cost-aware version of the SPATS decision making objective:

$$\mathbf{x}_t^j = \underset{\mathbf{x}}{argmax} \lambda_r \mathbb{E}_{y|\mathbf{x},\tilde{\boldsymbol{\beta}}}[-\|\tilde{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}, y\})\|_2^2] - \lambda_d c_d(x_{t-1}^j, x) - \lambda_s c_s \quad (4)$$

Since the sensing cost c_s is a constant for all actions, it does not affect the action selected for different c_s. Instead, the agent will optimize only for the reward vs. travel cost, weighted by the reward coefficient λ_r and the travel cost coefficient λ_d. λ_s is the sensing cost coefficient, such that $\lambda_r + \lambda_d + \lambda_s = 1$.

SPATS-pareto. Second, we consider a pareto-optimization approach to augment the myopic decision making step of a SPATS agent. For each feasible action $\mathbf{x}$, the agent constructs a myopic reward-cost vector: $\begin{bmatrix} \mathbb{E}_{y|\mathbf{x},\tilde{\boldsymbol{\beta}}}[-\|\tilde{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}, y\})\|_2^2] \\ -(c_d(x_{t-1}^j, x) + c_s) \end{bmatrix}$. The pareto-front over such reward-cost vectors would exclude all actions with a lower reward *and* a higher cost. Then the agent selects the action from the pareto-front having the maximum one-step reward per unit cost:

$$\mathbf{x}_t^j = \underset{\mathbf{x}}{argmax} \frac{\mathbb{E}_{y|\mathbf{x},\tilde{\boldsymbol{\beta}}}[-\|\tilde{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}, y\})\|_2^2]}{c_d(x_{t-1}^j, x) + c_s}. \quad (5)$$

Unlike SPATS-scalarize, SPATS-pareto does not depend on λ_d.

Table 3: Total cost (mean and s.e. over 10 trials) to achieve full recovery in a 16×16 grid with J agents, $k = 5$ targets. CAST outperforms the myopic cost-aware modifications of SPATS across different team sizes and cost scenarios.

Algorithm	J	$c_s = 0$s	$c_s = 50$s
CAST	4	**655.9 (39.4)**	**6852.3 (314.1)**
SPATS-scalarize		673.5 (61.5)	13853.6 (845.4)
SPATS-pareto		693.7 (33.5)	23470.7 (1341.9)
CAST	8	**827.0 (48.4)**	**9529.7 (350.6)**
SPATS-scalarize		851.2 (53.4)	16296.8 (956.8)
SPATS-pareto		1018.0 (74.8)	26783.4 (1392.1)
CAST	12	**991.6 (39.6)**	**7647.6 (445.4)**
SPATS-scalarize		1001.2 (64.7)	18598.6 (1067.6)
SPATS-pareto		1122.4 (146.6)	32472.2 (1487.6)

In Table 3 and Table 4, we observe that CAST still outperforms these modified cost-aware myopic baselines, SPATS-scalarize and SPATS-pareto, across different number of targets, different team sizes and different cost scenarios. For SPATS-scalarize, when c_s=0s, grid search over λ_d indicates minimum cost incurred for λ_d=0.5, so we set λ_d=0.5, λ_r=0.5 in our experiments. λ_s does not affect the action selected, so we set λ_s=0. When c_s=50s, SPATS-scalarize with λ_d=0.5 selects the same actions as when c_s=0s, thus incurring a noticeably higher cost compared to cost-agnostic SPATS with λ_d=0,λ_s=0. SPATS-pareto selects the action from its pareto-front maximizing the one-step reward per unit cost (Equation (5)), which as we discussed in Section 4.2 would prefer actions with a higher incurred cost for the same reward. As a result, SPATS-scalarize outperforms SPATS-pareto in the same myopic decision making setting. In contrast, lookahead planning in CAST enables cost-aware action selection and incurs a lower cumulative cost than these baselines. CAST shows noticeable performance gain especially across different team sizes (J) with a higher sensing cost ($c_s = 50$s) or more number of targets (k) in the search space. These observations therefore imply the need for careful consideration of how cost-awareness is incorporated in multi-agent active search and validate our algorithm CAST in this setting.[5]

Table 4: Total cost (mean and s.e. over 5 trials) to achieve full recovery in a 16×16 grid with $J = 8$ agents, k targets. CAST outperforms the myopic cost-aware modifications of SPATS for different number of targets in the search space, with increasing cost-efficiency in comparison for higher k.

