Anytime Guaranteed Search using Spanning Trees

Geoffrey Hollinger Athanasios Kehagias Sanjiv Singh
Dave Ferguson Siddhartha Srinivasa

CMU-RI-TR-08-36

July 2008

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

This technical report presents an anytime algorithm for solving the multi-robot guaranteed search problem. Guaranteed search requires a team of robots to clear an environment of a potentially adversarial target. In other words, a team of searchers must generate a search strategy guaranteed to find a target. This problem is known to be NP-complete on arbitrary graphs but can be solved in linear-time on trees. Our proposed algorithm reduces an environment to a graph representation and then randomly generates a spanning tree of the graph. Guards are then placed on nodes in the graph to eliminate non-tree edges, and a search strategy for the remaining tree is found using a linear-time algorithm from prior work. To reduce the number of guards, our algorithm takes advantage of the temporal characteristics of the search schedule to reuse guards no longer necessary at their previous locations. Many spanning trees are analyzed prior to search to determine the tree that yields the best search strategy. At any time, spanning tree generation can be stopped yielding the best strategy thus far. Our proposed algorithm is demonstrated on two complex graphs derived from physical environments, and several methods for generating candidate spanning trees are compared.

Contents

1 Introduction

Imagine you are with a group in a large, complex building like a museum, supermarket, or office. 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.

This paper examines the problem of guaranteed search in indoor environments with multiple autonomous robots. The problem of coordinating teams of mobile robots to search large indoor 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, the 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.

In general, multi-robot search problems scale exponentially with increasing searchers, which makes them computationally intractable for large teams and large environments. This intractability arises because multiple searchers must consider the joint action space over all searchers. The size of this planning space grows exponentially in the number of searchers. Considering the joint action 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 is particularly problematic during guaranteed search because progress may be impossible without coordination between searchers. Market-based techniques use synthetic auctions to explicitly coordinate only when necessary. This is one way to strike a balance between explicit coordination and no coordination at all, but determining when to explicitly coordinate and how much explicit coordination is necessary is a difficult problem in itself.

If searchers share information 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 information. We present an algorithm that utilizes *implicit coordination* to achieve better scalability than centralized and market-based approaches. An implicitly coordinated solution is one in which robots share information but do not plan the actions of their teammates.

Simple implicit coordination can provide poor solutions in domains like guaranteed search that require tight coordination. To improve performance for these problems, searchers can perform a shared pre-processing step, which transforms the environment representation into one solvable through implicit coordination. This paper shows how generating spanning trees of the environment can yield implicitly coordinated search strategies for guaranteed search. This approach leads to an "anytime" algorithm for guaranteed search. An anytime algorithm quickly finds a feasible solution and then generates improved solutions over time. Our technique yields feasible guaranteed search paths even in very large environments.

This paper presents a novel guaranteed search algorithm that improves its performance with increasing runtime. The algorithm quickly generates a feasible solution and then finds better solutions by sampling the space of spanning trees. This paper is organized as follows. Section 2 describes related work in graph search, pursuit-evasion, and approximation algorithms. A survey of the literature shows that a scalable approximation algorithm for guaranteed search does not exist. Section 3 formally defines the guaranteed search problem on both arbitrary graphs and trees. Section 4 describes our anytime approximation algorithm for solving guaranteed search. Section 5 gives both theoretical and simulated experimental analysis of our proposed algorithm. Finally, Section 6 draws conclusions and discusses avenues for future work.

2 Related Work

Guaranteed search on graphs has a long history in both robotics and mathematics. Parsons developed some of the earliest methods for solving the adversarial pursuit-evasion problem on graphs (Parsons, 1976). 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 later found to be an NP-complete problem (Megiddo et al., 1988). 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). This research is primarily concerned with examining the hardness of the guaranteed search problem on graphs. They do not present an algorithm for guaranteed search 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. Barriere et al. introduced the idea of connected search during which searchers must maintain a connected subgraph of cleared nodes (Barrière et al., 2003). This 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. Barriere et al. argue that this is an important quality for search strategies in the network decontamination 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 an unconnected one (Fomin et al., 2004).

