Auton Robot (2010) 29: 99–118
DOI 10.1007/s10514-010-9189-9

GSST: anytime guaranteed search

Geoffrey Hollinger · Athanasios Kehagias · Sanjiv Singh

Received: 31 May 2009 / Accepted: 7 April 2010 / Published online: 23 April 2010

Abstract We present Guaranteed Search with Spanning Trees (GSST), an anytime algorithm for multi-robot search. The problem is as follows: clear the environment of any adversarial target using the fewest number of searchers. This problem is NP-hard on arbitrary graphs but can be solved in linear-time on trees. Our algorithm generates spanning trees of a graphical representation of the environment to guide the search. At any time, spanning tree generation can be stopped yielding the best strategy so far. The resulting strategy can be modified online if additional information becomes available. Though GSST does not have performance guarantees after its first iteration, we prove that several variations will find an optimal solution given sufficient runtime. We test GSST in simulation and on a human-robot search team using a distributed implementation. GSST quickly generates clearing schedules with as few as 50% of the searchers used by competing algorithms.

Keywords Multi-robot coordination · Graph search · Anytime algorithms · Decentralized computation · Pursuit/evasion

Electronic supplementary material The online version of this article (doi:10.1007/s10514-010-9189-9) contains supplementary material, which is available to authorized users.

G. Hollinger (✉) · S. Singh
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15217, USA
e-mail: gholling@ri.cmu.edu

S. Singh
e-mail: ssingh@ri.cmu.edu

A. Kehagias
Faculty of Engineering, Aristotle University of Thessaloniki, Box 464, Thessaloniki 54124, Greece
e-mail: kehagiat@auth.gr

1 Introduction

Suppose you are with a group in a large, complex building like a museum, supermarket, or office, and you suddenly realize that a member of the group is gone. You now need to coordinate the group to find the lost person. After searching fruitlessly for a while, you may wonder if it is possible to coordinate the group in such a way that you are guaranteed to find the lost group member. We refer to this as the *guaranteed search* problem, where searchers work together to scour the environment to ensure that they find a target if one exists. It is important to note that, depending on the complexity of the environment, a given number of searchers may be insufficient to guarantee finding the lost group member.

Guaranteed search requires the coordination of multiple agents such that a target cannot escape detection. This situation arises in at least two cases. The first is if the target is acting adversarially, and the second is if an accurate motion model of the target is unavailable. In both cases, the searchers wish to guarantee that the target will be found regardless of its movement pattern. In contrast with efficient search, which seeks to exploit a motion model to maximize capture probability (Hollinger et al. 2009b), guaranteed search makes a worst-case assumption on the target's path.

This paper examines the problem of guaranteed search in physical environments with multiple autonomous robots. The problem of coordinating a team of mobile robots to search large physical environments is relevant to many scenarios of interest in robotics. Military and first response teams often need to locate lost team members or survivors in disaster scenarios. The increasing use of search and rescue robots and mechanized infantry necessitates the development of algorithms for autonomously searching such environments. Similarly, a major application that has motivated this work is that of locating a lost first responder in an indoor environment (Kumar et al. 2004). In this application, a moving first responder is lost during disaster response, and a team of robots must locate him or her.

Optimal solutions to tightly coupled multi-robot problems typically scale exponentially with increasing robots, which makes them computationally intractable for large teams and large environments. This intractability arises in domains like guaranteed search because the result of each robot's actions depends heavily on the actions of the other robots. Thus, in many cases, we cannot determine the optimal action for a single robot without considering a large number of possible actions for the entire group. Optimally resolving such tight coupling requires considering the exponentially growing joint path space of all searchers (see Sect. 3 for a formal definition of the joint path space). Considering the joint space is an example of *explicit coordination* during which the searchers explicitly plan for their teammates.

Alternatively, if each searcher plans individually without taking into account the future actions of its teammates, the size of the search space does not increase. Since the searchers are no longer coordinating in any way, this is an instance of *no coordination*. Paths generated without any coordination often perform poorly because the searchers have no mechanism for reasoning about their teammates' actions. This limitation is particularly problematic during guaranteed search because progress may be impossible without coordination between searchers.

If searchers share their plans during planning and execution, they can use this information to improve the joint search plan. In this case, the searchers are not explicitly planning for their teammates, but they are implicitly coordinating by sharing plans. We present an algorithm that utilizes *implicit coordination* to achieve better scalability than centralized and market-based approaches.

Uninformed implicit coordination can provide poor solutions in domains like guaranteed search that require tight coordination. To improve performance for these problems, searchers can execute a shared pre-processing step, which transforms the environment representation into one solvable through implicit coordination. We show how utilizing spanning trees of the environment can yield implicitly coordinated search strategies for guaranteed search (see Fig. 1 for examples). This approach leads to an "anytime" algorithm for guaranteed search. Our anytime algorithm quickly finds a solution with potentially many searchers and then continues to generate solutions with fewer searchers over time. Our technique yields guaranteed search paths with few searchers even in very large environments.

In addition to its anytime capabilities, GSST can be distributed, and it allows for some modifications of the initial strategy to be performed during execution. Example modifications include dealing with the case of fortuitous information (e.g., an area of the environment happens to be cleared during the schedule), and correcting for poorly synchronized movement between the searchers. We demonstrate these capabilities on a team of human-robot searchers in an indoor urban search and rescue scenario.

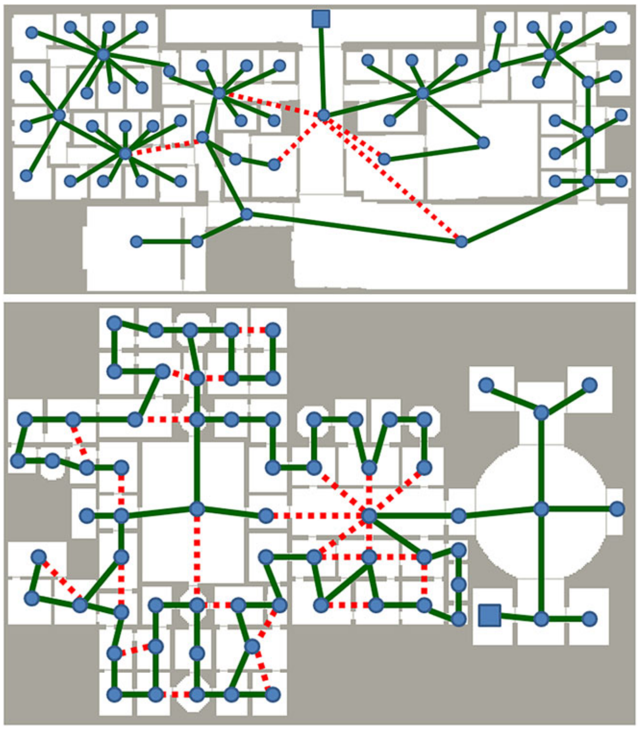


Fig. 1 Spanning trees are utilized to generate search strategies that clear a graphical representation of a physical environment of any potential adversary. The searchers utilize the optimal search strategy on the underlying tree to guide the search of the graph. Guards are used to prevent recontamination on edges not in the spanning tree. The office (*top*) admits a search strategy with three searchers, and the museum (*bottom*) admits one with five. *Solid lines* denote edges in the spanning tree, and *dashed lines* denote edges that need to be guarded during search. *The square* denotes the searchers' starting position

This paper is organized as follows. We first describe related work in graph search, pursuit-evasion, and approximation algorithms (Sect. 2). Our survey of the literature highlights the need for scalable guaranteed search algorithms, and it shows the lack of an anytime algorithm for guaranteed search. We then formally define the guaranteed search problem on both arbitrary graphs and trees (Sect. 3), and we demonstrate why uninformed implicit coordination can perform poorly when applied to guaranteed search (Sect. 4). This leads us to Sect. 5, in which we describe our anytime guaranteed search algorithm and several variants. We then prove that several variants of our algorithm are probabilistically complete, and we provide loose performance bounds relative to the number of edges in the graph. We give a simulated analysis of our proposed algorithm as well as tests on a human-robot search team (Sect. 6). Finally, we draw conclusions and discuss avenues for future work (Sect. 7).

2 Related work

Guaranteed search on graphs has a long history in both robotics and mathematics. Parsons (1976) developed some of the earliest methods for solving the adversarial pursuit-evasion problem on graphs. He considered the graph to be a system of tunnels represented by the edges of the graph in which an evader was hiding, and he defined the *search number* of a graph to be the minimum number of pursuers necessary to catch an adversarial evader with arbitrarily high speed. Determining the search number of a graph was found to be NP-hard by Megiddo et al. (1988), and later to be NP-complete due to the monotonicity of optimal edge search schedules (Bienstock and Seymour 1991; LaPaugh 1993). In this early work in pursuit-evasion, the evader can only hide in the edges of the graph, which does not fit with the intuitive representation of many environments (e.g., the rooms of a building naturally correspond to nodes, not edges of a graph). Early researchers were primarily concerned with examining the hardness of the guaranteed search problem on graphs, and they did not introduce algorithms that are scalable to large teams in realistic environments.

Recent work in graph search discusses several interesting variations of the guaranteed search problem. The traditional formulation does not restrict the movement of searchers. In other words, searchers are allowed to “teleport” between nodes in the graph without following the edges between them. This enables searchers to clear disjoint parts of the graph without maintaining a route to a starting node. Barrière et al. (2003) introduced the idea of connected search during which searchers must maintain a connected subgraph of cleared nodes. Connected search guarantees that a path exists to the starting nodes at all times and that searchers are connected by a cleared or “safe” region of the graph. Barrière et al. argue that this is an important quality for search strategies in the computer network domain.

Connectedness is also an important characteristic of guaranteed search strategies in the physical world. Real robots cannot teleport between nodes in the graph because these nodes represent physical locations. Instead, robots must restrict their search paths to those traversable in the environment. Furthermore, domains like urban search and rescue and military reconnaissance require a safe path back to the starting point to aid in evacuation. This motivates the examination of connected search paths during guaranteed search. However, there is a “price of connectedness” because clearing a graph with a connected search strategy may require more searchers than with a disconnected one (Fomin et al. 2004).

The fact that finding the search number of a graph is NP-hard suggests that exact solutions to the guaranteed search problem on large graphs are intractable (unless $P = NP$). This has motivated researchers to develop guaranteed search algorithms on special cases of graphs. Barrière et al. (2002) showed that a connected search strategy with minimum number of searchers can be found in linear-time on trees. Polynomial-time algorithms for searching hypercubes (Flocchini et al. 2008), tori, and chordal rings (Flocchini et al. 2007) have also been developed. Unfortunately, most environments in the physical world cannot be represented by these special cases.

Since many applications require guaranteed search in arbitrary environments, prior work has proposed some heuristic algorithms that provide good performance. Flocchini et al. (2005) examined a genetic algorithm approach for clearing arbitrary graphs. Their approach does not take into account prior information about the environment, and it does not allow for extensive coordination between searchers. Thus, it can require many searchers in fairly simple environments.

Kolling and Carpin (2010) showed how a polynomial-time graph cut algorithm can be applied to a weighted graph formulation of the multi-robot surveillance domain. Their algorithm operates on an alternative formulation of the clearing problem, which requires several searchers to clear areas in the environment. They consider the optimal traversal of the minimum spanning tree to help generate a search schedule, but they do not examine alternative spanning trees or traversal strategies.

We have shown that a dynamic programming inspired algorithm, which iteratively maximizes the number of cleared nodes, can work well on complex graphs (Kehagias et al. 2009b). However, this algorithm is unable to find a search strategy with the minimal number of searchers on many complex graphs (see Sect. 6). On the other hand, this algorithm can produce nonmonotonic searches and can deal with both finite evader speed and non-local visibility.