Algorithm	k	$c_s = 0$s	$c_s = 50$s
CAST	4	**740.7 (26.5)**	**8130.4 (293.1)**
SPATS-scalarize		777.5 (66.7)	15522.7 (1563.5)
SPATS-pareto		909.0 (78.5)	26157.2 (1805.2)
CAST	8	**735.3 (57.9)**	**9157.0 (476.7)**
SPATS-scalarize		884.8 (67.5)	19669.7 (944.2)
SPATS-pareto		1010.4 (55.9)	29126.9 (752.3)
CAST	16	**843.9 (29.9)**	**8880.8 (316.2)**
SPATS-scalarize		976.6 (44.6)	21328.7 (918.3)
SPATS-pareto		1049.6 (49.5)	29651.4 (870.2)

6 Conclusion

We have proposed CAST, an online cost-aware asynchronous multi-agent active search algorithm without a central planner. CAST is grounded on realistic assumptions of region sensing and observation sensor noise. CAST combines posterior sampling for decentralized multi-agent decision making, MCTS for multi-step lookahead planning and LCB based pareto-front construction for multi-objective optimization while overcoming the sample variability in estimating lookahead reward values. Similar to other tree search algorithms, CAST also experiences increasing computation time as the size of the search space or the tree depth increases. This can be mitigated by using a policy network as is common in game-tree based MCTS [46] or by implementing techniques for parallelizing MCTS, which is an active area of research [34, 54]. We leave such modifications as potential steps for future work.

References

[1] J-Y Audibert, R Munos, and C Szepesvari, 'Use of variance estimation in the multi-armed bandit problem', in *NIPS Workshop on On-line Trading of Exploration and Exploitation*, (2006).

[2] J Azimi, A Fern, X Z Fern, G Borradaile, and B Heeringa, 'Batch active learning via coordinated matching', in *ICML*, (2012).

[3] A Bai, F Wu, Z Zhang, and X Chen, 'Thompson sampling based monte-carlo planning in pomdps', in *ICAPS*, volume 24, (2014).

[4] D S Bernstein, R Givan, N Immerman, and S Zilberstein, 'The complexity of decentralized control of markov decision processes', *Mathematics of operations research*, **27**(4), (2002).

[5] Please additionally refer to the supplementary material at this link.

[5] G Best, O M Cliff, T Patten, R R Mettu, and R Fitch, 'Decentralised monte carlo tree search for active perception', in *Algorithmic Foundations of Robotics XII*, Springer, (2020).

[6] C B Browne et al., 'A survey of monte carlo tree search methods', *IEEE T-CIAIG*, (2012).

[7] W Chen and L Liu, 'Pareto monte carlo tree search for multi-objective informative planning.', in *RSS*, (2019).

[8] S Choudhury, N Gruver, and M J Kochenderfer, 'Adaptive informative path planning with multimodal sensing', in *ICAPS*.

[9] R Coulom, 'Computing "elo ratings" of move patterns in the game of go', *ICGA journal*, **30**(4), (2007).

[10] J Fischer and Ö S Tas, 'Information particle filter tree: An online algorithm for pomdps with belief-based rewards on continuous domains', in *ICML*, (2020).

[11] G Flaspohler, V Preston, A PM Michel, Y Girdhar, and N Roy, 'Information-guided robotic maximum seek-and-sample in partially observable continuous environments', *IEEE RAL*, (2019).

[12] R Garnett, Y Krishnamurthy, D Wang, J Schneider, and R Mann, 'Bayesian optimal active search on graphs', in *Ninth Workshop on Mining and Learning with Graphs*, (2011).

[13] R Garnett, Y Krishnamurthy, X Xiong, J Schneider, and R Mann, 'Bayesian optimal active search and surveying', in *ICML*, (2012).

[14] R Ghods, A Banerjee, and J Schneider, 'Decentralized multi-agent active search for sparse signals', in *UAI*, (2021).

[15] R Ghods, W J Durkin, and J Schneider, 'Multi-agent active search using realistic depth-aware noise model', in *IEEE ICRA*, (2021).

[16] A Gopalan and S Mannor, 'Thompson sampling for learning parameterized markov decision processes', in *COLT*, (2015).

[17] A Gopalan, S Mannor, and Y Mansour, 'Thompson sampling for complex online problems', in *ICML*, (2014).

[18] A Gupta, D Bessonov, and P Li, 'A decision-theoretic approach to detection-based target search with a uav', in *2017 IEEE/RSJ IROS*, (2017).

[19] C Igoe, R Ghods, and J Schneider, 'Multi-agent active search: A reinforcement learning approach', *IEEE RAL*, (2021).

[20] S Jiang, R Garnett, and B Moseley, 'Cost effective active search', in *NeurIPS*, (2019).