Early research in guaranteed search showed that many variations of the problem are NP-hard on arbitrary graphs (Megiddo et al., 1988). This result suggests that exact solutions to the guaranteed search problem on large graphs are intractable (unless P = NP). The NP-hardness of search problems has motivated researchers to develop guaranteed search algorithms on special cases of graphs. Barriere et al. showed that an optimal connected search strategy with minimum searchers can be found in linear-time on trees (Barrière et al., 2002). 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. Our proposed guaranteed search algorithm extends the advantages of special case solutions by transforming a graph into its spanning tree through the use of guards and then running an exact search algorithm on the spanning tree.

Since many applications require guaranteed search in arbitrary environments, prior work has proposed some approximation algorithms that provide good performance. Flocchini et al. examined a genetic algorithm approach for clearing arbitrary graphs (Flocchini et al., 2005). 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 (more than the optimal) in fairly simple environments. We have shown that a dynamic programming inspired algorithm, which attempts to iteratively maximize the number of cleared cells, can work well on complex graphs (Kehagias et al., 2008). However, this algorithm does not provide a mechanism for minimizing clearing times during search and often forces the searchers to take many unnecessary steps before finally clearing the environment. In addition, this method is unable to find a minimal search strategies on many complex graphs. 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 a minimal (i.e., using minimum number of searchers) 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 minimal solution to the guaranteed (invisible) search problem. Approximation algorithms for treewidth and pathwidth have been proposed (Klok, 1994), but they are not guaranteed to provide connected or internal path decompositions.

Fraigniaud et al. proposed an algorithm for connected search by finding approximately minimal width tree decompositions (Fraigniaud and Nisse, 2006). 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, their approximation bound degrades fairly quickly with the size of the graph.

With robotic applications in mind, Guibas and LaValle extended guaranteed search techniques to guarantee capture in polygonal environments (LaValle et al., 1997; Guibas et al., 1999; LaValle,

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

2006). Their algorithm discretizes polygonal environments into conservative visibility regions and then uses an information space approach to develop complete algorithms that guarantee capture in one-searchable graphs. For a single pursuer, these algorithms are guaranteed to find a solution if one exists. When scaled to multiple pursuers, however, they lose this property. Additionally, these algorithms are difficult to extend to complex environments because of the sheer number of (often very small) cells necessary in a conservative visibility discretization.

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 presented Hoplites, an algorithm that utilizes auction-based plan sharing to perform tightly coupled tasks (Kalra, 2006). 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. Methods using implicit coordination, on the other hand, provide an alternative that does not require the overhead of auctions.

Gerkey et al. also developed a parallel stochastic hill-climbing method for small teams that is closely related to auction-based methods (Gerkey et al., 2005). 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.

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 quick, near-optimal solutions to guaranteed search problems.

3 Problem Setup

Guaranteed search requires the coordination of multiple robotic searchers 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 and Singh, 2008), guaranteed search makes a worst-case assumption on the target's path.

To formulate the guaranteed search problem, we need to describe the environment in which the searchers and target are located. We first divide the environment into convex cells. The convexity of the cells guarantee that a searcher in a given cell will have line-of-sight to a target in the same cell. The searcher's goal is now to move into the same cell as the target. Gaining line-of-sight is relevant to most sensors that a mobile robot would carry including cameras and laser rangefinders. Our method for discretization takes 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. Alternatively, a suitable discretization can be found automatically using a convex region finding algorithm (such as Quine-McClusky (Singh and Wagh, 1987)).

Taking into account the cell adjacency in a discretized map yields an undirected graph that the searchers can traverse. Figure 1 shows two example floorplans used our experiments. We use the two large floorplans for simulated trials in Section 5. The museum floorplan is particularly challenging because it contains many cycles by which the target can avoid line-of-sight contact with the searchers.