A key insight in the development of approximate guaranteed search algorithms is the connection between graph search and the graph parameters of treewidth and pathwidth (Dendris et al. 1994). A tree decomposition of a graph is a new graph, which (a) is a tree; (b) has nodes which correspond to *sets of nodes* of the original graph (these new “supernodes” are called *bags*); (c) satisfies some additional technical conditions (listed in Dendris et al. 1994). The *width* of a tree decomposition is the cardinality of its largest bag minus one. The *treewidth* of a graph G is the minimum of the widths of all tree decompositions of G. A minimum width tree decomposition of G yields an optimal clearing schedule for the *visible* search problem (i.e., when the searchers know the location of the target).[1] Similarly, a path decomposition of a graph G is a tree decomposition where the tree is also a path; the *pathwidth* of G is the minimum of the widths of all path decompositions of G; a minimum width path decomposition provides an optimal solution to the guaranteed *invisible* search problem. Approximation algorithms for treewidth and pathwidth have been proposed (Kloks 1994), but they are not guaranteed to provide connected or internal path decompositions.

Fraigniaud and Nisse (2006) proposed an algorithm for connected search by finding approximately minimal width tree decompositions. They show how these decompositions can be used to find connected search strategies. Though polynomial-time, their algorithm grows in complexity with both the search number and the size of the graph. In addition, the approximation bound degrades fairly quickly with the size of the graph. They do not implement their algorithm, and they do not provide sufficient information to demonstrate that an implementation would be efficient or feasible.

Much of the above work examines the "edge search" problem in which the evader hides in the edges of the graph. A more intuitive representation for physical environments is for the nodes to correspond to rooms and hallways and for the edges to correspond to doorways or borders between the rooms. In the resulting formulation, the evader hides in the nodes of the graph. We discuss many of the theoretical properties of node search in Kehagias et al. (2009a), and we show its formal relationship to edge search.[2]

With robotic applications in mind, Guibas and LaValle extended guaranteed search techniques to polygonal environments (LaValle et al. 1997; Guibas et al. 1999; LaValle 2006). Their algorithm discretizes polygonal environments into conservative visibility regions and then uses an information space approach to guarantee capture. For a single pursuer, their approach is guaranteed to find a solution if one exists. To maintain completeness with multiple searchers, their algorithm would need to generate a potentially exponential number of visibility regions and search the resulting information space. This is only tractable for few searchers and small environments. An alternative is to use an iterative visibility-based method, but such an approach loses completeness and becomes similar to uninformed implicit coordination (see Sect. 4).

In addition, Isler et al. (2005) showed that randomized strategies can lead to arbitrarily high probability of capture in simply connected polygons and trees even with a single pursuer. They relax the assumption that the evader is fully aware of the pursuer's plan, and they allow the pursuer to make unpredictable random movements. However, unlike the approaches discussed above, their randomized approach does not provide a definitive time at which the environment is *cleared* (i.e., it cannot contain an evader).

Many of the methods mentioned above are not scalable to large teams of searchers. To improve scalability, robotics researchers have applied auction methods to multi-agent coordinated search domains. Kalra (2006) presented Hoplites, an algorithm that utilizes auction-based plan sharing to perform tightly coupled tasks. Hoplites allows searchers to actively coordinate by running auctions when they are presented with high-cost situations, and it provides a framework for incorporating team constraints, which she demonstrates in the constrained search domain. Hoplites depends on multi-robot auctions to generate good search plans, but it does not provide a mechanism for deciding when to hold an auction if it is not obvious. Setting a synthetic threshold is one option, but this leads to poor performance if the threshold is set incorrectly.

Gerkey et al. (2005) also developed a parallel stochastic hill-climbing method for small teams (PARISH) that is closely related to auction-based methods. Rather than using the market metaphor, they frame guaranteed search as a parallel optimization problem. Their algorithm dynamically forms teams of searchers that work together to solve tasks. Team formation and path generation are guided by a heuristic, which makes their algorithm's performance sensitive to the choice of heuristic. Regardless of the heuristic used, the algorithm requires explicit coordination within teams, which can lead to high computation in large environments. We show in Sect. 6 that our algorithm clears several test environments with fewer searchers than PARISH.

While auction-based algorithms are more scalable than coupled planning approaches, they still rely on auctions and/or team formation, which can consume large amounts of communication bandwidth and planning time. We demonstrate that implicit coordination with the addition of an informed pre-processing step allows for fast solutions to guaranteed search problems. Our technique allows for distributed and online operation, which is also typically found in auction-based techniques.

Guaranteed search algorithms in the literature do not demonstrate "anytime" capabilities. An anytime algorithm is one that quickly generates an initial solution and then improves solution quality with increasing runtime. Anytime algorithms have been successfully applied to POMDP planning (Smith 2007), dynamic path planning (Likhachev et al. 2005), and many other domains. These state-of-the-art algorithms allow for high-quality solutions with varying levels of computational resources. Zilberstein (1996) discusses some

[1]Note that finding a minimal width tree decomposition of an arbitrary graph G also is an NP-hard problem.

[2]Note that several alternative versions of "node search" appear in the literature (see Alspach 2006 and Fomin and Thilikos 2008 for surveys). In one formulation, the evader resides in the edges of the graph, 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. We do not consider these variants in this paper.

of the desirable qualities of anytime algorithms in intelligent systems. Our proposed algorithm (GSST) shows many of these qualities, including monotonicity (the solution only improves over time), recognizable quality (the quality of the solution, i.e. number of searchers, can be determined at runtime), consistency (the algorithm will not spend too much time finding a single solution), and interruptibility. Expanding GSST to incorporate other desirable qualities of anytime algorithms is discussed in Sect. 7. To the best of our knowledge, GSST is the first algorithm to bring the advantages of anytime algorithms to the domain of guaranteed search.

3 Problem setup

We now formally define the guaranteed search problem on graphs. Let $G = (N, E)$ be the undirected environment graph with vertices N and edges E. At any time t, the k-th of K searchers exists on vertex $s_k(t) = u \in N$. The searcher's movement is deterministically controlled, and each may travel to vertex $s_k(t+1) = v$ if there exists an edge between u and v. A target also exists on this graph on vertex $e(t) = u \in N$. The target moves along edges between vertices. A searcher "captures" the target by moving onto the same vertex (i.e., $\exists k, t : s_k(t) = e(t)$). In this paper, we assume that a searcher on a given node will always detect a target on the same node, and the target is assumed to have potentially unbounded speed.

At any time during the search, there are nodes that may contain the target (dirty nodes) and nodes that may not (cleared nodes). We denote the set of dirty nodes at time t as $N_D(t) \subseteq N$ and the cleared set as $N_C(t) = N \setminus N_D(t)$. The searchers' goal is to progressively decrease the set of dirty nodes to the empty set, thus guaranteeing capture of any target in the environment.

The guaranteed search optimization problem is to generate a clearing strategy for a given graph requiring the minimum number of searchers. Let $S(t) = \{s_1(t), \ldots, s_K(t)\}$ (the set of locations of each searcher at time t), let $\mathbf{S} = S(0), \ldots, S(t_f)$ (the full search schedule), let $\overline{sn}(\mathbf{S}) = K$ (the number of searchers required for the schedule), and let $N_D(t|\mathbf{S})$ be the dirty set at time t given $\mathbf{S}$. Finally let $\Psi^F(t_f)$ be the set of feasible searcher paths on the search graph from time $t = 0, \ldots, t_f$.[3] The optimization problem is to generate a feasible schedule $\mathbf{S}$ with minimal searchers K for which $N_D(t_f|\mathbf{S}) = \emptyset$ as shown in (1). In this paper, we assume that the variable t_f is left unspecified and can be modified as part of the optimization. Note that (1) can (and likely does) have many possible optimal solutions

$$\mathbf{S}^* = \operatorname*{argmin}_{t_f, \mathbf{S}} \overline{sn}(\mathbf{S}) \quad \text{s.t.} \quad N_D(t_f|\mathbf{S}) = \emptyset. \tag{1}$$

Equation (1) does not directly consider minimizing the length of the schedule t_f (i.e., the *clearing time* of the schedule). However, if many schedules are generated with a minimal number of searchers, the shortest schedule can easily be chosen. In addition, our ongoing work shows that a variation of GSST can be used to minimize clearing time given a fixed number of searchers (Hollinger et al. 2009a).

We examine the guaranteed search problem on graphs with the understanding that both the target and searchers exist on the vertices of the graph. We refer to this problem as node search, which is different from the edge search problem discussed in the literature (i.e., the target exists in the edges of the graph) (Parsons 1976). Searching built environments lends itself to the node search formulation because rooms and hallways can be easily decomposed into nodes on the graph. The edge search formulation, on the other hand, describes a situation in which the edges on the graph are contaminated (e.g., with poison gas) or the search is performed in a system of tunnels. We deal with the node search formulation because of its direct connection with indoor searching. In our ongoing work, we have shown that any edge clearing schedule is also a node clearing schedule Kehagias et al. (2009a), which allows us to take advantage of algorithms from the edge search literature to find node clearing schedules.

It is important to note that we only consider search strategies that are internal, monotone, and connected.[4]

Definition 1 A node search schedule $\mathbf{S}$ is internal, monotone, connected (IMC), and rooted if the following hold.

1. *Internal*: once placed on the graph, searchers can only move along the edges, and searchers are never removed from the graph.
2. *Monotone*: for all t, we have $N_C(t-1) \subseteq N_C(t)$.
3. *Connected*: for all t, $(N_C(t), E_C(t))$ is a connected subgraph of G.
4. *Rooted*: searchers only can be placed onto a single, prespecified node called the *root* of the search.

Internal search strategies restrict the movement of the searchers to those feasible on the graph (i.e., searchers cannot teleport), and connected search strategies always maintain a connected subgraph of cleared nodes. Both of these

[3]Formally, the joint path space (or set of possible paths) for a graph $G = (N, E)$ considers the possible locations $s_k(t) \in N$ of K searchers at times $t \in \{0, 1, \ldots, t_f\}$. The searchers' configuration space at time t is $\Phi(t) = \{(s_1(t), \ldots, s_K(t)) | s_1(t) \in N, \ldots, s_K(t) \in N\}$. The searchers' joint path space is defined as the Cartesian product $\Psi(t_f) = \Phi(0) \times \cdots \times \Phi(t_f)$.

[4]Throughout this paper we will refer to a *minimal* search strategy as one that requires the smallest possible number of searchers while remaining internal, monotone, and connected. The *attained minimal* search strategy is the best strategy found thus far.

characteristics are desirable in robotic search applications. The movements of robots in the real world are restricted to those that are physically possible. In addition, robot teams will often start in the same location, thus allowing the cleared regions to grow as a connected subgraph from that location.

3.1 Environment discretization

The application of GSST to physical environments requires discretization into a number of cells. We accomplish this by dividing the environment into a convex tessellation. The convexity of the cells guarantees that a searcher in a given cell will have line-of-sight to a target in the same cell. Gaining line-of-sight is relevant to most sensors that a mobile robot would carry including cameras and laser scanners.

Sufficiently small convex cells allow for clearing schedules with only a 180 degree field-of-view sensor. The requirements are that (1) a searcher can see an entire cell while at a boundary (pointing inward) and (2) the searcher can move from one cell border to another while keeping the entire destination border in view at all times (Gerkey et al. 2005). These requirements are met with small convex cells, and we implement several 180 degree FOV schedules in Sect. 6.

One method for discretization is to take advantage of the inherent characteristics of indoor environments. To discretize an indoor map by hand, simply label convex hallways and rooms as cells and arbitrarily collapse overlapping sections. Taking into account the cell adjacency in a discretized map yields an undirected graph that the searchers can traverse. Figure 2 shows two example floorplans used in our experiments. Ongoing work by Kolling and Carpin (2008) has attempted to automate the generation of this type of discretization.

Alternatively, a suitable discretization can be found automatically by generating a constrained Delaunay triangulation of the environment (Shewchuk 2002). The constrained Delaunay triangulation ensures that the edges of the obstacles correspond to edges in the triangulation, which generates a convex tessellation of the free space. The resulting search graph has a bounded branching factor of three because it consists only of triangles. Delaunay triangulations can be generated with angle and size constraints on the polygons, which allows for the inclusion of range constraints on the sensors. If a triangular tessellation is not suitable, any convex tessellation can be used, but we note that the relationship between the discretization and the number of searchers required is not well understood (see Sect. 7 for some avenues for future work in this area).