[21] S Jiang, G Malkomes, M Abbott, B Moseley, and R Garnett, 'Efficient nonmyopic batch active search', in *NeurIPS*, (2018).

[22] S Jiang, G Malkomes, G Converse, A Shofner, B Moseley, and R Garnett, 'Efficient nonmyopic active search', in *ICML*, (2017).

[23] L P Kaelbling, M L Littman, and A R Cassandra, 'Planning and acting in partially observable stochastic domains', *Artificial intelligence*, **101**(1-2), (1998).

[24] K Kandasamy, A Krishnamurthy, J Schneider, and B Póczos, 'Parallelised bayesian optimisation via thompson sampling', in *AISTATS*, (2018).

[25] D Kent and S Chernova, 'Human-centric active perception for autonomous observation', in *2020 IEEE ICRA*, (2020).

[26] L Kocsis and C Szepesvári, 'Bandit based monte-carlo planning', in *ECML*, (2006).

[27] M Lauri and F Oliehoek, 'Multi-agent active perception with prediction rewards', *NeurIPS*, **33**, (2020).

[28] M Lauri, J Pajarinen, and J Peters, 'Multi-agent active information gathering in discrete and continuous-state decentralized pomdps by policy graph improvement', *AAMAS*, **34**(42), (2020).

[29] S Lazebnik, C Schmid, and J Ponce, 'Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories', in *CVPR*, volume 2, (2006).

[30] J Lee, G-H Kim, P Poupart, and K-E Kim, 'Monte-carlo tree search for constrained pomdps', in *NeurIPS*, (2018).

[31] J Leike, T Lattimore, L Orseau, and M Hutter, 'Thompson sampling is asymptotically optimal in general environments', *arXiv:1602.07905*, (2016).

[32] Z W Lim, D Hsu, and W S Lee, 'Adaptive informative path planning in metric spaces', *IJRR*, (2016).

[33] J Linchant, J Lisein, J Semeki, P Lejeune, and C Vermeulen, 'Are unmanned aircraft systems (uas s) the future of wildlife monitoring? a review of accomplishments and challenges', *Mammal Review*, **45**(4), (2015).

[34] A Liu, Y Liang, J Liu, G Van den Broeck, and J Chen, 'On effective parallelization of monte carlo tree search', *arXiv:2006.08785*, (2020).

[35] Y Ma, R Garnett, and J Schneider, 'Active search for sparse signals with region sensing', in *AAAI*, (2017).

[36] R Marchant, F Ramos, S Sanner, et al., 'Sequential bayesian optimisation for spatial-temporal monitoring.', in *UAI*, (2014).

[37] A A Meera, M Popović, A Millane, and R Siegwart, 'Obstacle-aware adaptive informative path planning for uav-based target search', in *IEEE ICRA*, (2019).

[38] F A Oliehoek, C Amato, et al., *A concise introduction to decentralized POMDPs*, volume 1, Springer, 2016.

[39] R Pěnička, J Faigl, M Saska, and P Váňa, 'Data collection planning with non-zero sensing distance for a budget and curvature constrained unmanned aerial vehicle', *Autonomous Robots*, **43**(8), (2019).

[40] M Popović, T Vidal-Calleja, G Hitz, I Sa, R Siegwart, and J Nieto, 'Multiresolution mapping and informative path planning for uav-based terrain monitoring', in *2017 IEEE/RSJ IROS*, (2017).

[41] P Rajan, W Han, R Sznitman, P Frazier, and B Jedynak, 'Bayesian multiple target localization', *Journal of Machine Learning Research*, **37**, (2015).

[42] C Robin and S Lacroix, 'Multi-robot target detection and tracking: taxonomy and survey', *Autonomous Robots*, **40**(4), (2016).

[43] D M Roijers, P Vamplew, S Whiteson, and R Dazeley, 'A survey of multi-objective sequential decision-making', *Journal of Artificial Intelligence Research*, (2013).

[44] D Russo, B Van Roy, A Kazerouni, I Osband, and Z Wen, 'A tutorial on thompson sampling', *arXiv:1707.02038*, (2017).

[45] D Shah, Q Xie, and Z Xu, 'Non-asymptotic analysis of monte carlo tree search', in *2020 SIGMETRICS*.

[46] D Silver et al., 'Mastering the game of go with deep neural networks and tree search', *nature*, **529**(7587), (2016).

[47] D Silver et al., 'Mastering the game of go without human knowledge', *nature*, **550**(7676), (2017).

[48] D Silver and J Veness, 'Monte-carlo planning in large pomdps', in *NIPS*, (2010).

