Improving the Efficiency of Clearing with Multi-Agent Teams

Geoffrey Hollinger (corresponding author), Sanjiv Singh
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
gholling@ri.cmu.edu, ssingh@ri.cmu.edu

Athanasios Kehagias
Division of Mathematics
Department of Mathematics, Physics, and Computer Sciences
Aristotle University of Thessaloniki
Thessaloniki GR54124, Greece
kehagiat@gen.auth.gr

Abstract

We present an anytime algorithm for coordinating multiple autonomous searchers to find a potentially adversarial target on a graphical representation of a physical environment. This problem is closely related to the mathematical problem of searching for an adversary on a graph. Prior methods in the literature treat multi-agent search as either a worst-case problem (i.e., clear an environment of an adversarial evader with potentially infinite speed), or an average-case problem (i.e., minimize average capture time given a model of the target's motion). Both of these problems have been shown to be NP-hard, and optimal solutions typically scale exponentially in the number of searchers. We propose treating search as a resource allocation problem, which leads to a scalable anytime algorithm for generating schedules that clear the environment of a worst-case adversarial target *and* have good average-case performance considering a non-adversarial motion model. Our algorithm yields theoretically bounded average-case performance and allows for online and decentralized operation, making it applicable to real-world search tasks. We validate our proposed algorithm through a large number of experiments in simulation and with a team of robot and human searchers in an office building.

1 Introduction

Imagine you are the leader of a team of agents (humans, robots, and/or virtual agents), and you enter a building looking for a person, moving object, or contaminant. You wish either to locate a target in the environment or authoritatively say that no target exists. Such a scenario may occur in urban search and rescue (Kumar et al., 2004), military operations, network decontamination (Barrière et al., 2002), or even aged care (Roy et al., 2003). In some special cases, you may have a perfect model of how the target is moving; however, in most cases you will only have an approximate model or even no model at all. To complicate the situation further, the target may be adversarial and actively avoiding being found.

Known algorithms would force you, the leader, to make a choice in this situation. Do you make the worst-case assumption and choose to treat the target as adversarial? This would allow you to utilize graph search algorithms to guarantee finding the target (if one exists), but it would not allow you to take advantage of any model of the target's motion. As a result your search might take a very long time. Or do you decide to trust your motion model of the target and assume that the target is non-adversarial? This assumption would allow the use of efficient (average-case) search methods from the optimization literature, but it would eliminate any guarantees if the model is inaccurate. In this case, your target may avoid you entirely. It is necessary to make one of these choices because no existing method provides fast search times and also guarantees finding a target if the model is wrong.

Figure 1: Volunteer firefighter searching with a Pioneer mobile robot. The robot and humans execute a combined schedule that both clears the environment of an adversarial target and optimizes a non-adversarial target motion model. Our decentralized algorithm allows for robot and humans flexibly to fill the different search roles. The agents share their paths and an estimation of the target's position.

In this paper, we bridge the gap between worst-case (or guaranteed) search and average-case (or efficient) search. We propose a novel algorithm that augments a guaranteed clearing schedule with an efficient component based on a non-adversarial target motion model. We extend a guaranteed search spanning tree traversal algorithm to optimize clearing time, and we augment it with a decentralized finite-horizon planning method in which agents implicitly coordinate by sharing plans. We show that the average-case performance of the combined algorithm is bounded, and we demonstrate how our algorithm can be used in an anytime fashion by providing additional search schedules with increasing runtime. This produces a family of clearing schedules that can easily be selected before the search or as new information becomes available during the search. We validate our approach using extensive simulated experiments as well as on a human-robot search team (shown in Figure 1). The contribution of this paper is the first algorithm that provides guaranteed solutions to the clearing problem and (given additional knowledge of the target's behavior) improves average performance.

The remainder of this paper is organized as follows. We first discuss related work in both worst-case search and average-case search highlighting the lack of a combined treatment (Section 2). We then define both the worst-case and average-case search problems and show the formal connection

between the two (Section 3). This leads us to the presentation of our search algorithm, including a description of the finite-horizon and spanning tree traversal components, as well as theoretical analysis of performance bounds (Sections 4 and 5). We then test our algorithm through simulated trials and through experiments on a heterogeneous human/robot search team (Section 6). Finally, we conclude and discuss avenues for future work (Section 7).

2 Related Work

As mentioned above, literature in autonomous search can be partitioned into average-case and worst-case problems. The initial formulation of the worst-case graph search problem is due to Parsons (1976). He described a scenario where a team of agents searches for an omniscient, adversarial evader with unbounded speed in a cave-like topology represented by a graph. In this formulation, the edges of the graph represent passages in the cave, and the nodes represent intersections. The evader hides in the edges (passages) of the graph, and it can only move through nodes that are not guarded by searchers. This problem was later referred to as *edge search* since the evader hides in the edges of the graph. Parsons defined the edge search number of a graph as the number of searchers needed to guarantee the capture of an evader on that graph. Megiddo et al. (1988) later showed that finding the edge search number is NP-hard for arbitrary graphs. Fomin and Thilikos (2008) recently provided a comprehensive survey of recent results in guaranteed graph searching.

Several variations of the edge search problem appear in the literature that place restrictions on the form of the cleared set (i.e., the edges of the graph that have been cleared of a potential target). Connected edge search requires that the cleared set be a connected subgraph at all times during the search. Barrière et al. (2002) argued that connected edge search is important for network decontamination problems because decontaminating agents should not traverse dangerous contaminated parts of the graph. They also formulated a linear-time algorithm that generates search schedules with the minimal number of searchers on trees. Borie et al. (2009) more recently explored complexity results and algorithms for various edge search problems on trees and several other special types of graphs.

Unfortunately, edge search does not apply directly to many robotics problems. The possible paths of an evader in many indoor and outdoor environments in the physical world cannot be accurately represented as the edges in a graph.[1] For this reason, robotics researchers have studied alternative formulations of the guaranteed search problem. Guibas et al. (1999) formulated the search problem in polygonal environments and presented a complete algorithm for clearing with a single searcher. However, their algorithm shows poor scalability to large teams and complex environments. Gerkey et al. (2005) demonstrated how a stochastic optimization algorithm (PARISH) can be used to coordinate multiple robotic searchers in small indoor environments. Though it is more scalable than complete algorithms, PARISH still requires considerable computation and communication between searchers, which makes it difficult to apply to complex environments.

Kolling and Carpin (2010) showed how an extension of edge search algorithms can be applied to a weighted graph formulation of the multi-robot surveillance problem. Their algorithm operates on an alternative formulation of the clearing problem, which requires several searchers to clear areas in the environment. They consider a probabilistic variant with imperfect capture sensors (Kolling

[1] One exception would be a road network, which could be modeled as an edge search problem. The application of our combined algorithm to these types of environments is left for future work.

and Carpin, 2009), but they do not provide any mechanism to utilize information about the target's motion beyond a worst-case assumption.

In our prior work, we examined the *node search* problem in which the evader hides in the nodes of a graph, and we showed how search problems in the physical world can be represented as node search.[2] We proposed the Guaranteed Search with Spanning Trees (GSST) anytime algorithm that finds connected node search clearing schedules with few searchers (Hollinger et al., 2008, 2010). GSST is linearly scalable in the number of nodes in the graph, which makes it applicable to large teams and complex environments. Note that the algorithms described above (including our own prior work) do not optimize the time to clear the environment, and they do not take into account a model of the target's movement beyond a worst-case model.

A different, but closely related, search problem arises if we relax the need to deal with an adversarial target. If the target's motion model is non-adversarial and approximately known to the searchers, the searchers can optimize the average-case performance of their search schedule given this motion model. Assuming a Markovian motion model (i.e., the target's next position is dependent only on the target's current position), the average-case search problem can be expressed as a Partially Observable Markov Decision Process (POMDP). It is possible to compute near-optimal solutions to fairly large POMDPs (Smith, 2007), particularly those where parts of the state are fully observed (e.g., the searchers' locations) (Ong et al., 2009). However, the size of multi-robot search problems scales exponentially in the number of searchers, which puts search problems with even moderately sized teams outside the scope of general POMDP solvers.

In response to the intractability of optimal solutions, researchers have proposed several approximation algorithms for average-case search. Sarmiento et al. (2004) examined the case of a stationary target and presented a scalable heuristic. Singh et al. (2007) discussed the related problem of multi-robot informative path planning and showed a scalable constant factor approximation algorithm in that domain. In our prior work, we extended these approximation guarantees to average-case search with our Finite-Horizon Path Enumeration with Sequential Allocation (FHPE+SA) algorithm (Hollinger et al., 2009b). Our algorithm provides near-optimal performance in our test domains, but it does not consider the possibility that the model is inaccurate. Thus, the search may last for infinite time if the target is acting in a way that is not properly modeled. For instance, if a target is modeled as moving but is actually stationary, the searchers may never examine parts of the environment because they believe the target would have moved out of them.

The search problem has also been studied in conjunction with the problem of exploration and mapping. Calisi et al. (2007) provided a solution to the single robot exploration and search problem using petri-nets. Their work provides a principled architecture for robotic mission planning during urban search and rescue operations. However, they formulate their algorithm for the case of a single searcher, and it is difficult to see how it could extend to the multi-searcher case. In addition, they do not consider guaranteed strategies for clearing the environment of a worst-case target.