In indoor environments, there are several different categories of "obstacles" that may exist. In this paper, we include major obstacles (e.g., walls and buildings), but we do

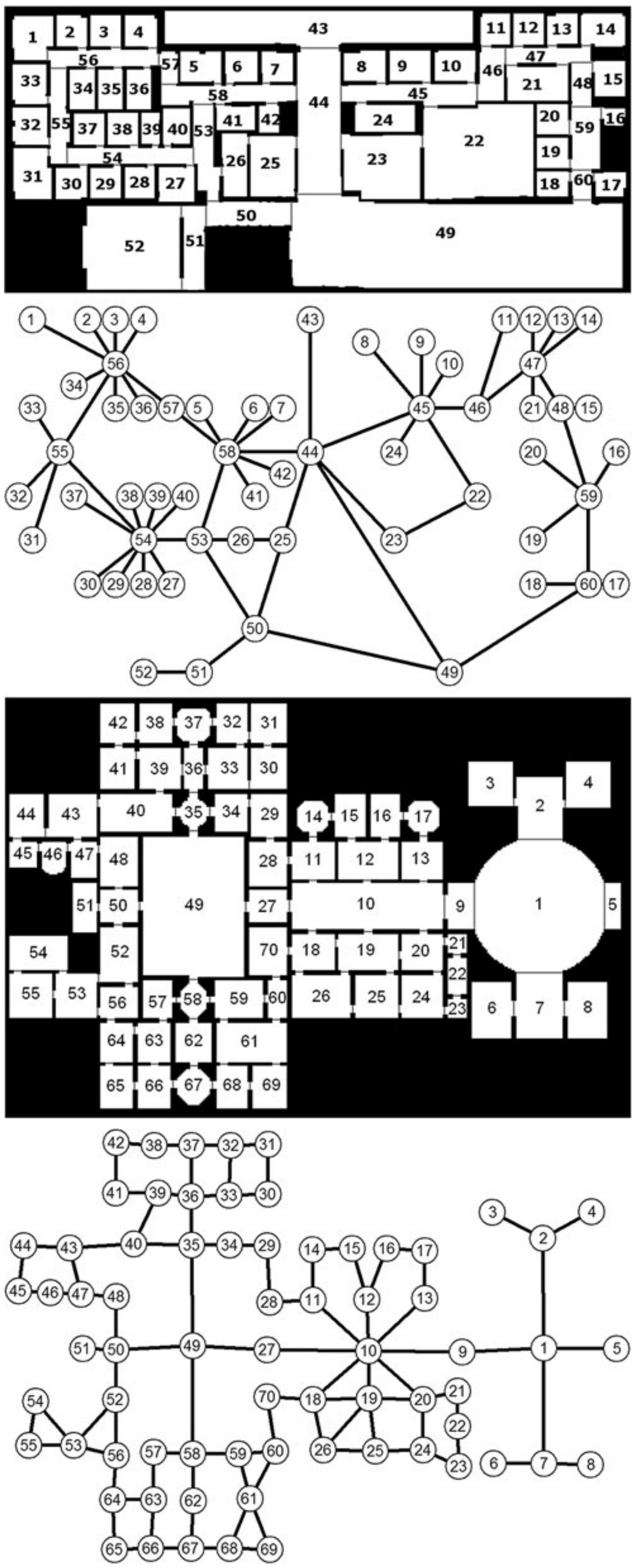


Fig. 2 Example floorplans and graphical representation of environments used for guaranteed search trials. The Newell-Simon Hall (*top*) and National Gallery of Art (*bottom*) were used for simulated testing

not consider minor obstacles, such as small furniture. We assume that the robots' sensors can see through minor obstacles, or that the adversary is large enough that it could not hide behind these obstacles. Of course, a finer discretization taking into account minor obstacles can also be obtained and used with GSST, at the possible cost of using more searchers. The types of obstacles to consider during the discretization in various environments is a design decision and a subject of ongoing research.

4 Uninformed implicit coordination

The multi-robot guaranteed search problem is difficult to solve due to the tight coupling of the searchers in the joint path space and the necessity to search a large number of paths to find an optimal strategy. If an environment graph has a branching factor b and requires K searchers t_f steps to clear, the number of possible paths is $O(b^{Kt_f})$. As the environment size increases, both the number of searchers required and the time to clear will increase. This quickly leads to the intractability of any exhaustive search of the joint space.

In prior work, we presented FHPE + SA, an approximation algorithm for solving the efficient search path planning problem (Hollinger et al. 2009b). In this domain, searchers must maximize the probability of finding a non-adversarial target. FHPE + SA has searchers plan paths by enumerating all possible strategies to a finite-horizon. The robots do so greedily and sequentially, which allows for linear scalability in the number of searchers. Planning sequentially and sharing paths is a form of implicit coordination because the robots do not explicitly plan for their teammates.

We can extend this approximation algorithm to guaranteed search with a simple modification. During finite-horizon planning, the searchers can limit their paths to those that do not cause recontamination (i.e., paths that do not allow clean nodes to become dirty). If the searchers plan sequentially and share their paths to construct a partial schedule incrementally, this leads to a piggyback effect during which each successive searcher extends the clearing schedule further in the environment. By iteratively planning on the receding horizon, the searchers can find a clearing schedule in this manner.

Algorithm 1 gives a description of this receding horizon algorithm, which we refer to as Receding Horizon Greedy Clearing (RHGC). The parameter d represents the horizon length in the receding-horizon planner. The running time of RHGC scales linearly with the number of searchers K but exponentially with increasing d: $O(Kb^d)$, where b is the branching factor of the search graph. Thus, d should be set to a fairly small value, particularly if the branching factor is large.

Algorithm 1 Receding Horizon Greedy Clearing (RHGC) for guaranteed search

```
1:  Input: Graph $G = (N, E)$, Start node $b \in N$
2:  $K \leftarrow 1$, $t \leftarrow 0$, $s_1(0) \leftarrow b$
3:  **while** $N_D \neq \emptyset$ **do**
4:      **for** searcher $k = 1$ to $K$ **do**
5:          % $P$ is a feasible path for searcher $k$ in the search graph to horizon $d$
6:          % $\mathbf{S} = s_1(0), \dots, s_K(0), \dots, s_1(t+d), \dots, s_{k-1}(t+d)$ is a partial schedule
7:          % $N_D(t+d|\mathbf{S})$ is the dirty set at time $t+d$ given $\mathbf{S}$
8:          $s_k(t+1), \dots, s_k(t+d) \leftarrow \operatorname{argmin}_P |N_D(t+d|P, \mathbf{S})|$
9:      **end for**
10:     $t \leftarrow t + 1$
11:     **if** no searcher can move without recontamination **then**
12:         % Add a new searcher and reset to start
13:         $K \leftarrow K + 1$, $t \leftarrow 0$, $s_k(0) \leftarrow b$ for all searchers
14:     **end if**
15: **end while**
16: Output: Search strategy $\mathbf{S}$, clearing time $t$
```

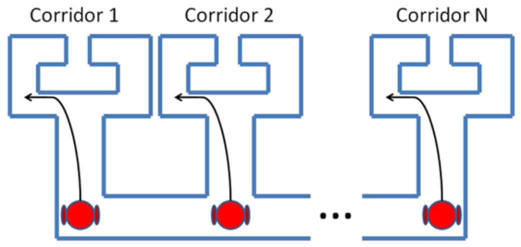


Fig. 3 Example of an environment in which uninformed implicit coordination leads to a large number of searchers. If each searcher maximizes its own path on the finite-horizon, each will choose to go down a different corridor, requiring $N + 1$ searchers to clear the environment. A minimal clearing strategy requires only three searchers regardless of the number of corridors: one searcher guards the main hallway, and two searchers clear the corridors together. In contrast, if the environment is discretized into a tree, a three-searcher solution can be found easily with Algorithm 3

Algorithm 1 is an application of linearly scalable implicit coordination to the guaranteed search domain. Unfortunately, uninformed implicit coordination can perform poorly because it ignores the requirement for tight coordination. In other words, the searchers sometimes must work together to make any progress in clearing the environment. Figure 3 gives an example where uniformed implicit coordination leads to a large number of searchers. We also compare RHGC to GSST in Sect. 6.

Given the potentially poor performance of implicit coordination during guaranteed search, it may be tempting to utilize market-based techniques or other methods of injecting explicit coordination into the search schedule. GSST pro-

Algorithm 2 Edge labeling for trees

```
1: Input: Tree $T = (N, E)$, Start node $b \in N$
2: $O \leftarrow$ leafs of $T$
3: while $O \neq \emptyset$ do
4:   $l \leftarrow$ any node in $O$, $O \leftarrow O \setminus l$
5:   if $l$ is a leaf then
6:     $e \leftarrow$ only edge of $l$
7:     $\lambda(e) = 1$
8:   else if $l$ has exactly one unlabeled edge then
9:     $e \leftarrow$ unlabeled edge of $l$
10:    $e_1, \dots, e_d \leftarrow$ labeled edges of $l$
11:    $\lambda_m \leftarrow \max\{\lambda(e_1), \dots, \lambda(e_d)\}$
12:    if multiple edges of $l$ have label $\lambda_m$ then
13:      $\lambda(e) \leftarrow \lambda_m + 1$
14:    else
15:      $\lambda(e) \leftarrow \lambda_m$
16:    end if
17:  end if
18:  if $l \neq b$ and $parent(l)$ has exactly one unlabeled edge
     then
19:    $O \leftarrow O \cup parent(l)$
20:  end if
21: end while
22: Output: Edge labeling $\lambda(E)$
```

Algorithm 3 Guaranteed search schedule for trees

```
1: Input: Tree $T = (N, E')$, Edge labeling $\lambda : E' \to Z$,
   Start node $b \in N$
2: % Define $s_k(t)$ as the node occupied by the $k \leq \mu$
   searcher at time $t$
3: Calculate required searchers, $\mu =$ edge label into $b$
4: $t \leftarrow 0$, $K \leftarrow \mu$, $s_k(t) \leftarrow b$ for all $k$, $N_D \leftarrow N \setminus b$
5: while $N_D \neq \emptyset$ do
6:   $t \leftarrow t + 1$
7:   for all searchers $k$ do
8:     if searcher cannot move without recontamination
       then
9:       $s_k(t) \leftarrow s_k(t-1)$
10:    else if positive edge label exists incident to
       $s_k(t-1)$ then
11:      $s_k(t) \leftarrow$ node reached through *lowest* labeled
         edge
12:      Decrement $\lambda(e)$ of edge traversed
13:    else
14:      $s_k(t) \leftarrow$ node reached through negative labeled
         edge
15:      Increment $\lambda(e)$ of edge traversed
16:    end if
17:    $N_D \leftarrow N_D \setminus s_k(t)$
18:  end for
19: end while
20: Output: Search schedule $\mathbf{S}$, clearing time $t$
```

vides an alternative method for improving the performance of implicit coordination by exploiting the relatively easy problem of finding a guaranteed search schedule on trees.

5 The GSST algorithm

In this section, we present GSST, an anytime algorithm for generating guaranteed node search schedules on graphs by utilizing their spanning trees. Our algorithm takes advantage of the minimal edge search on the underlying spanning tree to guide node search on the full graph. Though GSST does not have a performance guarantee with respect to the optimal solution, we present several traversal variants that are probabilistically complete (i.e., they have a nonzero probability of producing an optimal search schedule at each iteration). GSST and its variants allow for scalable guaranteed search with long term planning and coordination.

5.1 Guaranteed search on trees

Guaranteed search on arbitrary graphs is an NP-hard problem, but linear-time algorithms exist for guaranteed search on trees. This section describes a non-recursive version of the algorithm from Barrière et al. (2002). This algorithm generates a minimal strategy for a given tree in time linear in the nodes in the search graph. Eliminating the need for recursion allows for the direct application of implicit coordination (see Sect. 6).