[49] A Singh, A Krause, and W J Kaiser, 'Nonmyopic adaptive informative path planning for multiple robots', in *IJCAI*, (2009).

[50] F Sukkar, G Best, C Yoo, and R Fitch, 'Multi-robot region-of-interest reconstruction with dec-mcts', in *IEEE ICRA*, (2019).

[51] W R Thompson, 'On the likelihood that one unknown probability exceeds another in view of the evidence of two samples', *Biometrika*, **25**(3/4), (1933).

[52] W Wang and M Sebag, 'Multi-objective monte-carlo tree search', in *ACML*, (2012).

[53] Z Yan, N Jouandeau, and A A Cherif, 'A survey and analysis of multi-robot coordination', *IJARS*, **10**(12), (2013).

[54] X Yang, T K Aasawat, and K Yoshizoe, 'Practical massively parallel monte-carlo tree search applied to molecular design', *arXiv:2006.10504*, (2021).

6.1 Belief model for CAST

In what follows, we will describe the belief model used in CAST as discussed in Section 4.2.

Notations: Non-bold characters represent scalars. Lowercase and uppercase boldface letters represent column vectors and matrices respectively. $\mathbf{A}^T$ is the transpose for a matrix $\mathbf{A}$. $\mathbf{I}_n$ denotes an $n \times n$ identity matrix. The ith entry of a vector $\mathbf{a}$ is $[\mathbf{a}]_i$ and the (i,j)th entry of a matrix $\mathbf{A}$ is $[\mathbf{A}]_{ij}$. diag($\mathbf{a}$) is a square matrix with $\mathbf{a}$ on the main diagonal. $q * A$ indicates multiplication of scalar q with every element of $\mathbf{A}$. $\mathbf{1}_{n\times 1}$ indicates a $n \times 1$ dimensional vector of ones.

Assuming that the targets are sparsely distributed in the environment, an agent's belief over search vector $\boldsymbol{\beta} \in \mathbb{R}^n$ is modeled by a sparse prior b_0:

$$b_0 = \mathrm{P}(\boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{6}$$

where $\boldsymbol{\mu}_0 = \frac{1}{n} * \mathbf{1}_{n\times 1}$ and $\boldsymbol{\Sigma}_0 = \mathrm{diag}(\boldsymbol{\tau})$, with hyperparameter $\boldsymbol{\tau} \in \mathbb{R}^n$. Given the measurement set $\mathbf{D}_t^j = \{\mathbf{x}_i^j, y_i^j\}_{i=1}^{t-1}$ we define $\mathbf{X}_t^j$ as the matrix $[{\mathbf{x}_1^j}^{\mathrm{T}} \ldots {\mathbf{x}_{t-1}^j}^{\mathrm{T}}]^{\mathrm{T}}$ and $\mathbf{y}_t^j$ as the column vector $[y_1^j \ldots y_{t-1}^j]^{\mathrm{T}}$. The likelihood function is $\mathrm{P}(\mathbf{y}_t^j|\mathbf{X}_t^j, \boldsymbol{\beta}) = \mathcal{N}(\mathbf{X}_t^j\boldsymbol{\beta}, \sigma^2 * \mathbf{I}_{t-1})$ from Equation (1). Therefore, the posterior belief over $\boldsymbol{\beta}$ is

$$\mathrm{P}(\boldsymbol{\beta}|\mathbf{D}_t^j, \boldsymbol{\tau}) = \mathcal{N}(\boldsymbol{\mu}_\beta^j(\boldsymbol{\tau}), \boldsymbol{\Sigma}_\beta^j(\boldsymbol{\tau})) \tag{7}$$

where

$$\boldsymbol{\Sigma}_\beta^j(\boldsymbol{\tau}) = ((\boldsymbol{\Sigma}_0)^{-1} + \frac{1}{\sigma^2}{\mathbf{X}_t^j}^{\mathrm{T}}\mathbf{X}_t^j)^{-1} \tag{8}$$

$$\boldsymbol{\mu}_\beta^j(\boldsymbol{\tau}) = \boldsymbol{\mu}_0 + \frac{1}{\sigma^2}\boldsymbol{\Sigma}_\beta^j(\boldsymbol{\tau}){\mathbf{X}_t^j}^{\mathrm{T}}(\mathbf{y}_t^j - \mathbf{X}_t^j\boldsymbol{\mu}_0). \tag{9}$$

The hyperparameter $\boldsymbol{\tau}$ can be estimated using Expectation Maximization over $p = 1, 2, \ldots, n_{EM}$ iterations as follows.