It is important to note that related work in static sensor networks combines worst-case and average-case guarantees. Krause et al. (2008) presented the SATURATE algorithm, which guides the placement of stationary sensors against an adversarial agent. SATURATE's runtime scales

[2]Note that several alternative versions of "node search" appear in the literature (see Alspach (2006) for a survey). In one formulation, the evader resides in the edges of the graph (hence, despite the name, this is really an edge search problem), and these edges are cleared by trapping (i.e., two searchers occupy the adjacent nodes). In another, the pursuers have knowledge of the evader's position while attempting to capture the evader by moving onto the same node.

linearly with the number of actions available to the adversary. In search problems, the number of actions (paths) available to the target scales exponentially with the size of the graph. Thus, SATURATE is infeasible for all but the smallest search problems.

This paper draws on some algorithms from our prior work, but the main contribution is independent of these tools. The FHPE+SA algorithm (Hollinger et al., 2009b) is utilized to determine the schedules for the average-case searchers in the current paper's combined algorithm. The proposed G-GSST algorithm in the current paper is an extension of the GSST algorithm (Hollinger et al., 2010). Unlike GSST, G-GSST allows for the optimization of clearing time and the implicit use of guards. The current paper is the first to consider combining the worst-case and average-case search problem, which includes analysis of both hardness and connection with graph search. Finally, the combined algorithm is in itself a novel resource allocation algorithm for solving the worst-case/average-case search problem, which is independent of the use of FHPE+SA and G-GSST as its components. A preliminary treatment of the results in the current paper appear in an earlier conference paper (Hollinger et al., 2009a).

To the best of our knowledge, no search algorithm exists that can clear an environment of a worst-case target and improve average-case search performance based on additional information (e.g., a target motion model or online sensor data). Our algorithm fills this gap.

3 Problem Setup

In this section, we define the search problem with respect to both a worst-case adversarial target and a non-adversarial target. We also show the formal connection between worst-case search and the guaranteed search problem found in the literature. Assume we are given K searchers and a graph $G = (N, E)$ with $|N|$ nodes and $|E|$ edges. The nodes in the graph represent possible locations in the environment, and the edges represent connections between them (see Figure 3 for examples in indoor environments). At all times $t = 1, 2, \ldots, T$, the searchers and the target exist on the nodes of this graph and can move through an edge to arrive at another node at time $t + 1$. Given a graph of possible locations and times, we can generate a time-evolving, time-unfolded graph $G' = (N', E')$, which gives a time-indexed representation of the nodes in the environment:

Definition 1 *The time-augmented search graph G' is a* directed graph *and is obtained from G as follows: if u is a node of G then (u, t) is a node of G', where $t = 1, 2, ..., T$ is the* time stamp*; if uv is an edge of G, then $(u, t)\,(v, t + 1)$ and $(v, t)\,(u, t + 1)$ are* directed *edges of G' for every t. There is also a directed edge from (u, t) to $(u, t + 1)$ for all u and t. In other words, G' is a "time evolving" version of G and every path in G' is a "time-unfolded" path in G.*

The searchers' movements are controlled, and they are limited to feasible paths on G'. We will refer to the time-stamped nodes visited by a searcher k as $A_k \subseteq N'$, and the combined set of visited time-stamped nodes as $A = A_1 \cup \ldots \cup A_K$.[3] The searchers receive reward by moving onto the same node as the target. This reward is discounted by the time at which this occurs. Given that a target visits time-stamped nodes Y, the searchers receive reward $F_Y(A) = \gamma^{t_A}$, where $t_A = \min\{t : (m, t) \in A \cap Y\}$ (i.e., the first time at which Y intersects A), with the understanding

[3]Note that this formulation assumes that multiple searchers can visit the same node at the same time. The time-stamped nodes visited by each searcher also define its path. For ease of notation, we will sometimes refer to A_k as a searcher path and A as a set of paths.

that $\gamma \in (0,1)$, $\min \emptyset = \infty$, and $\gamma^{\infty} = 0$. Thus, if their paths do not intersect the target, the searchers receive zero reward.

This paper considers two possible assumptions on the target's behavior, which yield the average-case and worst-case search problems. If we make a non-adversarial assumption on the target's behavior, we can utilize a target motion model independent of the locations of the searchers. This yields a probability $P(Y)$ for all possible target paths $Y \in \Psi$, where Ψ is the set of feasible paths the target can take. We can now define the optimization problem in Equation 1.

$$A^* = \underset{A \subseteq N'}{\operatorname{argmax}} \sum_{Y \in \Psi} P(Y) F_Y(A), \tag{1}$$

where $A = A_1 \cup \ldots \cup A_K$, and A_k is a feasible path for all searchers k. Equation 1 maximizes the *average-case* reward given a motion model defined by $P(Y)$. Note that if the target's motion model obeys the Markov property, we can estimate its location efficiently at each time step using matrix algebra.

The average-case optimization problem in Equation 1 does not consider the possibility that the motion model may be incorrect. An alternative assumption on the target's behavior is to assume that it actively avoids the searchers as best possible. For the search problem, this implies that the target chooses path Y that minimizes $F_Y(A)$, which yields the game theoretic optimization problem in Equation 2.

$$A^* = \underset{A \subseteq N'}{\operatorname{argmax}} \min_{Y} F_Y(A), \tag{2}$$

where $A = A_1 \cup \ldots \cup A_K$, and A_k is a feasible path for all searchers k. Here, the searchers' goal is to maximize the *worst-case* reward if the target acts as best it can to reduce reward.

It is important to note that in the above worst-case formulation, the searchers must reveal their entire paths before the target chooses its path, so both sides must have a deterministic optimal strategy. The searchers choose a schedule, and the target chooses the path that best evades that schedule. Additionally, this formulation assumes that the searchers have a "perfect" sensor that will always detect the target if it resides in the same cell. We can relax this assumption somewhat by modeling a non-unity capture probability into the average-case reward function. For the worst-case reward function, the searchers could run the schedule several times to generate a bound on the worst-case probability of missing the target (i.e., each schedule will be guaranteed to have some nonzero probability of locating the target). More sophisticated modeling approaches could also be applied to determine a probabilistic worst-case guarantee with imperfect sensors (Kolling and Carpin, 2009).

Given the worst-case and average-case assumptions on the target's behavior (either of which could be correct), the searchers' goal is to generate a feasible set of paths A such that the reward of both optimization problems are maximized. One option is to use scalarization to generate a final weighted optimization problem as shown in Equation 3. The variable α is a weighting value that can be tuned depending on how likely the target is to follow the average-case model.

$$A^* = \underset{A \subseteq N'}{\operatorname{argmax}} \quad \alpha \sum_{Y \in \Psi} P(Y) F_Y(A) + (1-\alpha) \min_{Y} F_Y(A), \tag{3}$$

where $A = A_1 \cup \ldots \cup A_K$, A_k is a feasible path for all searchers k, and $\alpha \in [0,1]$ is a weighting variable to be tuned based on the application.

While scalarization is a viable approach, we argue that it makes the problem more difficult to solve. Note that the underlying function $F_Y(A)$ is nondecreasing and submodular[4] as is its expectation in Equation 1. In fact, Equation 1 is the efficient search (MESPP) problem (Hollinger et al., 2009b), which can be optimized using the FHPE+SA algorithm. FHPE+SA yields a performance guarantee on the finite-horizon and has been shown to perform near-optimally in practice. In contrast, $\min_Y F_Y(A)$ is nondecreasing but is *not* submodular. Thus, FHPE+SA does not provide a performance guarantee and, in fact, performs poorly in practice. Furthermore, we show below that the optimization problem in Equation 2 does not yield any bounded approximation (unless $P = NP$).[5] Thus, scalarization combines an easier problem with a more difficult problem, which prevents exploiting the structure of the easier MESPP problem.

We propose treating the combined search problem as a resource allocation problem. More precisely, some searchers make the average-case assumption while others make the worst-case assumption. One case where this approach can improve the search schedule is in the (very common) scenario where a portion of the map must be cleared before progressing. In this scenario, several searchers are often assigned to guard locations while waiting for the other searchers to finish clearing. Some of these guards can be used to explore the uncleared portion of the map. An example of this case is shown in our human-robot experiments in Section 6.

If the searchers are properly allocated to the average-case and worst-case tasks, we can generate search schedules with good performance under both assumptions. The decentralized nature of the multi-robot search task makes this approach feasible by allowing different robots to optimize the two separate components of the combined problem. The question now becomes: how many searchers do we assign to each task? Several observations relating worst-case search to graph theoretic node search can help answer this question.

The graph theoretic node search optimization problem is to find a feasible schedule S for a minimal number of searchers such that S is a clearing schedule. In other words, find a schedule that clears the environment of an adversarial evader using the fewest searchers. We will now show the connection between graph theoretic node search and the worst-case search problem described above. Several formal definitions are required to begin our analysis.