We assume that the starting node of the searchers is known and the same for all searchers. Denote this starting location as $b \in N$. First, label the edges on the tree $T = (N, E)$ with $\lambda : E \to Z^+$ as in Algorithm 2. The mapping $\lambda(e)$ describes the number of searchers that must move down the tree along that edge during the search strategy.

Now, make the edges directional by pointing them down the tree from the start node to the leaves. Double the edges and give these new edges opposite direction. Label the doubled edges with $\lambda(e_2) = -\lambda(e)$, where e_2 is the double of edge e. The negative values represent recursive steps back up the tree after clearing. Refer to the set of edges and their doubles as E'.

A minimal edge search strategy can be generated from this labeling in a distributed manner. Algorithm 3 uses a variation of the recursive algorithm in prior work to clear the edges of a tree T rooted at node b with the minimum number of searchers (Barrière et al. 2002). The number of searchers necessary is equivalent to the edge labeling of an edge entering the start node. Refer to this value as μ. Since this schedule is an edge clearing schedule, it is also a node clearing schedule of the underlying tree (Kehagias et al. 2009a).

Algorithm 4 Randomized depth-first spanning tree generation

```
1: Input: Graph $G = (N, E)$, Start node $b \in N$
2: $V \leftarrow \emptyset$, $S \leftarrow \emptyset$, $B \leftarrow \emptyset$, $x \leftarrow b$
3: while some edges are in neither $S$ nor $B$ do
4:   $V \leftarrow V \cup x$, $R \leftarrow \emptyset$
5:   for all nodes $y$ adjacent to node $x$ do
6:     if $y \in V$ then
7:       $B \leftarrow B \cup e(x, y)$
8:     else if $y$ not already visited from $x$ then
9:       $R \leftarrow R \cup y$
10:    end if
11:  end for
12:  if $R$ is empty then
13:    $x \leftarrow$ parent of $x$
14:  else
15:    Choose random node $z \in R$
16:    Set parent of $z$ to $x$
17:    $S \leftarrow S \cup e(x, z)$, $x \leftarrow z$
18:  end if
19: end while
20: Output: Set of tree edges $S$, Set of non-tree edges $B$
```

5.2 Generating spanning trees

The algorithm described above for trees does not apply to arbitrary graphs because the edge labeling is not possible with cycles. However, additional searchers can be used as guards to transform an arbitrary graph $G = (N, E)$ into a tree $T = (N, S)$. The non-guard searchers can then traverse the resulting tree using the algorithm described above. This reduces the guaranteed search problem to that of generating a "good" spanning tree on which to base the search.

The problem of uniformly sampling the space of spanning trees has been heavily studied. Wilson's algorithm based on the use of loop-erased random walks is both efficient and conceptually simple (Wilson 1996). As an alternative, we propose Algorithm 4, which uses a randomized depth-first search to find a spanning tree. This algorithm focuses the search on trees that have non-tree edges incident to few nodes. This intuitively leads to trees that require fewer guards. We compare these methods for spanning tree generation in Sect. 6.

5.3 Labeled traversal

Given a set of tree edges S and a set of non-tree edges B, a naive search strategy can be found by assigning guards to a node incident to each non-tree edge and then searching the tree as in Algorithm 3. This technique ignores two important characteristics of the problem. First, adding a guard for every non-tree edge will likely be redundant. If several non-tree edges are incident to a single node, one guard will suffice for all of them. Second, the search strategy occurs over a time interval. Guards that are necessary at earlier times may be free to guard other edges at later times.

Algorithm 5 Guaranteed Search with Labeled Traversal

```
1: Input: Graph $G = (N, E)$, Start node $b$
2: while time is available do
3:   Find a spanning tree $T = (N, S)$
4:   Label edges of $T$ using Algorithm 2
5:   Calculate required searchers, $\mu$ = edge label into $b$
6:   Generate $\mu$ tree searchers
7:   while graph is not cleared do
8:     Move tree searchers according to Algorithm 3
9:     if a tree searcher reaches a node $c$ with incident
       non-tree edge then
10:      if guard can move without recontamination
         then
11:        Move guard to node $c$
12:      else
13:        Generate new guard and move to node $c$
14:        Increment number of guards
15:      end if
16:    end if
17:  end while
18:  Record $\eta$ = number of tree searchers plus guards
19: end while
20: Output: Search strategy with lowest $\eta$
```

Algorithm 5 shows how these observations can be taken into consideration during search to generate a traversal strategy. We will refer to the resulting traversal as labeled traversal (GSST-L). Algorithm 5 also demonstrates the anytime nature of GSST. As more spanning trees are generated, the lowest attained search number is stored. More strategies are generated with increasing computation, which leads to lower attained search numbers.

Note that Algorithm 5 can be modified for more conservative use of guards. Instead of calling for a guard whenever a tree searcher reaches a node with incident non-tree edges, the algorithm can wait for all tree searchers to reach such nodes. This ensures that moving tree searchers will not release a previously stuck searcher. Furthermore, the algorithm can reassign tree searchers that are no longer necessary. For instance, three searchers may be needed to clear an early portion of the graph, but the remaining subgraph may only require two. In this case, a tree searcher can be reassigned as a guard after it is no longer needed as a tree searcher. These reassignments may affect search number because they can provide extra guards later in the search schedule. These two extensions are used in Sect. 6.

The guard allocation step can be seen as an instance of implicit coordination. Each searcher determines where it can best assist the search schedule and then broadcasts that information to the other searchers. The tree searchers do not

Algorithm 6 Randomized traversal

```
1:  Input: Graph $G$
2:  Generate a random spanning tree of $G$
3:  Initialize the cleared set $N_C$ as the root node
4:  Generate a searcher at the root
5:  while graph not cleared do
6:      Randomly choose edge $e$ in the spanning tree and adjacent to $N_C$
7:      if a searcher can traverse $e$ without recontamination then
8:          Move to and traverse $e$ with that searcher, update $N_C$
9:      else
10:         Generate a searcher at the root
11:     end if
12: end while
13: Output: search schedule $\mathbf{S}$
```

explicitly coordinate with the guards to cover the non-tree edges. Instead, the searchers utilize the shared spanning tree representation to help determine their task assignments. The generation of a spanning tree is a shared pre-processing step, which occurs before implicit coordination.

5.4 Randomized and label-weighted traversal

We now consider several traversal variants for GSST that utilize the labeling in different ways. Algorithm 6 shows a randomized traversal strategy (GSST-R). The randomized strategy sends searchers down branches of the spanning tree without considering the tree labeling. Unlike labeled traversal, randomized traversal does not utilize information from optimal search on the spanning tree.

A simple modification of random traversal is to weight the edge selection based on their labeling. We will refer to this as label-weighted traversal (GSST-LW). We prove that these variations of GSST are probabilistically complete (see Theorem 2 below).

5.5 Label-dominated traversal

Another alternative is to traverse labeled edges that lead to branches with known clearing numbers. This can be done by labeling edges that lead to parts of the graph that are trees (subtrees of the graph). A list of searchers who can move without recontamination can be maintained during search. If an edge adjacent to N_C leads to a subtree of the graph, and enough free searchers are available, clearing this subtree can only improve the search strategy. The additional clearing moves can be determined randomly or using a label-weighted approach. We refer to this as label-dominated traversal (GSST-LD).

5.6 Correctness

Theorem 1 proves that all variations of GSST are guaranteed to generate a node clearing schedule with a finite number of searchers given a spanning tree. From this, we see that every iteration of the anytime algorithm will generate a clearing strategy with a finite number of searchers.

Theorem 1 *For every graph G and spanning tree T, all presented variations of GSST generate a node clearing strategy with a finite number of searchers.*

Proof Any spanning tree T of G contains all nodes in G. Also, any edge clearing strategy of T is also a node clearing strategy of T (Kehagias et al. 2009a). Thus, GSST-L will generate a clearing schedule of G if sufficient guards are added to prevent recontamination. Since the number of guards cannot be larger than the number of non-tree edges on the graph, the strategy uses finitely many searchers.

GSST-R, GSST-LW, and GSST-LD start a single searcher at the root and progressively makes clearing moves with searchers that can move without recontamination. Searchers are added when non-contaminating clearing moves are impossible. Thus, progress is made at each step, and recontamination does not occur, which leads to a node clearing schedule. $\square$

5.7 Completeness

Theorem 2 shows that random and label-weighted traversal have a nonzero chance to find a minimal monotone, connected node clearing schedule on any graph. In other words, every iteration of the anytime algorithm has a chance to generate a minimal search schedule on the graph. Thus, if the algorithm were run for a sufficiently long time, it would find a minimal schedule on any graph.

Theorem 2 *At each iteration, both random traversal (GSST-R) and label-weighted random traversal (GSST-LW) of a uniformly generated spanning tree have a nonzero chance of yielding a minimal monotone/connected node search strategy on any graph.*

The proof of Theorem 2 is given in the Appendix B. We conjecture that the same holds for label-dominated random traversal, but we do not provide a formal proof. The completeness of labeled traversal is an open problem, which we discuss in Sect. 7. In addition, our proofs assume that a spanning tree generation strategy is employed that has a nonzero chance of producing any spanning tree. Uniform sampling has this property, but randomized depth-first sampling (Algorithm 4) does not. Thus, GSST may not be probabilistically complete when used in conjunction with depth-first sampling.

5.8 Solution quality

We now give theoretical analysis regarding the performance of GSST. This analysis is limited to labeled traversal. The performance of the search schedules generated by GSST is determined by the number of total searchers (guards plus tree searchers) required to clear the graph. Barrière et al. (2002) show that the worst-case bound on trees is $\mu \leq \log_2(|N|)$. This is an upper bound on the number of tree searchers needed for a graph with $|N|$ nodes. Labeled traversal requires both tree searchers and guards. As described above, the maximum number of guards needed is $|B| = |E| - |N| + 1$. Thus, the worst-case total number of searchers η for the algorithm is $\eta \leq \log_2(|N|) + |E| - |N| + 1$.

The maximum number of searchers describe above is a worst-case bound on the performance of the first spanning tree generated for a graph. Randomly searching over the space of spanning trees and using temporal aspects of the search to reuse guards significantly reduces this number. It may be possible to construct a traversal strategy such that this worst-case is bounded relative to optimal, and we leave this as an avenue for future work.

This analysis highlights the anytime nature of GSST. The searchers begin the initial step of generating spanning trees. If an acceptable strategy has been found or time has run out, the best spanning tree is utilized to perform the search. At any time during tree generation, a clearing schedule is available, though perhaps one requiring a large number of searchers.

5.9 Computational complexity

Each iteration of GSST performs the following steps: (1) generate a spanning tree, (2) label the spanning tree, (3) determine the attained search number, and (4) generate the traversal. We show that the first three steps can be performed in time linear in the number of nodes in the graph.

Finding a spanning tree using depth-first search requires visiting each node once to label its edges and is thus $O(|N|)$. We direct the reader to Wilson (1996) for a discussion of the complexity of uniform spanning tree generation.

As described by Barrière et al. (2002), edge labeling and determining search schedules on trees can be done in linear-time. Our non-recursive labeling (Algorithm 2) also is linear in the number of nodes, since it visits each node once.

Our labeled tree traversal algorithm (Algorithm 3) with K searchers is $O(K|N|)$ since it requires an inner loop that determines the schedule for each individual searcher for $|N|$ clearing moves.[5] The worst-case number of searchers to clear a tree is $O(\log|N|)$ (see above), which leads to $O(|N|\log|N|)$ computation. This has the advantage that it allows the computation to be distributed among the searchers.

[5]We note that a clearing schedule on a tree may require up to $2|N|$ moves since the searchers may need to recurse up the tree.

Replacing the inner loop with a centralized planner that moves multiple searchers during each iteration reduces the computation to $O(|N|)$. This is equivalent to the recursive planner used by Barrière et al. (2002) on trees, except that it also counts guard placements. The randomized traversal variants (e.g., Algorithm 6) also are $O(|N|)$ because they must make $|N|$ random choices of frontier nodes to generate a clearing schedule.