$$\textbf{E-step: } \boldsymbol{\mu}_{\boldsymbol{\beta}}^{(p)} = \boldsymbol{\mu}_{\boldsymbol{\beta}}(\boldsymbol{\tau}^{(p-1)}) \tag{10}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{\beta}}^{(p)} = \boldsymbol{\Sigma}_{\boldsymbol{\beta}}(\boldsymbol{\tau}^{(p-1)}) \tag{11}$$

$$\textbf{M-step: } \forall j = 1, \ldots, n$$

$$[\boldsymbol{\tau}^{(p)}]_j = [\boldsymbol{\Sigma}_{\boldsymbol{\beta}}^{(p)}]_{jj} + ([\boldsymbol{\mu}_{\boldsymbol{\beta}}^{(p)}]_j - \frac{1}{n})^2 \tag{12}$$

At time step t, after the agent executes action $\mathbf{x}_t^j$ and receives an observation y_t^j, we update its estimate $\hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}_t^j, y_t^j\})$ using the MAP estimator given by:

$$\hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}_t^j, y_t^j\})$$

$$= \big(\sigma^2 * {\boldsymbol{\Sigma}_0}^{-1} + \begin{bmatrix}{\mathbf{X}_t^j}^{\mathrm{T}} & \mathbf{x}_t^j\end{bmatrix}\begin{bmatrix}\mathbf{X}_t^j \\ {\mathbf{x}_t^j}^{\mathrm{T}}\end{bmatrix}\big)^{-1}\begin{bmatrix}{\mathbf{X}_t^j}^{\mathrm{T}} & \mathbf{x}_t^j\end{bmatrix}\begin{bmatrix}\mathbf{y}_t^j \\ y_t^j\end{bmatrix} \tag{13}$$

where $\boldsymbol{\Sigma}_0 = \mathrm{diag}(\boldsymbol{\tau}^{(n_{EM})})$.

6.2 Additional implementation details

Table 5 summarizes the various symbols and notations used in describing CAST in Section 4.3.

Now let us briefly describe the algorithms RSI, SPATS and PS which we compare against CAST.

Review of RSI (Region Sensing Index): RSI [35] is a single agent myopic active search algorithm wherein at any time step t, the agent selects the region sensing action $\mathbf{x}_t$ which would maximize the mutual information between the resulting observation y_t and the search vector $\boldsymbol{\beta}$ i.e.

$$\mathbf{x}_t = \underset{\mathbf{x}}{argmax}\, I(\boldsymbol{\beta}; y|\mathbf{x}, \mathbf{D}_t^1). \tag{14}$$

Table 5: Symbols and notations in Section 4.3

Notation	Definition
$b_t(\boldsymbol{\beta})$	Posterior belief over $\boldsymbol{\beta}$ at time step t
$\mathbf{D}_t^j$	Set of past actions and observations available to the agent j at time t
λ^-	One-step lookahead reward for a CAST agent
d_{max}	Maximum lookahead depth of the search tree. Successive levels differ by a depth of 0.5.
n	Dimension of $\boldsymbol{\beta}$
$n(h)$	Number of times belief node h is visited in search tree $\mathcal{T}_t$
$n(h,a)$	Number of times action node (h,a) is visited in search tree $\mathcal{T}_t$
x	Position of an agent in the search space after executing sensing action $\mathbf{x}$
a	Position of an agent in the search space after executing sensing action $\mathbf{a}$
$c_d(x,a)$	Time cost of agent travelling from position x to position a
c_s	Time cost of agent executing a sensing action $\mathbf{x}$
γ	Discount factor for computing multi-step lookahead reward over a finite horizon
α_s	Progressive widening parameter
m	Total number episodes of tree building in every decision making time step
J	Total number of agents performing active search

The mutual information I is computed using the posterior distribution $\mathrm{P}(\boldsymbol{\beta}|\mathbf{D}_t^1) = b_0 \prod_{t'=1}^{t-1} \mathrm{P}(y_{t'}|\mathbf{x}_{t'}, \boldsymbol{\beta})$ with a k-sparse uniform prior b_0 and the same likelihood distribution as in Section 6.1. Unfortunately, computing I is cumbersome for sparsity $k > 1$ and RSI addresses this by iteratively identifying the most likely target locations from its belief assuming $k = 1$.