Definition 2 *The cleared set $N_C(t) \subseteq N$ is the set of nodes that can no longer contain the target at time t. Conversely the dirty set $N_D(t) \subseteq N$ is the set of nodes that may still contain the target. $N = N_C(t) \cup N_D(t)$, which implies $N_C(t) = N \setminus N_D(t)$ for all t.*

Definition 3 *A schedule $S \subseteq N'$ (i.e., a feasible set of node/time pairs visited by the searchers) is a clearing schedule if the dirty set $N_D(t_f) = \emptyset$ at some time t_f (which also implies $N_D(t) = \emptyset$ for every $t \geq t_f$).*

Definition 4 *The node search number $s(G)$ of a graph G is the minimum number of searchers required to generate a clearing schedule on that graph.*

Given the definitions above, we need only define the recontamination rules to fully formulate the graph theoretic node search problem. We assume that the target exists on the nodes of the

[4] A function $F : \mathbf{P}(N') \rightarrow \Re_0^+$ is called *nondecreasing* iff for all $A, B \in \mathbf{P}(N')$, we have $A \subseteq B \Rightarrow F(A) \leq F(B)$. A function $F : \mathbf{P}(N') \rightarrow \Re_0^+$ is called *submodular* iff for all $A, B \in \mathbf{P}(N')$ and all *singletons* $C = \{(m, t)\} \in \mathbf{P}(N')$, we have $A \subseteq B \Rightarrow F(A \cup C) - F(A) \geq F(B \cup C) - F(B)$.

[5] It is also interesting to note that the more general adversarial sensor network placement problem of the same form described by Krause et al. (2007) has the same hardness property.

graph, and that it cannot move through nodes containing searchers. In addition, the target may be arbitrarily fast, and it is has complete knowledge of the searchers' strategy. Thus, any node in the cleared set that does not contain a searcher and is adjacent to a node in the dirty set will be recontaminated and switch to the dirty set.

Definition 5 *A node v is considered recontaminated at time t if $v \in N_C(t-1)$, v does not contain a searcher at time t, and there exists a path that does not contain a searcher to a node $u \in N_D(t)$. If these conditions are true then $v \in N_D(t)$.*

Since the target may be arbitrarily fast, recontamination will spread until either all nodes are in the dirty set or a frontier of searchers exists between the dirty and cleared sets. Thus, the searchers can make progress towards a clearing schedule by expanding the cleared set while maintaining a frontier to prevent recontamination. Propositions 1 and 2 show the connection between node clearing and the worst-case search problem defined in Equation 2.[6]

Proposition 1 *The value $\min_Y F_Y(A)$ is greater than 0 if and only if A is a clearing schedule (i.e., a set of paths that clear the environment of any target within it).*

Proof Assume that A is not a clearing schedule and $\min_Y F_Y(A) > 0$. Since A is not a clearing schedule, this implies that one or more nodes in G are dirty (i.e., may contain a target) at all times $t = 1, \ldots, T$. W.l.o.g. let Y be a target path that remains within the dirty set for all t. Such a path is feasible due to the assumptions of the evader and the recontamination rules. The dirty set at a given time is by definition not observed by the searchers at that time. Thus, $A \cap Y = \emptyset$, which implies that $F_Y(A) = 0$ for these A and Y. If the target chooses this path, we have a contradiction.

Now, assume that A is a clearing schedule and $\min_Y F_Y(A) = 0$. This assumption implies that $A \cap Y = \emptyset$. By definition of clearing schedule, there is a time T at which the dirty set is the empty set. However, this implies that $A \cap Y \neq \emptyset$ or else there would exist a dirty cell. Again we have a contradiction. ■

Proposition 2 *Let A be restricted to feasible paths for a given number of searchers K. Given a graph G, $\max_{A \subseteq N'} \min_Y F_Y(A) = 0$ if and only if $K < s(G)$, where $s(G)$ is the node search number of G.*

Proof The value of $s(G)$ implies that a clearing schedule exists for all $K \geq s(G)$. By Proposition 1, this implies that a schedule A exists such that $\min_Y F_Y(A) > 0$ for all $K \geq s(G)$.

Similarly, the value of $s(G)$ implies that a clearing schedule does not exist with $K < s(G)$. By Proposition 1, this implies that a schedule A does not exist such that $\min_Y F_Y(A) > 0$ for $K < s(G)$. ■

Proposition 1 shows that any non-zero solution to the optimization problem in Equation 2 will also be a node clearing solution with K searchers. Proposition 2 shows that $s(G)$, the node search number of G, will affect the optimization problem in Equation 2; if $K < s(G)$, then $\min_Y F_Y(A) = 0$ for all A. We now use the result from Proposition 2 to show that no bounded approximation can exist for the worst-case problem unless $P = NP$. The intuition behind this result is that in order to provide a nontrivial approximation guarantee we must achieve nonzero reward whenever the optimal reward is nonzero, which means we have to clear the graph if possible.

[6]Note that Propositions 1 and 2 hold for both monotone and non-monotone clearing schedules. In other words, the propositions hold regardless of whether recontamination is allowed in the schedule.

Theorem 1 *Assuming $P \neq NP$, there can be no bounded polynomial-time approximation algorithm for the worst-case optimization problem in Equation 2.*

Proof Let n be an arbitrary problem instance defined by the graph input G and number of searchers K. Let $f(n)$ be any strictly positive function of the problem instance (e.g., a function of the number of vertices, number of searchers, graph diameter, etc.). We prove that if there exists a polynomial-time algorithm that is guaranteed to find $A' \subseteq N'$ such that $\min_Y F_Y(A') \geq f(n) \max_{A \subseteq N'} \min_Y F_Y(A)$, then $P = NP$. Note that $f(n) > 1$ for any n would imply that $\min_Y F_Y(A')$ is greater than optimal, which is clearly not possible.

Given K searchers and a graph $G = (N, E)$, assume that a polynomial-time algorithm $\mathcal{F}$ exists that is guaranteed to generate paths A' s.t. $\min_Y F_Y(A') \geq f(n) \max_{A \subseteq N'} \min_Y F_Y(A)$. From Proposition 2, we know that $\max_{A \subseteq N'} \min_Y F_Y(A) = 0$ if and only if $K < s(G)$. Thus, we can test whether $s(G) \leq K$ by testing whether $\min_Y F_Y(A')$ is non-zero. Determining whether $s(G) \leq K$ is a known NP-hard problem (Kehagias et al., 2009). Thus, $\mathcal{F}$ would solve an NP-hard problem in polynomial time, which would imply $P = NP$. ■

Theorem 1 shows that we cannot expect to generate any approximation guarantee for the worst-case problem in Equation 2, and hence the worst-case component of the combined problem. However, as mentioned above, the average-case problem admits a constant factor approximation on the finite-horizon using sequential allocation (Hollinger et al., 2009b). This analysis motivates our decision to split the searchers into two groups: average-case searchers and worst-case searchers, which preserves some approximation guarantees in the average-case. From Proposition 2, we know that at least $s(G)$ searchers must make the worst-case assumption to generate a schedule with any nonzero worst-case reward. These theoretical results motivate the use of guaranteed search algorithms that minimize the attained search number. However, current guaranteed search algorithms in the literature do not minimize clearing time. We show how this limitation can be overcome in the next section.

4 Algorithm Description

Drawing off the observations in the previous section, we can design an algorithm that both clears the environment of an adversarial target and performs well with respect to a target motion model. The first step in developing a combined algorithm is to generate a guaranteed search schedule that improves clearing time with a given number of searchers. We extend the Guaranteed Search with Spanning Trees (GSST) algorithm (Hollinger et al., 2010) to do just this.

The GSST algorithm is an anytime algorithm that leverages the fact that guaranteed search is a linear-time solvable problem on trees. Barrière et al. (2002) show how to generate a recursive labeling of a tree (we will refer to this labeling as B-labeling) in linear-time. Informally, the labeling determines how many searchers must traverse a given edge of the tree in the optimal schedule (a non-recursive B-labeling algorithm is given in Algorithm 1). If an edge label is positive, it means that one or more searchers must still traverse that edge to clear the tree. From the labeling, a guaranteed search algorithm can be found using the minimal number of searchers on the tree. GSST generates spanning trees of a given graph and then B-labels them. The labeling is then combined with the use of "guards" on edges that are not in the spanning tree to generate a guaranteed search schedule on arbitrary graphs. In prior work, searchers executing clearing on the underlying tree would call

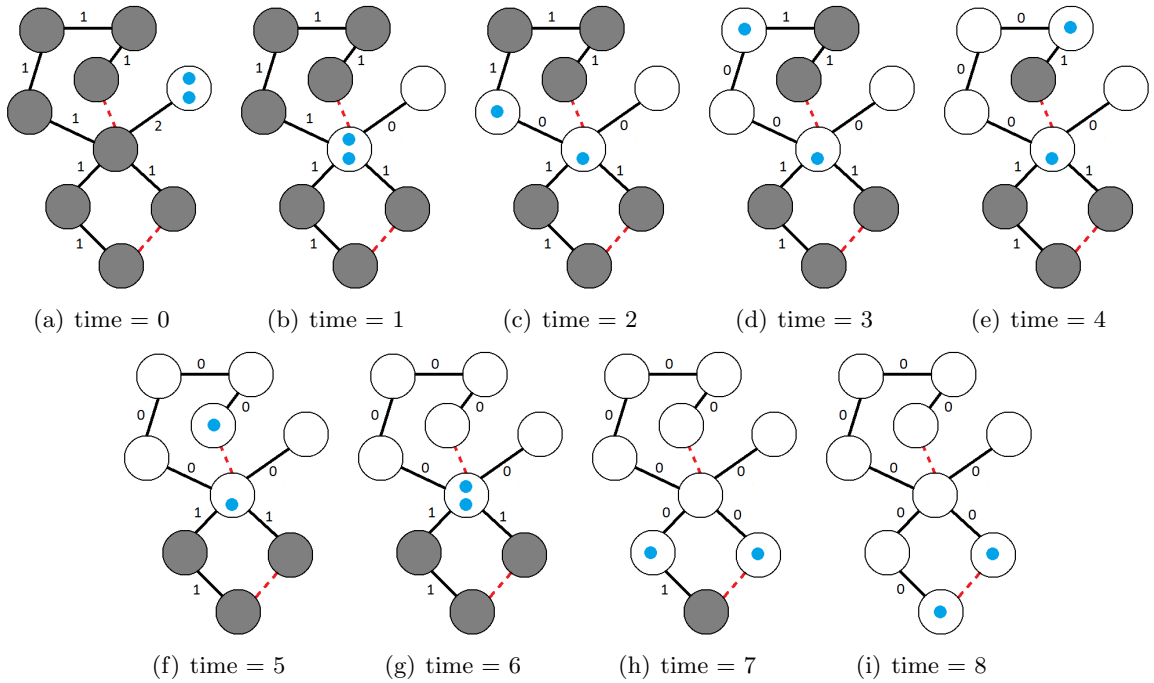
for guards when they could not progress due to edges non-tree edges. The number of guards was reduced due to the temporal aspects of the search (i.e., guards would not need to remain in place during the entire schedule). However, GSST gives no mechanism for utilizing more searchers than the minimal number. Thus, it does not optimize clearing time.