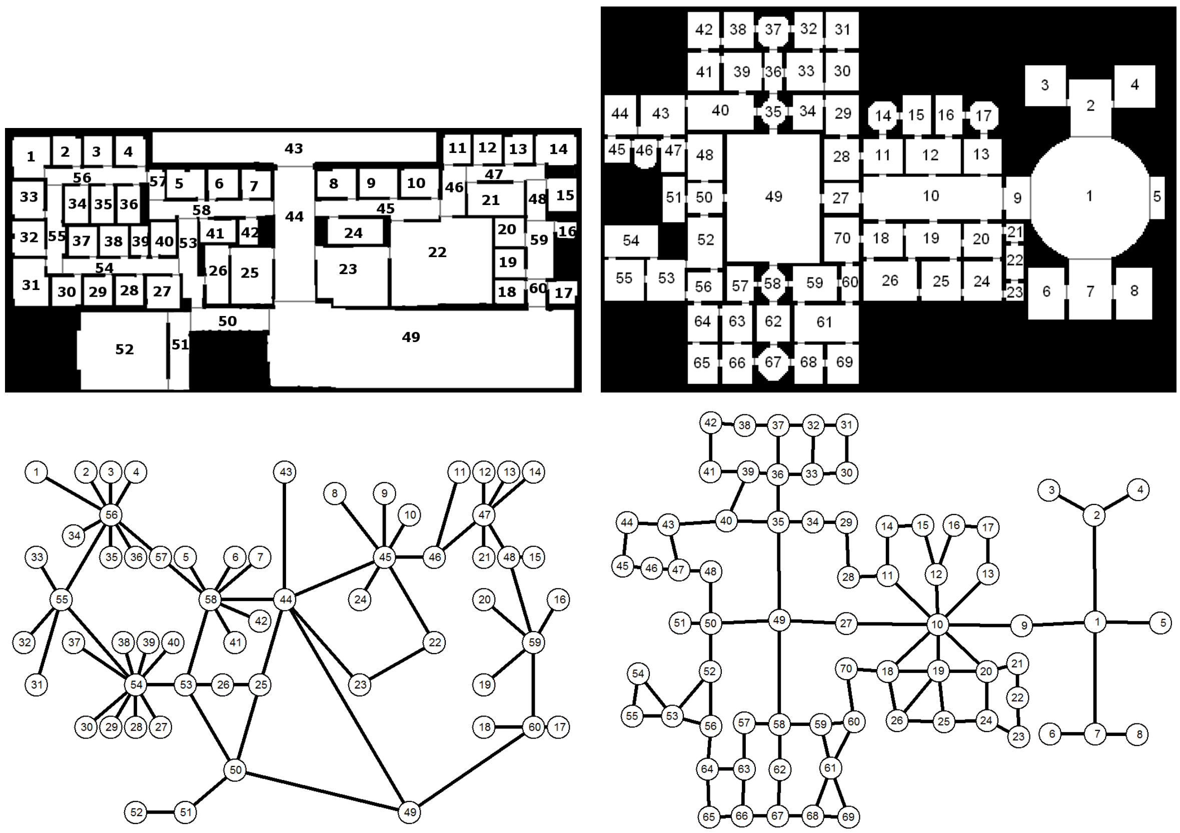

Figure 1: Example floorplans (top) and graphical representation (bottom) of environments used for guaranteed search trials. The office (left) and museum (right) were used for simulated testing.

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 searchers' movement is deterministically controlled, and they 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 vertexes. 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 that the target may have potentially unbounded speed. At any time during the search, there are cells that may contain the target (dirty cells) and cells that may not (cleared cells). The searchers' goal is to progressively decrease the set of dirty cells to the empty set, thus guaranteeing capture of any target in the environment.

This paper examines 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. This is different from the edge search problem discussed in the literature, during which the target exists in the edges of the graph (Parsons, 1976). Searching indoor 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. This paper deals with the node search formulation because of its direct connection with indoor searching.

This paper presents an algorithm for guaranteed node search on graphs by searching their spanning trees. The proposed algorithm draws on a previous algorithm by Barrière et al. for guaranteed search on trees that runs in $O(|N|)$ time, where $|N|$ is the number of nodes on the tree (Barrière et al., 2002). It is important to note that this algorithm generates search paths that are monotonic, connected, and internal. Internal search paths restrict the movement of the searchers to the edges of the graph (i.e. searchers cannot teleport), and connected search paths always maintain a connected subgraph of cleared cells. Both of these 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.

4 Algorithm Description

This section describes an algorithm for guaranteed search on arbitrary graphs generated from floorplans. It is first shown that simple implicit coordination can generate poor search schedules for guaranteed search. It is then shown how augmenting the environment representation with a spanning tree allows for better implicitly coordinated solutions.