Review of SPATS (Sparse Parallel Asynchronous Thompson Sampling): SPATS [14] is a multi-agent decentralized and asynchronous active search algorithm which uses Thompson sampling (TS) to determine the next sensing action, i.e.

$$\mathbf{x}_t^j = \underset{\mathbf{x}}{argmax}\, \mathbb{E}_{y|\mathbf{x},\tilde{\boldsymbol{\beta}}}[-\|\tilde{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}(\mathbf{D}_t^j \cup \{\mathbf{x}, y\})\|_2^2] \tag{15}$$

where $\tilde{\boldsymbol{\beta}}$ is a sample drawn from the posterior $\mathrm{P}(\boldsymbol{\beta}|\mathbf{D}_t^j)$ which is a normal distribution assuming a block sparse prior and the same likelihood distribution as in Section 6.1. Unfortunately, SPATS is myopic in nature and relies on a carefully tuned block length reduction schedule in its posterior belief update to overcome the limitations of purely TS based exploration strategy in active search as discussed in Section 4.1.

Sequential Point Sensing (PS): We design this as an exhaustive coverage baseline where an agent starts from one corner on the grid and traverses every grid cell sequentially, executing point sensing actions to cover the entire search space. In the multi-agent case, every agent follows the same trajectory, so the incurred cost is expected to increase with larger team sizes due to repetitive sensing actions.

6.3 Additional empirical results

We now provide additional empirical results for cost-aware active search in a 2-dimensional (2D) search space discretized into grids of sizes 8×8 and 8×16 respectively with square cells of width 10m.

2D search space discretized into 8×8 grid cells: Table 6 compares the total cost incurred in fully recovering $\boldsymbol{\beta}$ at 2 different sparsity rates $k \in \{1, 3\}$ with $J = 4$ agents in an 8×8 search space. For CAST, we varied the tree depth $d_{\text{max}} \in \{2, 3, 4\}$ and the number of simulation episodes $m \in \{5, 7.5, 10, 20\} \times 10^4$. Table 6 reports the results corresponding to the best performing d_{max} and m in each

case. CAST-1 indicates the performance with one-step lookahead i.e. $d_{\text{max}} = 1$ over $m = 20 \times 10^4$ episodes to emphasize the importance of multi-step lookahead over a finite horizon in our cost-aware algorithm. Figure 4 plots the corresponding full recovery rate across trials as a function of the total cost incurred. Each agent can choose from 85 region sensing actions over successive time steps. We observe that CAST outperforms SPATS, RSI and PS by incurring a lower total cost. RSI being information-optimal and modeling the assumption $k = 1$ in its hypothesis space is at an advantage in the single target setting. The stochastic nature of TS based active search in SPATS favours it in the multi-target setting when sensing is more expensive than travelling. Exhaustive coverage in PS is comparable only in a single target setting if travelling is expensive. But in cases that do not match their most favorable scenarios, all of them exhibit poor cost efficiency. In contrast, CAST's ability to perform adaptive lookahead plannning together with posterior sampling helps it achieve cost efficiency across single, multi-target and different cost scenarios.

Table 6: Total cost (s) (mean and s.e. over 10 trials) to achieve full recovery in an 8×8 grid with $J = 4$ agents.

Algorithm	k	$c_s = 0$s	$c_s = 50$s
CAST	1	**93.10 (12.35)**	**1268.98 (255.76)**
CAST-1		168.83 (12.40)	2125.62 (74.97)
SPATS		238.84 (39.42)	1570.91 (225.71)
RSI		124.24 (13.43)	1321.21 (124.18)
PS		186.61 (51.27)	4887.01 (1330.73)
CAST	3	**147.65 (16.11)**	**2392.21 (142.16)**
CAST-1		186.61 (8.13)	2678.65 (176.75)
SPATS		343.25 (44.69)	2454.76 (221.91)
RSI		233.12 (15.25)	2851.48 (182.40)
PS		368.93 (29.92)	9774.93 (843.40)

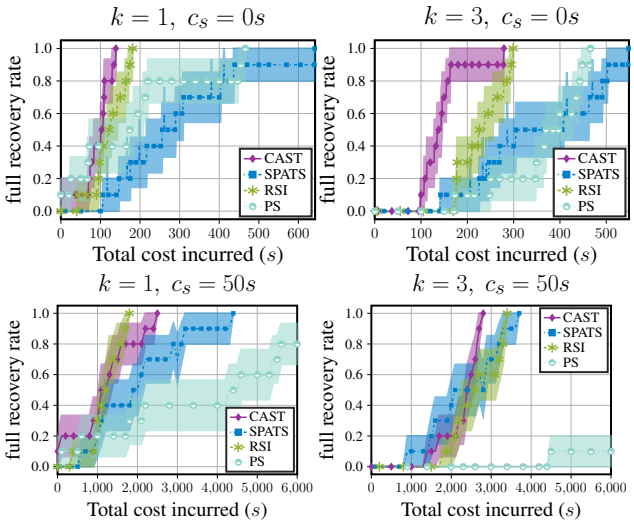


Figure 4: Full recovery rate versus total cost incurred in seconds in a 8×8 grid with $J = 4$ agents and k targets. Shaded regions indicate s.e.