Algorithm 1 B-labeling for Trees

```
1: Input: Tree T = (N, E), Start node b ∈ N
2: O ← leafs of T
3: while O ≠ ∅ do
4:    l ← any node in O, O ← O \ l
5:    if l is a leaf then
6:       e ← only edge of l, λ(e) = 1
7:    else
8:       e ← unlabeled edge of l
9:       e_1, ..., e_d ← labeled edges of l
10:      λ_m ← max{λ(e_1), ..., λ(e_d)}
11:      if multiple edges of l have label λ_m then
12:         λ(e) ← λ_m + 1
13:      else
14:         λ(e) ← λ_m
15:      end if
16:   end if
17:   if l ≠ b and parent(l) has exactly one unlabeled edge then
18:      O ← O ∪ parent(l)
19:   end if
20: end while
21: Output: B-labeling λ(E)
```

Algorithm 2 shows how GSST can be modified to improve clearing time with additional searchers. The algorithm uses B-labeling to guide different groups of searchers into subgraphs that are cleared simultaneously. Algorithm 2 does not explicitly use guards on non-tree edges; the guards are instead determined implicitly from the B-labeling (see Figure 2 for an example). Thus, we refer to it as Guardless-GSST or G-GSST. Note that the use of B-labeling allows the searchers to perform the schedule asynchronously. For example, a searcher who arrives at a node does not need to wait for other searchers to finish their movement before clearing the next node (assuming that the move does not cause a recontamination).

G-GSST can be combined with model-based search algorithms to generate a combined search schedule. We augment G-GSST with Finite-Horizon Path Enumeration and Sequential Allocation (FHPE+SA) to yield our combined search algorithm. In short, FHPE+SA algorithm has each searcher plan its own path on a finite-horizon and then share that path with other searchers in a sequential fashion (see Hollinger et al. (2009b) for a formal description). In other words, one searcher plans its finite-horizon path and shares it with the other searchers; then another searcher plans its finite-horizon path and shares it, and so on. After the initial sequential allocation, searchers replan asynchronously as they reach replanning points in the environment. We assume that the searchers have all-to-all communication, which allows each searcher to share its plan with any other searcher.



(a) time = 0 (b) time = 1 (c) time = 2 (d) time = 3 (e) time = 4

(f) time = 5 (g) time = 6 (h) time = 7 (i) time = 8

Figure 2: Example of clearing a simple environment represented as a graph using G-GSST. Solid edges denote edges in the spanning tree representation, and dotted edges denote non-tree edges. The searchers (circles) follow the B-labeling (shown next to each edge) to clear the graph. Cells that may contain an adversary at each time are shaded. One searcher remains as an implicit guard in the middle of the graph (any clearing move would cause contamination) while the other searcher moves to clear the top portion of the graph. Once the top portion is cleared, both searchers then progress to clear the bottom of the graph.

This assumption may be relaxed if certain areas of the map can be searched without affecting the schedules of members in other areas.

Algorithm 3 shows how G-GSST can be augmented with FHPE+SA to yield a combined algorithm. For the remainder of this paper, we will refer to searchers executing G-GSST as *clearers* and searchers executing FHPE+SA as *optimizers*. An important quality of the combined search is that, depending on the actions of the optimizers, the schedule may be non-monotone (i.e., it may allow for recontamination). However, since the schedule of the clearers is monotone, the search will still progress towards clearing the environment of a worst-case target.

It is important to note that the schedule of the clearers can be generated in conjunction with the optimizers. For instance, we can modify the clearing schedule based on the actions of the optimizers by pruning portions of the map that happen to be cleared by the optimizers.[7] This extension provides a tighter coupling between the clearing and average-case optimization. Alternative methods for integrating the search plans are discussed in Section 7.

[7]Note that implementing this extension requires ensuring that ignoring a portion of the map cleared by the optimizers does not lead to later recontamination. This can be determined by dynamically relabeling the spanning tree and taking into account the cleared set. Since labeling is a linear-time operation, this does not significantly affect runtime.

Algorithm 2 Guardless Guaranteed Search with Spanning Trees (G-GSST)

```
Input: Graph G, Spanning tree T with B-labeling, Searchers K_g
while graph not cleared do
  for all searchers do
    if moving will recontaminate then
      Do not move
    else if positive adjacent edge label exists then
      Travel along an edge with smallest positive label
      Decrement edge label
    else
      Move towards closest positive edge label while remaining in the clear set
    end if
  end for
  if no moves possible without recontamination then
    Return failure
  end if
end while
Return feasible clearing schedule
```

Finally, we generate many spanning trees (similar to the GSST algorithm) to develop many search schedules (see Algorithm 3). These schedules range in quality and also tradeoff worst-case and average-case performance. At any time, the user can stop the spanning tree generation and choose to execute the search schedule that best suits his/her needs. This yields an anytime solution to the worst-case/average-case search problems: one that, with increasing runtime, continues to generate progressively better solutions.

4.1 Decentralized and Online Operation

Once a schedule is found using Algorithm 3, the execution can be modified online with decentralized computation. During execution the optimizers can replan at every iteration based on new information that becomes available (e.g., noisy measurements of the target's positions). In addition, the clearers can continue their schedule, as long as their movement does not cause recontamination, even if the other clearers fall behind in their schedules. We do not provide a mechanism for dealing with online changes to the environment, and we leave this as an avenue for future work.

Modifications during execution are possible because of the decentralized nature of the FHPE+SA and G-GSST algorithms. A G-GSST clearing schedule is determined by the spanning tree representation, which is shared by the searchers. The searchers only need to share changes in the cleared set and the current edge labeling to determine their next move. Similarly, the sequential nature of FHPE+SA allows each searcher to plan its own actions and then share that information with its teammates. If the information is not current, or a robot fails, the average-case performance could suffer somewhat, but the search team can still continue operating. Thus, the optimizers are robust to robot failure. In Section 6 we show a decentralized implementation of the combined algorithm on a human-robot search team.

Algorithm 3 Anytime combined search algorithm

```
Input: Graph G, Searchers K, Maximum computation time τ
while computation time left do
  Generate spanning tree T of G and B-label it
  for K_g = K down to K_g = 1 do
    Assign K_g guaranteed searchers and K − K_g efficient searchers
    while graph not cleared do
      for all searchers do
        if guaranteed searcher then
          Run G-GSST step
        else
          Run FHPE+SA step
        end if
      end for
      if no moves possible then
        Break
      end if
    end while
    if clearing feasible then
      Store strategy
    else
      Break
    end if
  end for
end while
if strategies stored then
  Return strategy with maximal αR_avg + (1 − α)R_worst
else
  Run FHPE+SA to maximize R_avg (clearing schedule not found)
end if
```

5 Theoretical Analysis

5.1 Average-Case Performance Bounds

We now show that the average-case component of the combined algorithm is a bounded approximation on the finite-horizon. Let A be the set of nodes visited by the paths returned by the FHPE+SA algorithm. Let O be the set of nodes visited by the optimal average-case paths (maximizes Equation 1), and let O_k be the set of nodes visited by searcher k on the optimal path. Let $F^{AC}(A) = \sum_{Y \in \Psi} P(Y) F_Y(A)$, i.e. the average-case reward for a path A as described in Section 3. The average-case reward is bounded as in Theorem 2. This bound is simply the FHPE+SA bound for $K - K_g$ searchers.

Theorem 2

$$F^{AC}(A) \geq \frac{F^{AC}(O_1 \cup \ldots \cup O_{K-K_g}) - \epsilon}{2}, \tag{4}$$

where K is the total number of searchers, K_g is the number of searchers used for the clearing schedule, and ϵ is the finite-horizon error ($\epsilon = R\gamma^{d+1}$, where R is the reward received for locating the target, γ is the discount factor, and d is the search depth).