The output of all planners described above may contain teleporting guard moves, which will require further processing to make the schedule internal. For labeled traversal of arbitrary graphs, the search schedules of guards can be determined using Dijkstra's algorithm between guard points, which is $O(|E| + |N|\log|N|)$ per guard. A feasible guard schedule is always possible because the search schedule is monotone and connected. Note that this step only needs to be performed once when a schedule with sufficiently few searchers is found and ready to be executed.

In some cases, there may be a limited number, $K_{\max}$, of available searchers, and strategies requiring more than $K_{\max}$ searchers may be considered infeasible. In these cases, the determination of the attained search number can be stopped when $K_{\text{curr}} > K_{\max}$, and a new spanning tree can be generated at this time. Additionally, $K_{\max}$ can be set to the best solution so far, which will throw out any schedule as soon as it generates more than the current attained minimal. These modifications save computation and lead to finding a desirable strategy more quickly.

GSST is limited by the number of iterations (i.e., number of spanning trees) needed to find a schedule with sufficiently few searchers. For large graphs, an exponential number of spanning trees are possible. Our algorithm leverages the fact that many spanning trees will yield good search schedules (though not necessarily minimal ones).

5.10 Example

The following example demonstrates the importance of the choice of underlying spanning tree when using GSST. For explanatory purposes, we use labeled traversal for the example. Figure 4 shows a small house environment. If the spanning tree on the left of Fig. 4 is found, then two tree searchers are required to clear the tree. Assume that the searchers start in cell three. The searchers first move down the spanning tree to cell four. Then one searcher must remain at cell four while the other searcher clears the rest of the graph. Since cell four is the only cell that needs to be guarded to remove the non-tree edges, this yields a two searcher clearing strategy of the original graph. Two is also the minimal number of searchers capable of clearing this environment.

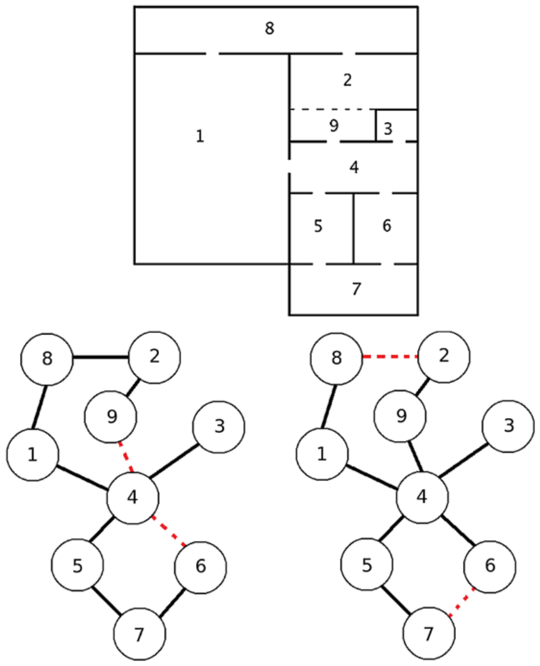


Fig. 4 Example discretization of house environment (*top*) and two example spanning trees of resulting graph (*bottom*). *Solid lines* are spanning tree edges, and *dashed lines* are non-tree edges. If searchers start in cell 3, the left spanning tree gives a two searcher clearing schedule while the right spanning tree gives a three searcher clearing schedule

Even in this simple example, all spanning trees do not yield a minimal (two searcher) strategy. Consider the right spanning tree in Fig. 4. Ignoring the non-tree edges, this spanning tree requires two searchers to clear. However, when clearing the original graph, an extra guard must be placed on one of the cycles. The guard at cell four cannot be reused because its movement would cause recontamination. Thus, this spanning tree yields a three searcher schedule on the original graph due to the necessity of guarding the non-tree edges. This is the motivation for generating many spanning trees and choosing the best schedule.

6 Results

We conducted experiments on a 3.2 GHz Pentium 4 processor with 2 GB RAM running Ubuntu Linux. We implemented and tested three methods for generating spanning trees. They are described below.

1. Spanning tree enumeration (Char 1968): Char's algorithm for enumerating all spanning trees. This is a brute force method for generating all possible spanning trees, which is computationally viable only on small graphs with few spanning trees.
2. Uniform sampling (Wilson 1996): Wilson's algorithm for uniformly sampling the space of spanning trees using loop-erased random walks. The algorithm generates a random spanning tree sampled uniformly from the space of all spanning trees.
3. Depth-first sampling (Sect. 5): randomized depth-first search to generate spanning trees. This technique does not sample the entire space of trees, but it attempts to bias sampling towards trees requiring a small number of guards.

A summary of the five traversal variants described in Sect. 5 is given below.

1. GSST-L: labeled traversal. The labeling of the underlying spanning tree is used deterministically to guide the search (see Algorithm 5).
2. GSST-R: randomized traversal. The underlying spanning tree is traversed randomly. The labeling is not used.
3. GSST-LW: label-weighted traversal. The underlying spanning is traversed in a randomized manner with weights determined by the labeling.[6]
4. GSST-LD: label-dominated traversal. Subtrees of the graph are deterministically cleared as available and other moves are determined randomly.
5. GSST-LDW: label-dominated/weighted traversal. Subtrees of the graph are deterministically cleared as available, and other clearing moves are determined in a randomized manner with weights determined by the labeling.

6.1 Simulated indoor environments

With robotic search in mind, we tested GSST on the two complex indoor environments shown in Fig. 2. The first map is a floorplan of the third floor of Newell-Simon Hall at Carnegie Mellon University. The second is a map of the first floor of the National Gallery of Art in Washington, DC. The Newell-Simon Hall ("office") environment has two major cycles, and the National Gallery ("museum") environment has many cycles by which the target can escape capture. Both of these environments are considerably larger than those searched by many authors using comparable methods (Guibas et al. 1999; Gerkey et al. 2005).

The office map is 100 m $\times$ 50 m discretized into 60 cells with 64 edges. Kirchhoff's matrix-tree theorem shows that the office has 3604 different spanning trees. The museum map is 150 m $\times$ 100 m discretized into 70 cells with 93 edges (5.3×10^{14} spanning trees). Thus, we can exhaustively search the space of spanning trees for the office but not for the museum. It is not immediately obvious, but the

[6]The exact weighting function used in this paper is to continuously step through the frontier edges and generate a random integer $\tau \in [0, \lambda_{\max}]$, where $\lambda_{\max}$ is the maximum label of all frontier edges. A frontier edge e is accepted for traversal if its label $\lambda(e) \leq \tau$. Any probabilistic function that is inversely proportional to the weights could be used in place of this.

Table 1 Comparison of variations of GSST in two environments. The first column gives the method of edge traversal, the second column the minimum number of searchers attained, the third column the percentage of minimal solutions over all computed solutions, and the fourth column the total execution time (for 100,000 spanning trees). The same minimal schedule may be counted more than once in the percent minimum. The starting node is fixed throughout the trials

Office	Uniform ST generation		
Edge traversal	Min	Percent min	Time
GSST-L	3	0.7780%	20.1886
GSST-R	3	0.0040%	15.4610
GSST-LW	3	0.0320%	20.7670
GSST-LD	3	0.3540%	21.5415
GSST-LDW	3	0.4070%	22.1352
	DFS ST generation		
GSST-L	3	19.9880%	18.1599
GSST-R	3	0.0510%	13.8348
GSST-LW	3	0.5390%	20.1243
GSST-LD	3	20.0480%	20.4685
GSST-LDW	3	20.005%	21.0810
Museum	Uniform ST generation		
Edge traversal	Min	Percent min	Time
GSST-L	5	0.0020%	47.5863
GSST-R	5	0.0010%	31.6156
GSST-LW	5	0.0040%	45.4039
GSST-LD	5	0.0090%	45.5225
GSST-LDW	5	0.0110%	45.9819
	DFS ST generation		
GSST-L	5	0.4220%	47.1498
GSST-R	5	0.3360%	29.0825
GSST-LW	5	0.5400%	48.0258
GSST-LD	5	0.8650%	47.4625
GSST-LDW	5	0.8330%	48.0719

office can be cleared with three searchers and the museum with five.[7]

Table 1 shows a comparison of different spanning tree generation techniques and traversal methods on the museum and office maps. GSST is able to generate and search 100,000 spanning trees in under a minute (often under 30 seconds) on these complex maps. We also note that all traversal variants yield similar running times, with the exception of GSST-R that does not need to perform the labeling step. Traversal methods using spanning tree labeling (i.e., all but GSST-R) generate more minimal schedules, which demonstrates the gain from using the optimal traversal of the underlying spanning tree to guide search. The gain from using the labeling is greater on the office map than on the museum map. This is as expected because the office map has fewer cycles and is thus closer to a tree.

On the office and museum, depth-first sampling yields a higher proportion of minimal schedules due to its bias towards graphs requiring fewer guards. However, particularly in the office, depth-first sampling severely limits the space of spanning trees, which leads to only ten distinct schedules (two of which are minimal). This demonstrates the incompleteness of depth-first sampling along with its tendency to produce minimal search schedules on the reduced space of trees.

Figure 5 illustrates the anytime behavior of GSST by showing the attained search number with increasing numbers of generated spanning trees. As above, in both environments, depth-first sampling generates solutions with fewer searchers much more quickly than uniform sampling. In addition, Fig. 6 displays histograms of the number of searchers to clear the graphs for many different spanning trees. Depth-first sampling clearly generates trees that lead to strategies with fewer searchers in these environments. Figure 1 shows example spanning trees yielding minimal search schedules in both environments.

6.2 Performance comparison

Here we compare GSST to several competing algorithms using three additional environments (shown in Fig. 7). The hallway, cave, and Gates Hall environments were introduced by Gerkey et al. (2005) to test their PARISH algorithm. We use their discretization for the Gates and hallway environments, and we generate a constrained Delaunay triangulation for the cave environment (see Sect. 3) since the discretization by Gerkey et al. is not available.

Table 2 shows the attained search number (i.e., the minimal number of searchers with which a clearing schedule was found) for GSST, PARISH, RHGC, and the iterative greedy algorithm. PARISH is a stochastic hill-climbing approach that has much in common with market-based techniques (Gerkey et al. 2005).[8] The use of finite-horizon planning with sequential allocation for guaranteed search is discussed in Sect. 4 and is an extension of our prior work in non-adversarial search (Hollinger et al. 2009b). We introduced the iterative greedy algorithm in our prior work, which uses an approach similar to dynamic programming to generate clearing schedules (Kehagias et al. 2009b). The

[7]While it is not proven that five is the minimal search strategy in the museum, we have been unable to find a four searcher strategy on this map using any method.

[8]The results reported in Table 2 for PARISH are taken directly from Gerkey et al. (2005) and that article's supplementary videos (Gerkey 2004) in which some of the same environments are examined. We have not implemented the PARISH algorithm for this paper.