2D search space discretized into 8×16 **grid cells:** Table 7 compares the total cost incurred in fully recovering $\boldsymbol{\beta}$ at 2 different sparsity rates $k \in \{1, 5\}$ with $J = 3$ agents in an 8×16 search space. Figure 5 plots the corresponding full recovery rate across trials as a function of the total cost incurred. We fixed the tree depth in CAST at $d_{\text{max}} = 2$ and the number of simulation episodes $m = 10^5$. Each agent can choose from 170 region sensing actions over successive time steps. In almost all the settings, CAST outperforms SPATS, RSI and PS by choosing cost-aware actions that reduce its total incurred cost. When sensing is so expensive that traveling cost is negligible ($c_s = 50s$), especially in the single target setting, we observe that the information seeking algorithm RSI is at an advantage compared to the shallow lookahead in CAST. But when travelling is expensive, even at a lookahead horizon of 2 actions, CAST enables better cost efficiency than RSI. Moreover, exhaustive coverage in PS also incurs lower cost compared to SPATS when travelling is expensive, further indicating the need for cost awareness in active search.

Table 7: Total cost in seconds (mean and s.e. over 10 trials) to achieve full recovery in an 8×16 grid with $J = 3$ agents.

Algorithm	k	$c_s = 0$s	$c_s = 50$s
CAST	1	**110.09 (30.48)**	2616.90 (213.31)
SPATS		551.74 (120.52)	1931.28 (211.27)
RSI		135.56 (9.96)	**1166.69 (84.72)**
PS		183.80 (83.04)	7383.20 (2146.24)
CAST	5	**258.04 (16.55)**	**3705.41 (211.81)**
SPATS		1010.94 (89.86)	5250.94 (442.07)
RSI		398.01 (8.99)	4243.57 (95.50)
PS		631.80 (34.71)	16472.80 (894.30)

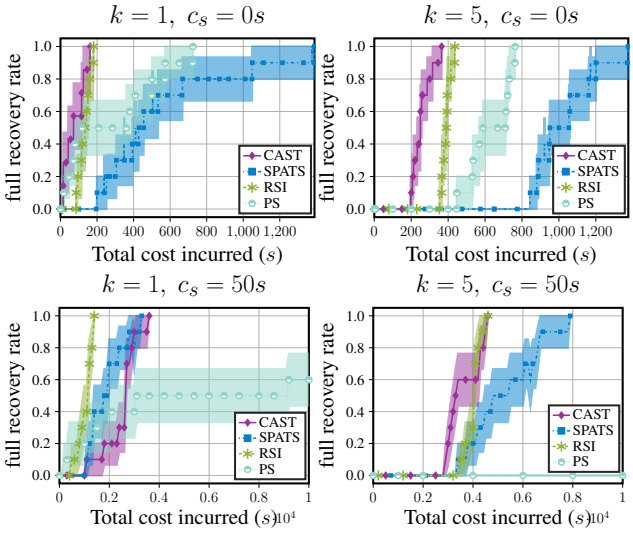


Figure 5: Full recovery rate versus total cost incurred in seconds in a 8×16 grid with $J = 3$. Shaded regions indicate s.e.

2D search space discretized into 16×16 **grid cells:** Figure 6 shows the full recovery rate against the total cost incurred by $J = 8$ agents in identifying $k \in \{4, 5, 8, 16\}$ targets in a 16×16 search space. As described in Section 5 and Table 2, the cost efficiency of CAST is robust to the varying sparsity rate as compared to RSI and SPATS. The plot for PS is excluded for the subfigure on the right for better visualization.

Additional visualizations: In the supplementary folder, we are also including animations from a single trial of each algorithm (for the same seed i.e. same target distribution) in the 8×8 search space with $J = 4$ agents and $k = 3$ targets. This helps to understand the difference in behavior of CAST and the other baselines. We observe that the sensing actions executed by CAST agents are not only information gathering but also cost-aware and adapt to the relative cost of

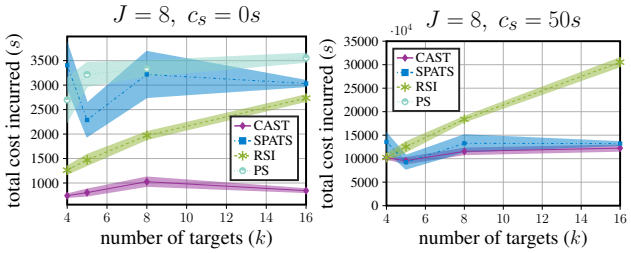


Figure 6: Total cost incurred versus number of targets $k \in \{4, 5, 8, 16\}$ in a 16×16 grid with $J = 8$ agents.

travelling versus sensing, thereby outperforming both RSI and SPATS in terms of the total cost incurred.