Proof This bound is immediate from the theoretical bounds on sequential allocation (Singh et al., 2007), the monotonic submodularity of F^{AC} (Hollinger et al., 2009b), and the fact that $K - K_g$ searchers are deployed to perform sequential allocation. The addition of searchers performing G-GSST cannot decrease $F^{AC}(A)$ due to the monotonicity of F^{AC} and the monotonicity of the cleared set in the G-GSST schedule. ■

We can extend the FHPE+SA component of the combined algorithm to optimize over several different known models. If there are M models being considered, and each has a probability of $\beta_1 \ldots \beta_M$, we can optimize the weighted average-case objective function in Equation 5.

$$F(A) = \beta_1 \sum_{Y \in \Psi} P_1(Y) F_Y(A) + \ldots + \beta_M \sum_{Y \in \Psi} P_M(Y) F_Y(A), \tag{5}$$

where $\beta_1 + \ldots + \beta_M = 1$, and $P_m(Y)$ describes the probability of the target taking path Y if it is following model m.

If all models obey the Markov assumption, this linear combination of models can be estimated using matrices as if it were a single model without an increase in computation. Additionally, monotonic submodularity is closed under nonnegative linear combination, so Theorem 2 holds in this extended case.

5.2 Computational Complexity

The labeling component of G-GSST requires visiting each node once and is $O(N)$, where N is the number of nodes in the search graph. The G-GSST traversal is $O(NK_g)$, where K_g is the number of guaranteed searchers. Finally, the FHPE+SA component replanning at each step is $O(N(K - K_g)b^d)$, where b is the branching factor of the search graph, and d is the FHPE search depth. Thus, generating a single plan with the combined algorithm is $O(N + NK_g + N(K - K_g)b^d)$. This is linear in all terms except the FHPE search depth, which often can be tuned to a very low number depending on the application.

6 Experimental Results

6.1 Simulated Results

We performed simulated testing of our combined search algorithm in several complex environments. The Newell-Simon Hall (NSH) and National Gallery (museum) maps were used in prior work and were discretized by hand into rooms and hallways. The cave map is taken from the Player/Stage project (Gerkey et al., 2003) and represents a sparsely cluttered environment. The McKenna MOUT map is a polygonal representation of a military testing facility. The cave and MOUT maps were discretized automatically using a constrained Delaunay triangulation (Shewchuk, 2002), which allows for size constraints on the triangles to model limited sensor range. The Merced and SDR maps were used in prior work with the GRAPH-CLEAR problem (Kolling and Carpin, 2008). Kolling and Carpin's method for discretization is not suitable for node search because it does not

generate convex regions. We discretize the SDR and Merced maps using a region growing method: each region is grown until its bounding boxes contains a number of obstacle pixels (set to 200). This region growing approach assumes that the evader is large enough that it cannot hide behind small obstacles. In addition, we set a maximum length for the bounding box to 100 pixels to model a limited sensor range. Table 1 gives the attained search numbers for these environments (i.e., the fewest number of searchers for which a clearing strategy was found). Figure 3 shows discretized maps of the six test environments.

Table 1: Statistics for test environments of increasing complexity (attained search numbers found using G-GSST).

	Cave	Office	Museum	Merced	SDR	MOUT
Nodes	43	61	70	130	188	227
Edges	46	65	93	197	258	289
Attained search number	3	3	5	7	8	9

6.1.1 G-GSST Comparison

We tested the G-GSST algorithm to determine its effectiveness at generating schedules with low clearing times. We compared G-GSST to a sequential greedy algorithm that iteratively decreases the dirty set on the receding horizon. This variant of sequential allocation is shown in Algorithm 4 for solving the guaranteed search problem. We refer to this algorithm as Receding Horizon Greedy Clearing (RHGC). This algorithm was also discussed in more detail in our prior work (Hollinger et al., 2010).

Algorithm 4 Receding Horizon Greedy Clearing

Input: Graph $G = (N, E)$, Number of searchers K, Start node $b \in N$
$t \leftarrow 0$, $s_k(0) \leftarrow b$ for all k
while $N_D \neq \emptyset$ **do**
  for searcher $k = 1$ to K **do**
    % P is a feasible path for searcher k in the search graph to horizon d
    % $S = s_1(0), \ldots, s_K(0), \ldots, s_1(t+d), \ldots, s_{k-1}(t+d)$ is a partial schedule
    % $N_D(t+d|P,S)$ is the dirty set at time $t+d$ given path P and schedule S are chosen
    $\{s_k(t+1), \ldots, s_k(t+d)\} \leftarrow \text{argmin}_P \, |N_D(t+d|P,S)|$
  end for
  if all searchers cannot move without recontamination **then**
    Return failure
  end if
  $t \leftarrow t+1$
end while
Return clearing schedule S and clearing time t

We also compare to the Parallel Stochastic Hill-Climbing for Small Teams (PARISH) algorithm introduced by Gerkey et al. (2005). PARISH stochastically samples a limited space of possible paths

while allowing for recontamination. As a result, the cleared set grows and shrinks until eventually a clearing schedules is found. In addition, PARISH biases the chosen paths towards those that clear more cells. The version of PARISH implemented in this paper has been modified from the version presented in prior work. Our implementation uses an exponential bias, which places significantly more weight on plans with that clear more cells. We have also modified the algorithm to consider a target located on the nodes rather than on the edges of the graph. Finally, by forcing PARISH to restart whenever a plan is unsuccessful, we allow the algorithm to run for a given period of time and return the best solution. Together these modifications improve the performance of PARISH over what was presented in prior work (Gerkey et al., 2005).

Figure 4 shows a comparison of G-GSST with PARISH and RHGC. Since PARISH and G-GSST are stochastic, they each were run for one minute in each trial, restarting after each successful or failed strategy. PARISH trials were considered failed after the schedule exceeded 1000 steps without clearing the environment. The best feasible strategy was taken after time ran out. RHGC was run once to completion, which took approximately one second in the larger environments. Even though the runtime of RHGC is lower, it does not provide any mechanism for improving its solution after completion. Stochastic algorithms like PARISH and G-GSST, on the other hand, allow the continued generation of potential strategies with increasing runtime.

In all environments, RHGC requires more searchers than G-GSST to find a feasible clearing schedule. As a result, fewer searchers will be available for assignment as optimizers. Furthermore, G-GSST yielded faster clearing times than RHGC in almost all cases, particularly when only few searchers are available. The one exception is on the MOUT map where G-GSST and RHGC perform competitively. This is likely due to the regularity of the triangular decomposition, which somewhat negates the benefit from long-term planning on the spanning tree representation. The PARISH algorithm was able to find strategies with few searchers, but the resulting clearing times were very large. The large clearing times are due to PARISH's stochastic nature and its use of recontamination. PARISH's clearing and recontamination hill climbing can take a very large number searcher moves before finding a clearing schedule.

The results above show that G-GSST yields fast clearing times with few searchers, which leads to better worst-case performance as well as more searchers available to improve average-case performance. We now examine the effectiveness of utilizing the additional searchers to optimize a target motion model.

6.1.2 Combined Algorithm Comparison

We ran our algorithm on 2000 random spanning trees on all maps, and we compared these results to the schedules found by pure average-case and pure worst-case algorithms. Table 2 gives a summary of these results. The first row shows the average-case steps (i.e., at each step one or more searchers moves between nodes) to capture a randomly moving target using FHPE+SA with all searchers (a lookahead distance of five was used for FHPE on these and all other trials)[8] This strategy does not have worst-case guarantees if the model is incorrect. The second row shows the best clearing time using GSST on 10,000 random spanning trees. Note that GSST does not optimize for clearing

[8]The expected capture steps for FHPE+SA were determined over 10,000 trials with a randomly moving target. The expected capture steps for the combined algorithm were computed in closed form. If the environment is cleared at time T and the probability of capture is $P(t)$ for all t, the expected capture steps is $E(T_C) = \sum_{t=1}^{T} P(t)t$. The probability of capture $P(t)$ was computed using matrix multiplication for a Markovian target model.

time and only utilizes the minimal number of searchers required to clear. The results show that our combined algorithm yields much lower clearing times than those found with GSST. This is due to the use of additional searchers as in Algorithm 2. In addition, our combined algorithm reduces the average capture time by over 75% in the office when compared to GSST. Furthermore, a worst-case guarantees on some maps can be gained by sacrificing only one step of average capture time.

The user can determine the weighting of average-case vs. worst-case performance by tuning the value of the α parameter, which explores the Pareto-optimal frontier of solutions. This frontier forms a convex hull of solutions, any of which can be selected after runtime. Figure 5 gives a scatter plot of average-case versus worst-case capture steps for all feasible schedules on the six test maps. The figure also shows the number of clearers, K_g, used for each data point. Note that the total number of searchers is fixed throughout the trials, and the remainder of the searchers performed FHPE+SA. In most cases, the lowest average-case capture times are from the lowest K_g. This is due to more searchers being used for average-case search and fewer for clearing. These results demonstrate the utility of using FHPE+SA searchers in the schedule. Similarly, the lowest clearing times are typically with $K_g = K$ (i.e., all searchers are guaranteed searchers). Note that better solutions yield points to the left/bottom of these plots.

Table 2: Average-case and worst-case capture steps comparison. The average-case is the expected steps to capture a randomly moving target. The worst-case is the number of steps to clear the environment. The proposed combined algorithm is compared to a purely average-case method (FHPE (Hollinger et al., 2009b)) and a purely worst-case method (GSST (Hollinger et al., 2010)).