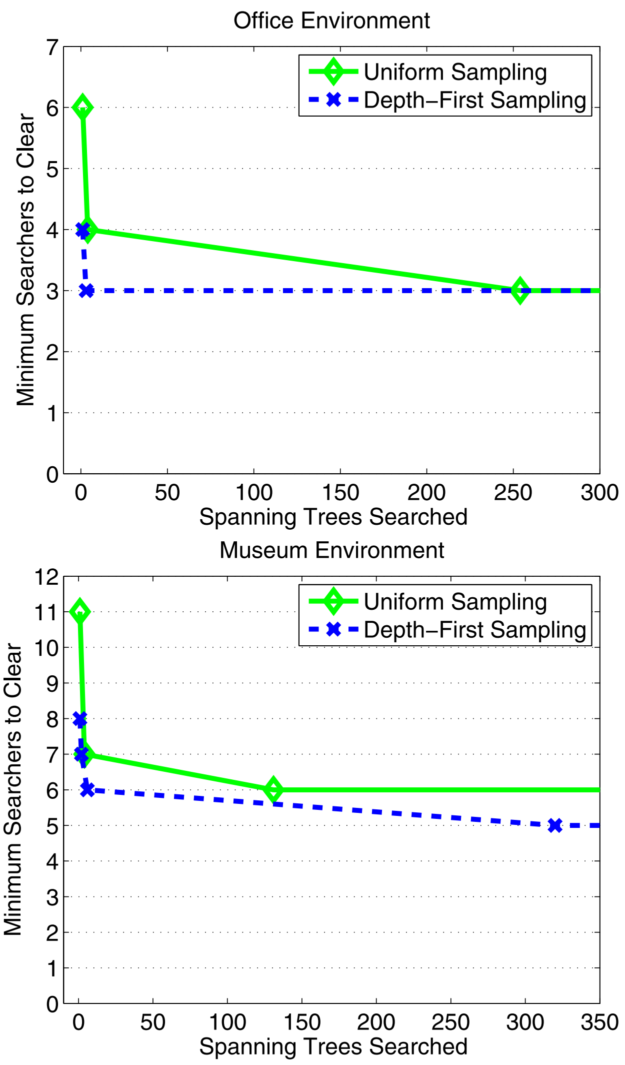


Fig. 5 Demonstration of anytime behavior of GSST in the office (*top*) and museum (*bottom*). In both environments, depth-first spanning tree generation more quickly generates a spanning tree with few searchers. Labeled traversal is used in these experiments

version of GSST employed in this section uses uniform spanning tree generation and labeled traversal. GSST generated 10,000 spanning trees before returning the best solution, which took less than 10 seconds in the larger environments. RHGC and iterative greedy were run with an increasing number of searchers (e.g., $K = 1, 2, \ldots$) until a clearing schedule was found. The horizon depth for RHGC was set to $d = 6$.

GSST yields the lowest attained search numbers in all environments versus these competing algorithms. The improvement in attained search number increases with the complexity of the environment (i.e., number of nodes and edges). This demonstrates the effectiveness of utilizing the underlying spanning tree to guide implicit coordination on large graphs. In addition, the linear scalability and anytime capabilities of GSST allow it to remain effective in large environments like the office and museum maps. Video Extension 1 shows a playback of several searches generated by GSST in these environments.

6.3 Human-robot teams

The experiments above were run on a single processor with shared memory, and they do not demonstrate GSST's potential to be distributed across a team of searchers. In addition, the schedules in the simulated experiments were generated offline, and the paths of the robots were assumed to be completely synchronized. In real-world applications, it is possible to make some modifications to the schedules generated by GSST online. If a searcher is behind in its schedule, then other searchers can continue their schedules until they reach a point where moving would cause recontamination. Furthermore, if an entire branch of the tree is cleared fortuitously (e.g., by some uncontrolled searchers or information), the schedule can prune this branch and continue without clearing it. Note, however, that some online modifications could alter the attained search number of the schedule (e.g., a searcher moving to clear an area instead of moving to a required guard position), and these should be avoided.

To test the distributed and online capabilities of GSST, we ran experiments with a human/robot search team on a single floor of an office building (shown in Fig. 8). Two humans and a single Pioneer robot share their position information across a wireless network, and the entire team is guided by a decentralized implementation of labeled traversal. Figure 9 shows a diagram describing the decentralized system architecture. The searchers communicate through the GSST modules, which receive position feedback and provide waypoints back to the team. This architecture does not require a centralized control mechanism. In addition, the GSST modules are anonymous: they do not need to know whether the position/clearing input comes from a human or a robot.

The robot's position is determined by laser AMCL, and it is given waypoints through the Player/Stage software (Gerkey et al. 2003). The robot uses the built in Wavefront planner and Vector Field Histogram (VFH) obstacle avoidance to follow the waypoints specified by the GSST module. The robot uses a 180 degree FOV camera with a video feed to the humans as its clearing sensor. The humans input their position through the keyboard, and they are given waypoints through a GUI.

Prior to search, the searchers generated spanning trees in parallel until a three-searcher schedule was found. After agreeing on an underlying spanning tree, the traversal was computed during search using the decentralized network. Thus, the search can be modified on-the-fly if necessary during clearing. For instance, one of the human searchers performed its clearing schedule more slowly than expected. The

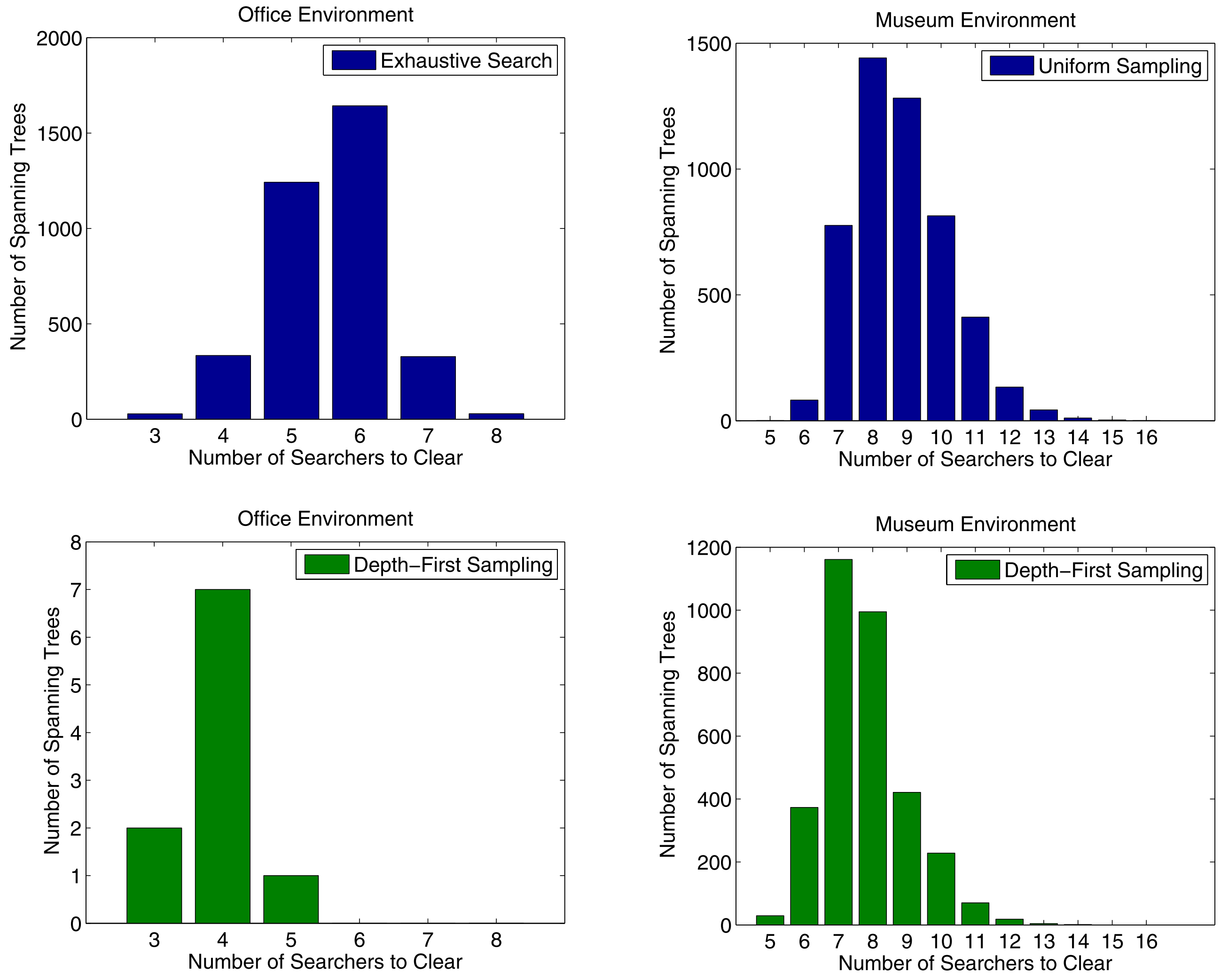


Fig. 6 Histograms of number of searchers to clear the office (*left*) and museum (*right*) using labeled traversal on many different spanning trees. In the office, exhaustive search is compared to depth-first sampling. Depth-first sampling limits the space of spanning trees searched but generates trees that require fewer searches. In the museum, uniform sampling of 30,000 trees is substituted for exhaustive search. Again, depth-first sampling generates spanning trees requiring fewer searchers

other searchers were able to continue their schedules without that human until they reached a point of recontamination. This avoided strict synchronization, which would have required more costly communication.

The first three-searcher schedule was generated very quickly (less than a second), and the network then came to consensus on an acceptable spanning tree. The human-robot search team proceeded to clear the floor of a potential adversary. The schedule clears the left portion of the map first and then continues to clear the right portion. Video Extension 2 shows a playback of the search. In addition to showing the distributed and online capabilities of GSST, our results with a human/robot team demonstrate the feasibility of the communication and computational requirements on a small team.

7 Conclusions

We have presented GSST, an anytime guaranteed search algorithm applicable to complex physical environments. GSST generates guaranteed search schedules on arbitrary graphs by leveraging the easier problem of generating a schedule on an underlying spanning tree. Our algorithm works in an anytime manner by quickly generating an initial schedule with potentially many searchers and then producing more schedules with increasing computation. We have shown several variations of GSST that utilize the underlying spanning tree in different ways, and we have given a novel method for generating random spanning trees that tend to yield search schedules with few searchers.

We have shown that GSST produces a feasible clearing schedule at every iteration, and we have proven that several

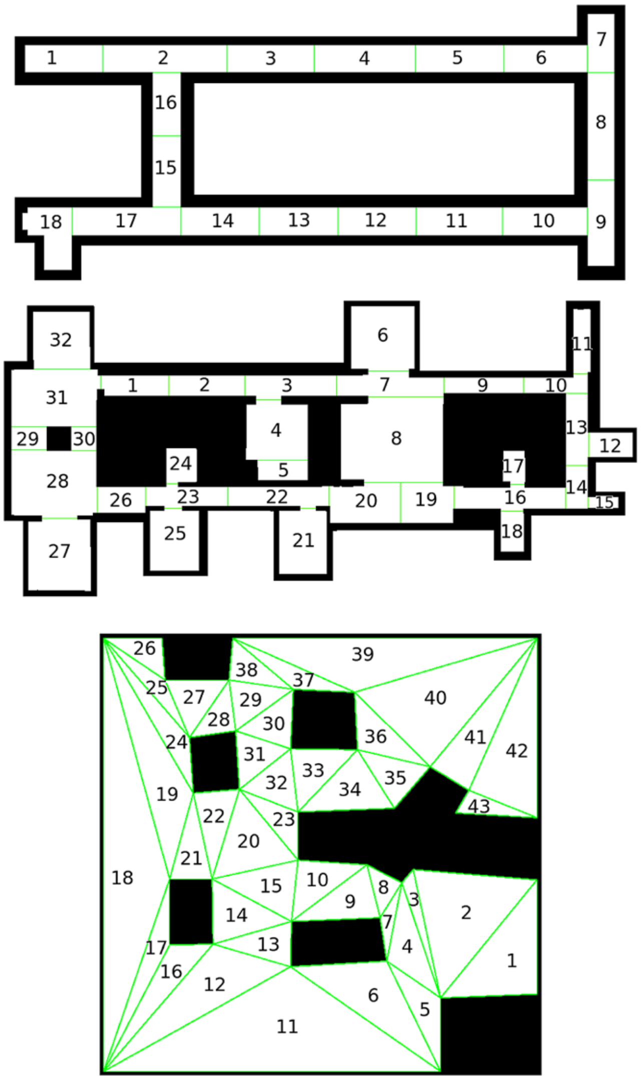


Fig. 7 Additional simulated environments used to compare GSST to competing algorithms. The hallway (*top*) and Gates (*middle*) environments use the same discretizations as Gerkey et al. (2005). The cave (*bottom*) was discretized using a constrained Delaunay triangulation

Table 2 Comparison of attained search numbers of GSST (proposed algorithm) with PARISH (Gerkey et al. 2005), RHGC (Sect. 4), and Iterative Greedy (Kehagias et al. 2009b). GSST yields the lowest attained search number in all environments with more improvement in more complex environments

	Hallway	Cave	Gates	Office	Museum
PARISH	2	4	5	–	–
RHGC	2	3	4	6	7
Iter. Greedy	2	3	4	4	6
GSST	2	3	3	3	5

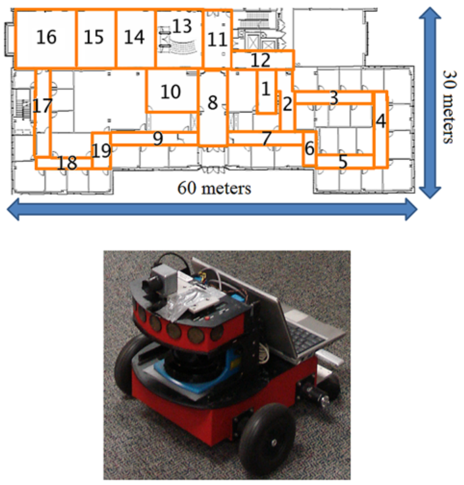


Fig. 8 Map of office building (*top*) used for experiments with a human/robot search team. Pioneer robot searcher (*bottom*) with laser scanner and camera. The robot used the laser scanner for localization, and it used the camera to provide a video feed of the search environment. The robot and human team started their search at the building entrance in cell eight

variations are probabilistically complete. In addition, GSST is linearly scalable in the number of nodes in the search graph, making it applicable to very large environments. We also have given loose performance bounds on the algorithm based on the worst-case number of searchers required to traverse the underlying spanning tree.