6.4 Scaling up CAST

The computational complexity of CAST (Algorithm 1) increases with the maximum depth of the search tree (d_{max}) and the size n of the search vector $\boldsymbol{\beta} \in \mathbb{R}^n$. For a larger n, the size of the action space being $O(n)$ (considering spatial hierarchical pyramid sensing actions) implies that the tree policy as well as the new action node addition policy has to evaluate a larger set of feasible actions at every belief node encountered during the selection and expansion phase and this quickly becomes computationally expensive with increasing n. Moreover, increasing d_{max} further exacerbates the time complexity since it expands the space of lookahead action sequences and as a result, more episodes are required for effective exploration within the search tree. Additionally, as the tree width increases with completion of more episodes, it also leads to an increase in the pareto front computation and update time at each tree node. In what follows, we describe two heuristic strategies that we implemented to scale CAST to a 16×16 search space (results shown in Section 5).

*6.4.1 **Sampling from actions***

In order to select the new action node to be added to the search tree (Line 20,Algorithm 1), we iterate over all feasible next actions at a belief node h (denoting the set by $\mathcal{A}_h$) and for each such action $\mathbf{a} \in \mathcal{A}_h$, we compute the change in entropy of the belief distribution per unit immediate cost incurred if $\mathbf{a}$ were executed. The action $\mathbf{a} \in \mathcal{A}_h$ that maximizes this quantity is selected and the tree expands to include the new action node (h, a). This strategy leads to more directed exploration within the search space than simple random sampling from $\mathcal{A}_h$. Unfortunately, it becomes computationally expensive as the size of the action space increases. Therefore, we propose sampling a subset $\mathcal{A}_h^s$ of size s from the feasible action pool ($\mathcal{A}_h^s \subset \mathcal{A}_h$) and select the action $\mathbf{a}' \in \mathcal{A}_h^s$ with the maximum change in entropy of the belief distribution per unit immediate incurred cost. This not only reduces the computational cost of CAST, it also introduces additional stochasticity in the search tree building phase in the multi-agent setting.

*6.4.2 **Pruning the tree***

In contrast to the progressive widening strategy followed while adding children action nodes at the interior belief nodes in the search tree $\mathcal{T}_t$, we observed that CAST performs better when throughout the m episodes, the root node has as its children *all* the feasible action nodes at time step t. Although it helps our CAST-UCT tree policy to balance exploration-exploitation with the knowledge of the entire feasible action set, it would not be scalable in terms of the number of episodes needed as the size of the action space increases. Therefore, we propose a pruning technique using which we can prune the action nodes at the root level of $\mathcal{T}_t$ after a pre-determined number of simulation episodes are completed. Note that the particular episodes when we prune the tree is a tunable hyperparameter that also determines the cost-aware performance. In order to achieve this, in the backpropagation phase of each episode, we additionally maintain the upper confidence bound (UCB) based reward-cost pareto front using the backed up values. In any episode m', the 1-step lookahead reward λ^- is computed at a belief node h. We maintain the UCB (r_h^{UCB}) of these rewards over $n(h)$ episodes using the Student's t-distribution to compute a 95% confidence interval. c_h is the immediate cost of executing the action that would result in transitioning to belief node h. At the leaf node h_ℓ for episode m', we define the UCB based immediate reward-cost vector $\mathbf{g}_{h_\ell}^{'} = \begin{bmatrix} r_{h_\ell}^{UCB} & -c_{h_\ell} \end{bmatrix}^{\mathrm{T}}$. To distinguish it from the LCB based vector $\mathbf{g}_h$ defined in Section 4.3, we will refer to $\mathbf{g}_h^{'}$ as our UCB pruning vector. During backpropagation, at each tree node visited in m', we update the UCB based lookahead reward-cost pareto-front in the same way as described for the LCB in Section 4.3. Assuming that $\mathcal{T}_t$ is to be pruned at the m''-th episode, we remove all those child actions at the root whose UCB based pareto-front is dominated by the LCB based pareto-front over all actions at the root. This pruned $\mathcal{T}_t$ is then used over subsequent simulation episodes at time t.