	Cave ($K = 5$)	NSH ($K = 5$)	Museum ($K = 7$)
FHPE+SA	A.C. 11.1 W.C. ∞	A.C. 5.4 W.C. ∞	A.C. 9.3 W.C. ∞
GSST	A.C. 14.7 W.C. 30	A.C. 24.6 W.C. 74	A.C. 21.8 W.C. 47
Combined ($\alpha = 0.25$)	A.C. 12.6 W.C. 23	A.C. 15.5 W.C. 39	A.C. 14.9 W.C. 37
Combined ($\alpha = 0.75$)	A.C. 11.6 W.C. 24	A.C. 8.6 W.C. 48	A.C 12.9 W.C. 43
Combined ($\alpha = 0.99$)	A.C. 11.6 W.C. 24	A.C. 6.9 W.C. 74	A.C. 12.8 W.C. 51
	Merced ($K = 10$)	SDR ($K = 10$)	MOUT ($K = 13$)
FHPE+SA	A.C. 13.3 W.C. ∞	A.C. 16.5 W.C. ∞	A.C. 22.6 W.C. ∞
GSST	A.C. 45.7 W.C. 112	A.C. 56.0 W.C. 135	A.C. 65.8 W.C. 150
Combined ($\alpha = 0.25$)	A.C. 32.4 W.C. 70	A.C. 48.7 W.C. 111	A.C. 44.9 W.C. 81
Combined ($\alpha = 0.5$)	A.C. 25.9 W.C. 75	A.C. 48.7 W.C. 111	A.C. 33.7 W.C. 87
Combined ($\alpha = 0.75$)	A.C. 25.9 W.C. 75	A.C. 33.5 W.C. 127	A.C 30.3 W.C. 92
Combined ($\alpha = 0.99$)	A.C. 20.1 W.C. 97	A.C. 27.8 W.C. 154	A.C. 28.7 W.C. 121

Figure 6 shows the number of feasible clearing schedules generated per second for various test environments with increasing available searchers. As the number of available searchers increases, the number of schedules generated initially increases as clearing schedules become easier to find. As the number of searchers continues to increase, the schedules per second decreases due to the added computational complexity. Even in the large environments with few searchers, the combined algorithm is often able to generate more than one feasible clearing schedule per second.

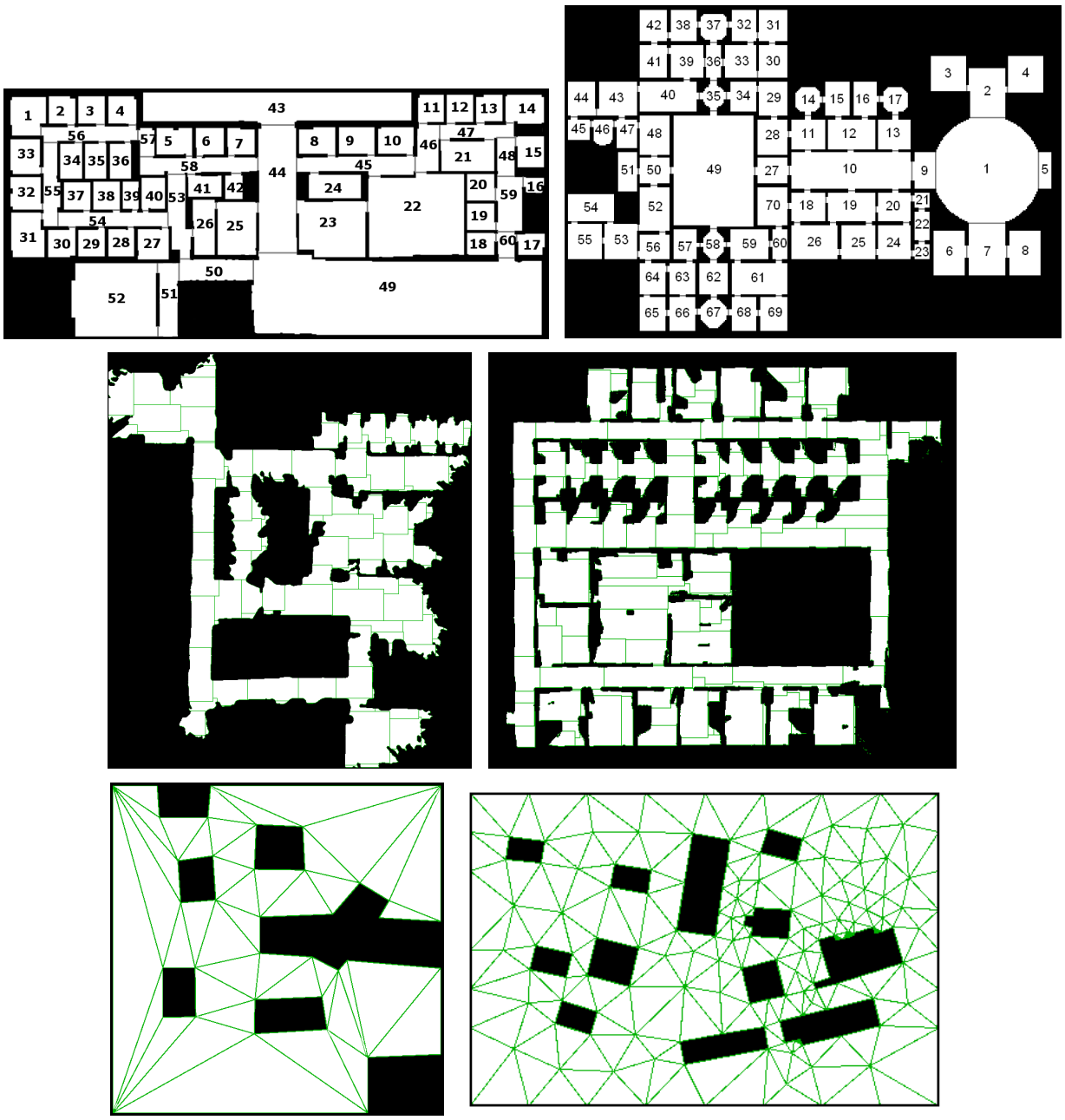


Figure 3: Maps of environments used for simulated coordinated search. The NSH (top left) and National Gallery (top right) were discretized by hand, the Merced (middle left) and SDR (middle right) were discretized using an orthogonal region growing method, and the cave (bottom left) and MOUT (bottom right) were discretized using a constrained Delaunay triangulation.

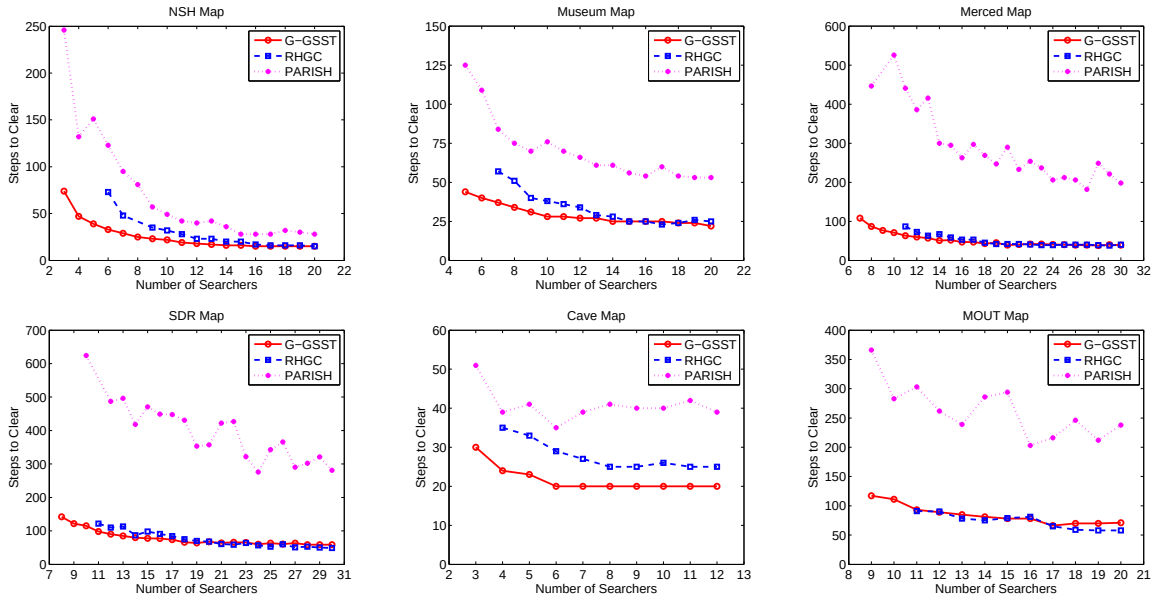


Figure 4: Clearing time comparison of the proposed G-GSST algorithm with PARISH (Gerkey et al., 2005) and RHGC (Algorithm 4). G-GSST and PARISH were given one minute to generate solutions, and the best was taken after the time. RHGC was run once to completion. The graph shows the number of steps to clear the graph (i.e., one or more searchers moves in a step) with increasing searchers. Values are only shown if a clearing schedule was found with that number of searchers.