GSST is an implicit coordination algorithm, which allows for each agent to plan for itself without considering the joint planning space. The agents share the spanning tree of the environment as an informed representation, and they communicate information about their actions to execute a joint schedule. This indirect form of coordination provides the necessary coupling to generate schedules in difficult clearing instances without requiring exponential computation.

Our results demonstrate both the scalability and effectiveness of GSST. Our implementation of GSST generates over 2000 schedules per second in an environment with 70 nodes and 93 edges. We have shown that GSST outperforms three competing algorithms in five environments ranging from a cave to a museum. In some cases, GSST reduces the required number of searchers by 50% over competing algorithms.

In addition, we have demonstrated the feasibility of GSST with an implementation on a human-robot search team. The distributed implementation utilizes existing wireless infrastructure to communicate the searchers' positions, and the team effectively clears a floor of an office building of any potentially adversarial target. Furthermore, the schedule is modified online to account for poor synchronization between the searcher paths. To our knowledge, this is the first implementation of a clearing schedule on this scale, thus highlighting the real-world applicability of GSST to robotic search.

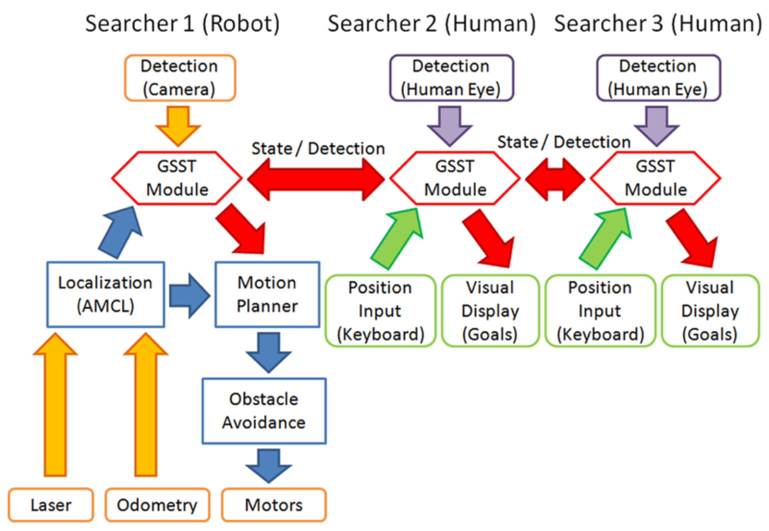


Fig. 9 Diagram describing system architecture for human-robot search experiments. The decentralized GSST modules receive position feedback from the heterogeneous search team and then provide search waypoints back to the team

7.1 Future work

Our findings open up several interesting avenues for improvements to GSST. In the current paper, trees were selected randomly using one of two methods. Alternatively, informed heuristics could be used to generate spanning trees with a high likelihood of yielding low-searcher strategies. GSST could also incorporate traversal strategies other than the ones presented here. One potential avenue to explore is the use of a traversal based on a graph cut algorithm similar to the one used by Kolling and Carpin (2010) for weighted graph searching on trees.

In addition, our proposed algorithm does not provide a bound on the current solution quality relative to optimal. In other words, when the search is stopped, we cannot say whether or not we have found a minimal (or near-minimal) search strategy. Solving this problem requires a method for bounding the minimum number of searchers from below. One of the main challenges in generating near-optimal performance guarantees for GSST is to do so while keeping the algorithm linearly scalable, distributable, modifiable online, and preserving of anytime capabilities.

The strategies discussed in this paper do not directly consider clearing time as a performance metric. In many applications, faster clearing schedules are desirable. Our ongoing work in combining efficient search with guaranteed search discusses a variation of GSST that incorporates clearing time considerations (Hollinger et al. 2009a).

As mentioned in Sect. 3, discretization of environments for search is still an active research area. Automatic discretization using the constrained Delaunay triangulation (Shewchuk 2002) or the Voronoi diagram (Kolling and Carpin 2008) are two possible approaches. Visibility-based methods that take advantage of environment geometry are an alternative (Guibas et al. 1999), but a visibility-based method that produces a sufficiently small number of convex cells in complex, cluttered environments is still an open problem. In some cases, a larger number of cells may actually require fewer searchers, which leaves open the question of how best to discretize to help minimize the number of searchers. In addition, alternative methods for taking into account varying sensor modalities (e.g., limited range and limited FOV) yield interesting avenues for future work.

An important open problem regarding GSST is the completeness of labeled traversal. In other words, can we guarantee that at least one minimal labeled traversal exists after having examined all spanning trees of a graph. It is possible to construct worst-case graphs where a minimal labeled traversal for a particular starting node is not possible (Kehagias et al. 2009a), but we have not found such an example if the starting node is left unspecified.

GSST restricts the moves of the searchers to those that do not recontaminate nodes in the graph. Node recontamination occurs if a searcher leaves a node unguarded, and one or more of its adjacent nodes are dirty (with the exception of the node the searcher is moving into). Node recontamination can improve the search number for connected search on graphs (Yang et al. 2004). However, situations in which recontamination helps may be rare in realistic environments, making it difficult to incorporate them into guaranteed search.

In addition to solving the guaranteed search problem, our algorithm serves as an efficient method for generating connected path decompositions of planar graphs. These decompositions are similar but not exactly the same as those studied in the literature (i.e., ours are derived from a node game rather than edge game). Generating path decompositions is an important graph theoretic problem with applications outside of guaranteed search (Kloks 1994). The linear scalability of our algorithm makes it well-suited for the generation of path decompositions on very large graphs. Future work includes studying the relationship between path decompositions generated from a node search game and those generated from an edge search game.

Acknowledgements The authors gratefully acknowledge Joseph Djugash, Ben Grocholsky, Dave Ferguson, and Siddhartha Srinivasa for their insightful comments. Thanks to volunteer firefighter (and professional roboticist) Seth Koterba for feedback on the system. Parts of this work utilized Jonathan Richard Shewchuk's free Delaunay triangulation software. This research is partially funded by the National Science Foundation under Grant No. IIS-0426945. Further thanks to Intel Research Pittsburgh.

Appendix A: Index to multimedia extensions

Extension	Media type	Description
1	Video	Simulated guaranteed search with GSST
2	Video	Guaranteed search with a human-robot team

Appendix B: Proof of completeness

B.1 Definitions

Definition 2 Given a graph G and an IMC node clearing schedule $\mathbf{S}$ defined over time $t = 1, 2, \dots, t_f$. Let $sn(\mathbf{S}, t)$ be the number of searchers required in the schedule at time step t. Let $\overline{sn}(\mathbf{S})$ be the maximum number of searchers required for the schedule.[9]

Definition 3 Given a graph G and a search $\mathbf{S}$ of G, the *frontier* at t (under $\mathbf{S}$) is

$$N_F(t) = \{u : u \in N_C(t) \text{ and } \exists v : v \in N_D(t), uv \in E\},$$

i.e., the clear nodes which are connected to dirty nodes.

B.2 Completeness

Lemma 1 *Given a graph $G = (N, E)$ and a rooted IMC node clearing search $\mathbf{S}$ of G. The clearing moves of $\mathbf{S}$ generate a sequence of trees, $(\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_{|N|})$, where (for $n = 1, 2, \dots, |N|$) $\mathbf{T}_n = (N_n, E_n)$ and the following hold*:

D1 $\mathbf{T}_0$ *is the empty graph* (*and* $N_0 = \emptyset$, $E_0 = \emptyset$);
D2 $N_{|N|} = N$, $E_{|N|} \subseteq E$;
D3 *for* $n = 1, 2, \dots, |N|$: $N_{n-1} \subseteq N_n \subseteq N$, $E_{n-1} \subseteq E_n \subseteq E$ (*in other words* $\mathbf{T}_{n-1}$ *is a subtree of* $\mathbf{T}_n$);
D4 *for* $n = 1, 2, \dots, |N|$: $N_n = N_{n-1} \cup \{u_n\}$, *and for* $n = 2, 3, \dots$: $E_n = E_{n-1} \cup \{u_i u_n\}$, *with* $i \in [1, n-1]$.

Proof Inductively. Since $\mathbf{S}$ is monotone, it involves $|N|$ clearing moves. $\mathbf{T}_0$ is the empty graph. $\mathbf{T}_1$ is formed by the first move of $\mathbf{S}$, which consists in placing a searcher at the root node. So $\mathbf{T}_1$ is the tree with a single node and trivially has $\mathbf{T}_0$ as a subgraph. Suppose D3 and D4 hold up to $m = n$ and consider the $(n+1)$-th clearing move of $\mathbf{S}$. Since $\mathbf{S}$ is connected, we add to $\mathbf{T}_n$ one node u_{n+1} and *exactly* one edge $u_i u_{n+1}$ (with $i \in [1, n]$) to obtain a new tree $\mathbf{T}_{n+1}$ (of $n + 1$ nodes and n edges) which also satisfies D3 and D4. Hence D3 and D4 hold for $m = 1, 2, \dots, |N|$. At $m = |N|$, $N_{|N|}$ contains $|N|$ nodes, hence $N_{|N|} = N$; since $\mathbf{T}_{|N|}$ is a tree, it is a spanning tree of G. □

Corollary 1 *Given a graph $G = (N, E)$, every minimal IMC node clearing search $\mathbf{S}$ of G generates a sequence of trees which satisfy the conditions of Lemma 1.*

Lemma 2 *Given a graph $G = (N, E)$ and a tree sequence $(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|})$. Then GSST-R/GSST-LW using a single uniformly generated spanning tree has a nonzero probability of producing a search $\mathbf{S}$ which generates $(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|})$.*

Proof The probability of generating the tree sequence $(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|})$ is:

$$\begin{aligned}
&\Pr(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|}) \\
&\quad = \left[\prod_{n=1}^{|N|} \Pr(\mathbf{T}_n | \mathbf{T}_{|N|}, \mathbf{T}_0, \dots, \mathbf{T}_{n-1}) \right] \\
&\qquad \times \Pr(\mathbf{T}_0 | \mathbf{T}_{|N|}) \Pr(\mathbf{T}_{|N|}).
\end{aligned}$$

Note that the conditioning in the above expression *always* includes $\mathbf{T}_{|N|}$, since this is the first choice made in running GSST-R / GSST-LW. Now obviously, $\Pr(\mathbf{T}_0 | \mathbf{T}_{|N|}) = 1$. $\Pr(\mathbf{T}_{|N|}) > 0$ is nonzero for any uniform spanning tree generator.