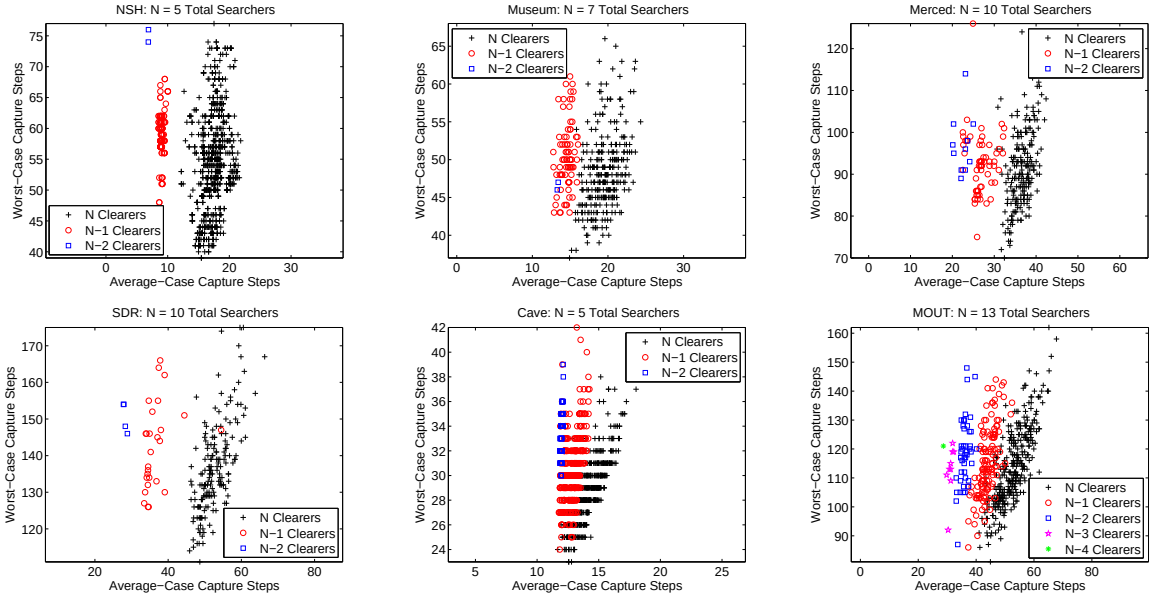


Figure 5: Scatter plot of search schedules from the combined algorithm run on 2000 spanning trees. Each data point represents a schedule generated by a different searcher allocation and spanning tree input to Algorithm 3. The data points are colored based on the allocation of searchers to the guaranteed and efficient search roles. The total searchers in each environment remains constant throughout the trials.

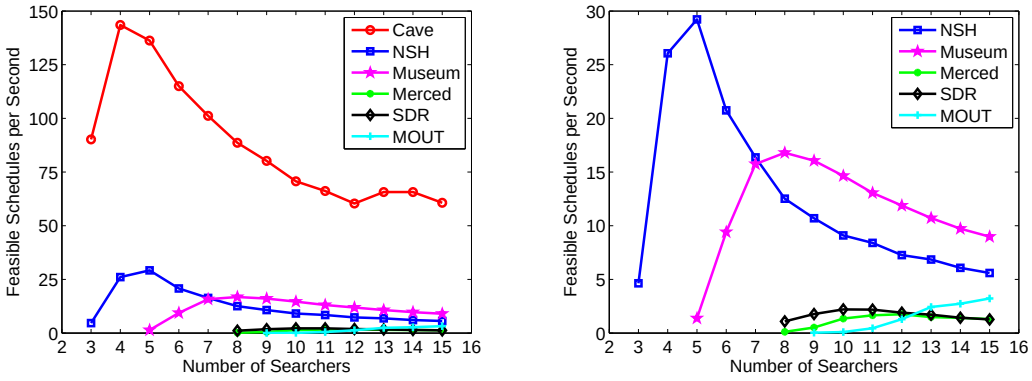


Figure 6: Number of feasible clearing schedules per second generated by the combined algorithm with a varying number of searchers. As the number of searchers increases, the number of schedules generated first increases as schedules become easier to find and then decreases as the computational complexity becomes greater. The same graph is shown zoomed out (left) and then zoomed in (right) for additional detail. Schedules with fewer searchers than those displayed were not found after examining 1000 trees on each graph.

6.1.3 Imperfect Motion Models

In some cases, the target will not act in accordance with the assumed model, but it also will not be fully adversarial. To test the robustness of the combined algorithm in these cases, we examined several complex motion patterns for the target. We introduce three target behaviors that we refer to as *daily*, *gallery*, and *escape*. Targets following the daily behavior act as if they are performing everyday tasks in an office building: they spend significantly more time in offices, they rarely backtrack when moving, and they only stop in hallways for a short time. Targets following the escape behavior act as if they are attempting to find an exit to the building: they never backtrack, rarely return to a room they have already been in, and spend very little time standing still. Finally, targets performing the gallery behavior act as if they are viewing art in a museum: they spend most of their time moving slowly through rooms that they have not yet visited.

The daily, gallery, and escape behaviors are implemented using a set of randomized rules and a memory. The simulated target remembers all locations that it has previously visited, and its actions are determined based on a randomization of the set schedule. The randomized policies are summarized below. They were implemented for a simulated target using a pseudorandom number generator and a complete memory of the target's previous path. The searchers do not have access to the target's true model during planning. None of the above strategies can be fully modeled using a Markovian model because they require remembering history of where the target has been.

Daily Behavior

- if in office (dead-end room): 20% leave office and move to hallway, 80% stay in office
- if not in office: 5% move to last room or hallway visited, 20% move to an office (if one ore more adjacent offices have already been visited visited, bias towards these offices), 60% keep moving to non-office (exclude hallways previously visited since leaving an office), 15% stay in current hallway

Gallery Behavior

- if in dead-end room: 80% leave this room, 20% stay in this room
- if not in dead-end room: 20% stay in this room, 20% move to last room previously visited, 60% move to a room not visited
- if all adjacent rooms have been visited and not in dead-end: move towards closest unvisited room

Escape Behavior

- if in dead-end room: 90% leave this room, 10% stay in this room
- if not in dead-end room: 5% move to last room or hallway previously visited, 30% move to dead-end room (only those never visited), 65% keep moving (only to a room never visited), 0% stay
- if all adjacent rooms have been visited and not in a dead-end: move towards closest unvisited room

We implemented the three behaviors described above on simulated targets in the NSH and museum environments. Searchers perform the combined algorithm assuming a random motion model. Figure 7 shows that the daily, gallery, and escape behavior all yield capture times very close to a target truly moving randomly. Thus, searchers that assume a random model will often quickly find targets performing these complex behaviors. For comparison, we also implement a stationary target. The stationary target does not move throughout the environment, which requires the searchers to examine all possible locations. The average time to find a stationary target is closer to the worst-case clearing time than that of the targets moving through the environment. These results are expected because a moving target is, on average, easier to find than a stationary one (Hollinger et al., 2009b). Note that, since the schedule clears the environment, a target following any model will be found by the time the schedule is completed.

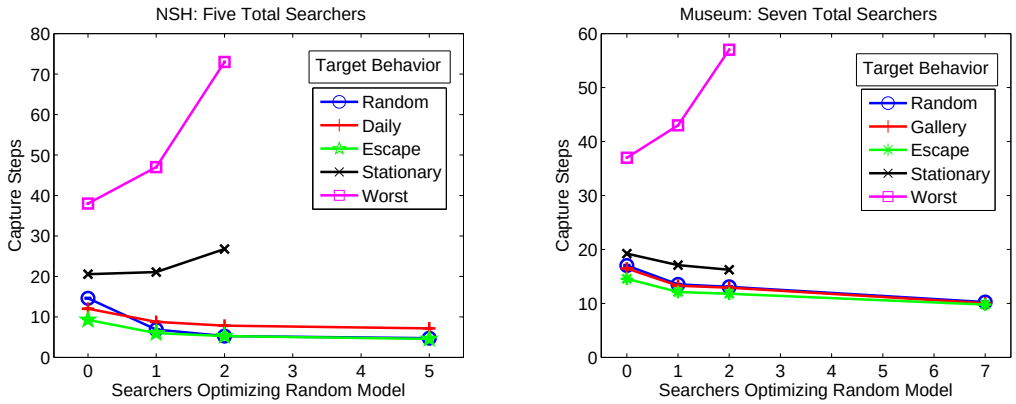


Figure 7: Capture times using various models of the target's movement in the NSH and museum environments. A fixed number of searchers was used with some dedicated to clearing and some to optimizing a random motion model. Averages were over 10,000 runs. SEMs are small and omitted.

6.2 Human-Robot Teams

To examine the feasibility of our algorithm for real-world applications, we ran several experiments with a human/robot search team. We conducted these experiments on a single floor of an office building as shown in Figure 8. For comparison, we used the same building as in our prior work in worst-case search Hollinger et al. (2010). Two humans and a single Pioneer robot share their position information through a wireless network, and the entire team is guided by a decentralized implementation of Algorithm 3. The robot's position is determined by laser AMCL, and it is given waypoints through the Player/Stage software (Gerkey et al., 2003). The humans input their position through the keyboard, and they are given waypoints through a GUI.

The human-robot search team uses a shared pre-processing step to determine a candidate spanning tree and searcher allocation. Once a schedule is found that properly trades-off average-case and worst-case performance, the execution is done online with distributed computation. Each processor plans the actions of a single searcher (human or robot), and the schedule continues executing even if some searchers fall behind. If moving will cause recontamination, and the searcher is a clearer, it will wait until the other searchers catch up before continuing. This avoids strict synchronization of the schedule, which reduces communication between the team. These online modifications are

particularly desirable for real-world applications because they avoid many of the robustness issues with executing an offline schedule.

In the first experiment, all three searchers (humans and robot) were assigned as clearers. This configuration takes 178 seconds to clear the floor of a potential adversary, and it yields an expected capture time of 78 seconds w.r.t. the random model. We also calculated the expected capture time if the model of the target's speed is 50% off (i.e., the target is moving 50% slower than expected). With this modification, the expected capture time increases to 83 seconds.

In the second experiment, the mobile robot was switched to an optimizer, which took 177 seconds to clear and yielded an expected capture time of 73 seconds. The expected capture time with a 50% inaccurate model was 78 seconds. Thus, switching the mobile robot to an optimizer yields a 5 second decrease in expected capture time without sacrificing any worst-case capture time. The 5 second decrease remains even if the model is inaccurate. This further confirms the simulated results in Figure 7, showing that the schedules are robust to changes in the target's motion model.

In both experiments, the schedule clears the left portion of the map first and then continues to clear the right portion. Video Extensions 1 and 2 include playback of both simulated and human-robot results (see Appendix). In this environment, the assignment of the robot as a clearer does not decrease the clearing time. The reason is that the robot spends a significant portion of the schedule as a redundant guard. Consequently, the right portion of the map is not searched until very late in the clearing schedule. In contrast, when the robot is used as an optimizer, the right hallway is searched early in the schedule, which would locate a non-adversarial target moving in that area.

In addition, our results with a human/robot team demonstrate the feasibility of the communication and computational requirements of our algorithm on a small team. The initial plan generation stage is distributed among the searcher network, and once stopped by the user, the best plan is chosen. During execution, there is a broadcast communication requirement as the searchers share their positions on the search graph. This small amount of information was easily handled by a standard wireless network in an academic building.

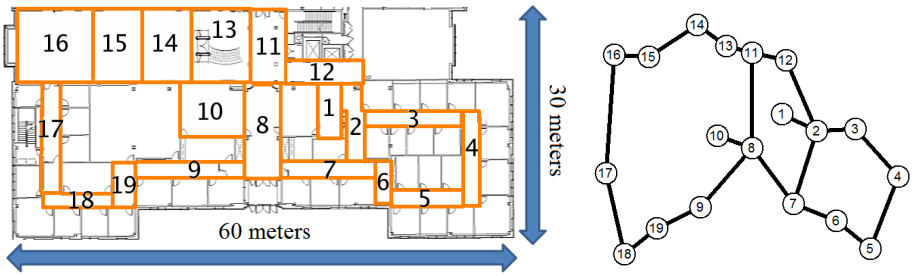


Figure 8: Map of office building (left) used for experiments with a human/robot search team, and corresponding search graph (right). The same environment was previously used to validate worst-case search techniques (Hollinger et al., 2010). The human/robot search team (see Figure 1) started at the entrance in cell eight.

7 Conclusion and Future Work

We have presented an algorithm for improving the efficiency of multi-robot clearing by generating multi-agent search paths that both clear an environment of a potential adversary *and* optimize over a non-adversarial target motion model. In addition, we have introduced a new algorithm for finding clearing schedules with low search times by utilizing the underlying spanning tree to guide clearing. We have integrated this approach into a combined algorithm to generate schedules that perform well under both average-case and worst-case assumptions on the target's behavior. We have shown through simulated experiments that our combined algorithm performs well when compared to search algorithms for both guaranteed and efficient search. In addition, we have demonstrated the feasibility of our algorithm on a heterogeneous human/robot search team in an office environment.

Our current algorithm does not directly use a weighting variable α to incorporate confidence in the model. Instead, we cache many solutions and allow the user to choose one at runtime in an anytime fashion. One method for directly incorporating α would be first to determine the lowest number of searchers capable of clearing and assign the remainder of the searchers proportional to α. An alternative would be to have searchers switch between G-GSST and FHPE+SA during the schedule with some probability related to α. This would allow for dynamic switching but would require careful tuning of the switching function.

Additional future work includes more extensive robustness testing with inaccurate motion models and analysis of the communication requirements with very large search teams. Future field testing includes applications in emergency response, military reconnaissance, and aged care. Combining search guarantees against an adversarial target with efficient performance using a model has the potential to improve autonomous search across this wide range of applications. This paper has contributed the first algorithm and results towards this goal.

Acknowledgment

The authors gratefully acknowledge Joseph Djugash, Ben Grocholsky, and Debadeepta Dey from CMU for their insightful comments and assistance with experiments. We thank volunteer firefighter (and professional roboticist) Seth Koterba for feedback on the system. This work is funded in part by National Science Foundation Grant No. IIS-0426945.

The final, definitive version of this paper has been published in *The International Journal of Robotics Research*, Vol. 29, No. 8, July 2010 by SAGE Publications Ltd, All rights reserved. ©Geoffrey Hollinger, Sanjiv Singh, and Athanasios Kehagias, 2010. It is available at:

`http://online.sagepub.com/`

References

Alspach, B. (2006). Searching and sweeping graphs: A brief survey. *Matematiche*, 59:5–37.

Barrière, L., Flocchini, P., Fraigniaud, P., and Santoro, N. (2002). Capture of an intruder by mobile agents. In *Proc. 14th ACM Symp. Parallel Algorithms and Architectures*, pages 200–209.

Borie, R., Tovey, C., and Koenig, S. (2009). Algorithms and complexity results for pursuit-evasion problems. In *Proc. Int. Joint Conf. Artificial Intelligence*.

Calisi, D., Farinelli, A., Locchi, L., and Nardi, D. (2007). Multi-objective exploration and search for autonomous rescue robots. *J. Field Robotics*, 24(8–9):763–777.

Fomin, F. and Thilikos, D. (2008). An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399:236–245.

Gerkey, B., Thrun, S., and Gordon, G. (2005). Parallel stochastic hill-climbing with small teams. In *Proc. 3rd Int. NRL Workshop Multi-Robot Systems*.

Gerkey, B., Vaughan, R., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proc. Int. Conf. Advanced Robotics*, pages 317–323.

Guibas, L., Latombe, J., LaValle, S., Lin, D., and Motwani, R. (1999). Visibility-based pursuit-evasion in a polygonal environment. *Int. J. Comp. Geometry and Applications*, 9(5):471–494.

Hollinger, G., Kehagias, A., and Singh, S. (2009a). Efficient, guaranteed search with multi-agent teams. In *Proc. Robotics: Science and Systems Conf.*

Hollinger, G., Kehagias, A., and Singh, S. (2010). GSST: Anytime guaranteed search. *Autonomous Robots*, 29(1):99–118.

Hollinger, G., Kehagias, A., Singh, S., Ferguson, D., and Srinivasa, S. (2008). Anytime guaranteed search using spanning trees. Technical Report CMU-RI-TR-08-36, Robotics Institute, Carnegie Mellon Univ.

Hollinger, G., Singh, S., Djugash, J., and Kehagias, A. (2009b). Efficient multi-robot search for a moving target. *Int. J. Robotics Research*, 28(2):201–219.

Kehagias, A., Hollinger, G., and Gelastopoulos, A. (2009). Searching the nodes of a graph: Theory and algorithms. Technical Report 0905.3359v1 [cs.DM], arXiv Repository.

Kolling, A. and Carpin, S. (2008). Extracting surveillance graphs from robot maps. In *Proc. Int. Conf. Intelligent Robots and Systems*.

Kolling, A. and Carpin, S. (2009). Probabilistic graph-clear. In *Proc. IEEE Int. Conf. Robotics and Automation*.

Kolling, A. and Carpin, S. (2010). Pursuit-evasion on trees by robot teams. *IEEE Trans. Robotics*, 26:32–47.

Krause, A., McMahan, B., Guestrin, C., and Gupta, A. (2007). Selecting observations against adversarial objectives. In *Proc. Neural Information Processing Systems*.

Krause, A., McMahan, B., Guestrin, C., and Gupta, A. (2008). Robust submodular observation selection. *J. Machine Learning Research*, 9:2761–2801.

Kumar, V., Rus, D., and Singh, S. (2004). Robot and sensor networks for first responders. *Pervasive Computing*, 3(4):24–33.

Megiddo, N., Hakimi, S., Garey, M., Johnson, D., and Papadimitriou, C. (1988). The complexity of searching a graph. *J. ACM*, 35(1):18–44.

Ong, S., Png, S., Hsu, D., and Lee, W. (2009). POMDPs for robotic tasks with mixed observability. In *Proc. Robotics: Science and Systems Conf.*

Parsons, T. (1976). Pursuit-evasion in a graph. In Alavi, Y. and Lick, D., editors, *Theory and Applications of Graphs*, pages 426–441. Springer, Berlin/Heidelberg.

Roy, N., Gordon, G., and Thrun, S. (2003). Planning under uncertainty for reliable health care robotics. In *Proc. Int. Conf. Field and Service Robotics*.

Sarmiento, A., Murrieta-Cid, R., and Hutchinson, S. (2004). A multi-robot strategy for rapidly searching a polygonal environment. In *Proc. 9th Ibero-American Conf. Artificial Intelligence*.

Shewchuk, J. (2002). Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1–3):21–74.

Singh, A., Krause, A., Guestrin, C., Kaiser, W., and Batalin, M. (2007). Efficient planning of informative paths for multiple robots. In *Proc. Int. Joint Conf. Artificial Intelligence*.

Smith, T. (2007). *Probabilistic Planning for Robotic Exploration*. PhD thesis, Robotics Institute, Carnegie Mellon Univ.

Appendix: Index to Multimedia Extensions

The multimedia extensions to this article can be found online by following the hyperlinks from www.ijrr.org.

Table 3: Index to multimedia extensions

Extension	Media Type	Description
1	Video	Simulated trials
2	Video	Human-robot trials