$\Pr(\mathbf{T}_n | \mathbf{T}_{|N|}, \mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{n-1})$ is the probability of expanding (at the n-th step) $\mathbf{T}_{n-1}$ by the edge $u_i u_n \in \mathbf{E}_n - \mathbf{E}_{n-1}$ which, by the construction of both GSST-R and GSST-LW, is always positive. Finally, $\Pr(\mathbf{T}_{|N|} | \mathbf{T}_{|N|}, \mathbf{T}_0, \mathbf{T}_1, \dots, T_{|N|-1}) = 1$. Hence $\Pr(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|}) > 0$ for every sequence $\mathbf{T}_1, \dots, \mathbf{T}_{|N|}$. □

Lemma 3 *Given a graph $G = (N, E)$ and a rooted IMC node clearing search $\mathbf{S}$ of G; let $(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|})$ be the tree sequence generated by $\mathbf{S}$. Let $\mathbf{S}'$ be a search produced by either GSST-R or GSST-LW and* also *generating $(\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_{|N|})$. Then $\overline{sn}(\mathbf{S}) \geq \overline{sn}(\mathbf{S}')$.*

[9]Note that $\overline{sn}(\mathbf{S}) = sn(\mathbf{S}, t_f)$ in an IMC clearing schedule since no searchers are removed.

Proof The proof is exactly the same for GSST-R and GSST-LW, so we only prove the first one, by induction. Let $t_1, \ldots, t_{|N|}$ be the clearing times of $\mathbf{S}$ and $t'_1, \ldots, t'_{|N|}$ be the clearing times of $\mathbf{S}'$. Also let $t_0 = t'_0 = 0$.

At $t_0 = t'_0 = 0$ we have $sn(\mathbf{S}', 0) = sn(\mathbf{S}, 0) = 0$.

The only times at which $sn(\mathbf{S}', t)$ may change are $1, t'_1 + 1, \ldots, t'_{|N|-1} + 1$. Suppose that

$$sn(\mathbf{S}, t_n) \geq sn(\mathbf{S}', t'_n).$$

Further, suppose that at $t'_n + 1$ a new searcher is introduced in $\mathbf{S}'$. This can only happen (in the $\mathbf{S}'$ search) if all of the following hold:

1. at t'_n exactly $|N_F(t)|$ searchers exist in G;
2. there are no searchers inside nodes $u \in N_C(t'_n) - N_F(t'_n)$ (i.e., all searchers are located inside frontier nodes);
3. all searchers are stuck (i.e., moving a searcher out of a frontier node u exposes u to recontamination).

The sequence $(\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_{|N|})$ and the clearing times determine the frontier $N_F(t)$ for every t. Hence $\mathbf{S}'$ at t'_n has the same frontier as $\mathbf{S}$ at t_n. If conditions 1–3 above hold in $\mathbf{S}'$, then every searcher is located in a frontier node and is stuck. It is possible that non-stuck searchers exist in $\mathbf{S}$ (located either in frontier or non-frontier nodes) but this also means that $sn(\mathbf{S}, t_n) \geq sn(\mathbf{S}', t'_n) + 1$; hence adding a searcher in $\mathbf{S}'$ at $t'_n + 1$ preserves

$$sn(\mathbf{S}, t_n) \geq sn(\mathbf{S}', t'_n + 1).$$

Since no searchers are added in $\mathbf{S}'$ for $t \in [t'_n + 2, t'_{n+1}]$ and no searchers are ever removed in $\mathbf{S}$ (i.e., $sn(\mathbf{S}, t_{n+1}) \geq sn(\mathbf{S}, t_n)$) we also get

$$sn(\mathbf{S}, t_{n+1}) \geq sn(\mathbf{S}', t'_{n+1}).$$

From the above inequality inductively we get $sn(\mathbf{S}, t_{|N|}) \geq sn(\mathbf{S}', t'_{|N|})$, which proves the lemma. □

Theorem 1 *At each iteration, both random traversal (GSST-R) and label-weighted random traversal (GSST-LW) of a uniformly generated spanning tree have a nonzero chance of yielding a minimal monotone/connected node search strategy on any graph.*

Proof Restating, we must prove that given a graph $G = (N, E)$:

1. GSST-R will generate a minimal monotone, connected clearing of G with probability greater than or equal to $1 - \alpha_1^M$ where M is the number of iterations and $\alpha_1 \in (0, 1)$.
2. GSST-LW will generate a minimal monotone, connected clearing of G with probability greater than or equal to $1 - \alpha_2^M$ where M is the number of iterations and $\alpha_2 \in (0, 1)$.

The proof is exactly the same for GSST-R and GSST-LW, so we only prove the first one. G has at least one minimal rooted IMC node clearing search $\mathbf{S}$ of G. Let $(\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_{|N|})$ be the tree sequence generated by $\mathbf{S}$. By Lemma 2, GSST-R has a nonzero probability, call it β_1, of generating *in a single iteration* a search $\mathbf{S}'$ with the same tree sequence as $\mathbf{S}$. Then, by Lemma 3,

$$\overline{sn}(\mathbf{S}) \geq \overline{sn}(\mathbf{S}').$$

Since $\mathbf{S}$ is minimal, $\overline{sn}(\mathbf{S}) = \overline{sn}(\mathbf{S}')$ and so $\mathbf{S}'$ is minimal too. Now, the probability of *not* generating $\mathbf{S}'$ in a single iteration is $\alpha_1 = 1 - \beta_1$; and the probability of *not* generating $\mathbf{S}'$ in M iterations is $\alpha_1^M = (1 - \beta_1)^M$, while the probability of generating $\mathbf{S}'$ in M iterations is $1 - \alpha_1^M$. □

References

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

Barrière, L., Flocchini, P., Fraigniaud, P., & Santoro, N. (2002). Capture of an intruder by mobile agents. In *Proc. 14th ACM symp. parallel algorithms and architectures* (pp. 200–209).

Barrière, L., Fraigniaud, P., Santoro, N., & Thilikos, D. (2003). Searching is not jumping. *Graph-Theoretic Concepts in Computer Science*, *2880*, 34–45.

Bienstock, D., & Seymour, P. (1991). Monotonicity in graph searching. *Journal of Algorithms*, *12*(2), 239–245.

Char, J. (1968). Generation of trees, two-trees, and storage of master forests. *IEEE Transactions on Circuit Theory*, *15*(3), 228–238.

Dendris, N., Kirousis, L., & Thilikos, D. (1994). Fugitive-search games on graphs and related parameters. In *Proc. 20th int. workshop graph-theoretic concepts in computer science* (pp. 331–342).

Flocchini, P., Nayak, A., & Schulz, A. (2005). Cleaning an arbitrary regular network with mobile agents. In *Proc. int. conf. distributed computing and Internet technology* (pp. 132–142).

Flocchini, P., Huang, M., & Luccio, F. (2007). Decontamination of chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, *18*(3), 547–564.

Flocchini, P., Huang, M., & Luccio, F. (2008). Decontamination of hypercubes by mobile agents. *Networks*, *52*(3), 167–178.

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

Fomin, F., Fraigniaud, P., & Thilikos, D. (2004). *The price of connectedness in expansions*. Technical Report LSI-04-28-R, UPC Barcelona.

Fraigniaud, P., & Nisse, N. (2006). Connected treewidth and connected graph searching. In *Proc. 7th Latin American symp. theoretical informatics*.

Gerkey, B. (2004). Pursuit-evasion with teams of robots. http://ai.stanford.edu/~gerkey/research/pe/index.html.

Gerkey, B., Vaughan, R., & Howard, A. (2003). The player/stage project: tools for multi-robot and distributed sensor systems. In *Proc. int. conf. advanced robotics* (pp. 317–323).

Gerkey, B., Thrun, S., & Gordon, G. (2005). Parallel stochastic hill-climbing with small teams. In *Proc. 3rd int. NRL workshop multi-robot systems*.

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

Hollinger, G., Kehagias, A., & Singh, S. (2009a). Efficient, guaranteed search with multi-agent teams. In *Proc. robotics: science and systems conf.*

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

Isler, V., Kannan, S., & Khanna, S. (2005). Randomized pursuit-evasion in a polygonal environment. *IEEE Transactions on Robotics*, *21*(5), 875–884.

Kalra, N. (2006). *A market-based framework for tightly-coupled planned coordination in multirobot teams*. Ph.D. thesis, Robotics Institute, Carnegie Mellon Univ.

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

Kehagias, A., Hollinger, G., & Singh, S. (2009b). A graph search algorithm for indoor pursuit/evasion. *Mathematical and Computer Modelling*, *50*(9–10), 1305–1317.

Kloks, T. (1994). *Treewidth: computations and approximations*. Berlin: Springer.

Kolling, A., & Carpin, S. (2008). Extracting surveillance graphs from robot maps. In *Proc. int. conf. intelligent robots and systems*.

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

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

LaPaugh, A. (1993). Recontamination does not help to search a graph. *Journal of ACM*, *40*(2), 224–245.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge: Cambridge University Press.

LaValle, S., Lin, D., Guibas, L., Latombe, J., & Motwani, R. (1997). Finding an unpredictable target in a workspace with obstacles. In *Proc. IEEE international conf. robotics and automation*.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). Anytime dynamic A*: an anytime, replanning algorithm. In *Proc. int. conf. automated planning and scheduling*.

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

Parsons, T. (1976). Pursuit-evasion in a graph. In Y. Alavi, & D. Lick (Eds.) *Theory and applications of graphs* (pp. 426–441). Berlin: Springer.

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

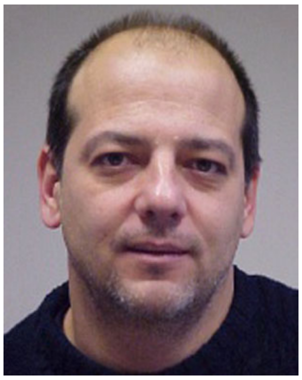
Smith, T. (2007). *Probabilistic planning for robotic exploration*. Ph.D. thesis, Robotics Institute, Carnegie Mellon Univ.

Wilson, D. (1996). Generating random spanning trees more quickly than the cover time. In *Proc. 28th ACM symp. theory of computing* (pp. 296–303).

Yang, B., Dyer, D., & Alspach, B. (2004). Sweeping graphs with large clique number. In *Proc. 5th international symp. algorithms and computation* (pp. 908–920).

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *Artificial Intelligence Magazine*, *17*(3), 73–86.

Geoffrey Hollinger is a Ph.D. Candidate at Carnegie Mellon University in the Robotics Institute. He is currently interested in designing scalable and distributed algorithms for estimation and multi-robot coordination in the physical world. He has worked on personal robotics at Intel Research Pittsburgh, active estimation at the University of Pennsylvania's GRASP Laboratory, and miniature inspection robots for the Space Shuttle at NASA's Marshall Space Flight Center. He received his M.S. in Robotics from Carnegie Mellon University in 2007 and his B.S. in General Engineering along with his B.A. in Philosophy from Swarthmore College in 2005.



Athanasios Kehagias is an Assistant Professor of Applied Mathematics at the Aristotle University of Thessaloniki. He received the Dipl. Ing. Degree in electrical engineering from the School of Engineering of Aristotle University of Thessaloniki in 1984, the M.Sc. in Applied Mathematics from Lehigh University in 1986 and the Ph.D. in applied mathematics from Brown University, Providence, RI, in 1992. Since November 1999, he has been with the Department of Mathematics, Physical and Computational Sciences, School of Engineering, Aristotle University of Thessaloniki, Thessaloniki, Greece. His research interests include mathematical modeling, applications of probability theory, algebra and fuzzy sets.

Sanjiv Singh is a Research Professor at the Robotics Institute, Carnegie Mellon University. His recent work has two main themes: perception in natural environments and multi-agent coordination. He has led projects in both ground and air vehicles operating in unknown or partially known environments, in applications such as mining, agriculture, emergency response, surveillance and exploration. He is also actively involved in the automation of complex tasks, such as the assembly of large space structures, that cannot be addressed by single agents and must necessarily be performed by teams. Prof. Singh received his B.S. in Computer Science from the University of Denver (1983), M.S. in Electrical Engineering from Lehigh University (1985) and a Ph.D. in Robotics from Carnegie Mellon (1995). He is the founder and Editor-in-Chief of the Journal of Field Robotics.