4.1 Simple Implicit Coordination

In prior work, we presented an approximation algorithm for solving the efficient search path planning problem (Hollinger and Singh, 2008). In this domain, searchers must maximize the probability of finding a non-adversarial target. During our efficient search approximation algorithm, searchers plan paths by enumerating all possible strategies to a finite-horizon. The robots do so sequentially, which allows for linear scalability in the number of searchers. Planning sequentially and sharing information is a form of implicit coordination because the robots do not explicitly plan for their teammates. We showed that this approximation algorithm yields near-optimal strategies for efficient search.

It is possible to 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 cells to become dirty). If the searchers plan sequentially and share their paths, this leads to a piggyback effect during which each successive searcher extends the clearing schedule further in the environment. By interleaving planning and execution on the receding horizon, the searchers can find a clearing schedule in this manner.

Such an algorithm is an application of simple implicit coordination to the guaranteed search domain. Unfortunately, simple implicit coordination can perform poorly because it spreads the searchers out and requires a large number of stationary guards. This stems from the requirement

Algorithm 1 Edge labeling for trees

Input: Tree $T(N, E)$, Start node $b \in N$
$A \leftarrow N \setminus b$
while $A \neq \emptyset$ **do**
  $l \leftarrow$ any node in A with exactly one unlabeled edge
  if l is a leaf **then**
    $e \leftarrow$ only edge of l
    $\lambda(e) = 1$
  else
    $e \leftarrow$ unlabeled edge of l
    $e_1, \ldots, e_d \leftarrow$ labeled edges of l
    $\lambda_m \leftarrow \max\{\lambda(e_1), \ldots, \lambda(e_d)\}$
    if multiple edges of l have label λ_m **then**
      $\lambda(e) \leftarrow \lambda_m + 1$
    else
      $\lambda(e) \leftarrow \lambda_m$
    end if
  end if
  $A \leftarrow A \setminus l$
end while
Output: Edge labeling $\lambda(E)$

of tight coupling between the searchers' actions. In other words, the searchers must work together to make any progress in clearing the environment.

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. The algorithm presented below provides an alternative method for improving the performance of implicit coordination by exploiting the relatively easy problem of finding a guaranteed search schedule on trees.

4.2 Guaranteed Search on Trees

A linear-time algorithm for guaranteed edge search on trees was given by Barrière et al. (Barrière et al., 2002). This section describes an implementation of this algorithm that does not require recursion. Eliminating the need for recursion allows for the direct application of implicit coordination (see Section 5). This section also proves that the tree search algorithm applies directly to the situation in which the target hides on the nodes (node search).

Assume that the starting node of the searchers is known and the same. Label this starting location as $s \in N$. First, label the edges on the tree $T(N, E)$ with $\lambda : E \rightarrow Z^+$ as in Algorithm 1. 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

Algorithm 2 Guaranteed search schedule for trees

Input: Tree $T(N, E')$, Edge labeling $\lambda : E' \rightarrow Z$, Start node $b \in N$
Define $s_k(t)$ as the node occupied by the $k \leq \mu$ searcher at time t
$t \leftarrow 0$
Let $s_k(t) \leftarrow b$ for all searchers
Mark all nodes except b as uncleared
while some nodes are uncleared **do**
  $t \leftarrow t + 1$
  for all searchers k **do**
    if searcher cannot move without recontamination **then**
      $c \leftarrow s_k(t-1)$
    else if a positive edge label exists incident to current cell **then**
      $c \leftarrow$ cell reached through *lowest* labeled edge
      Decrement $\lambda(e)$ of edge traversed
    else
      $c \leftarrow$ cell reached through negative labeled edge
      Increment $\lambda(e)$ of edge traversed
    end if
    $s_k(t) \leftarrow c$
  end for
  Mark nodes occupied at time t as cleared
end while
Output: Searcher paths s_k for all k, clearing time t

back up the tree after clearing. Refer to the set of edges and their doubles as E'.

An optimal edge search strategy can be generated from this labeling in a distributed manner. Algorithm 2 describes how to generate such a strategy. Algorithm 2 is equivalent to the recursive algorithm in prior work, and it will clear the edges of tree T 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.

4.3 Edge Clearing and Node Clearing

As already remarked, we are interested in *node* clearing. However, Algorithm 2 is an adaptation of Barriere's algorithm, which performs *edge* clearing. Hence we must show that Algorithm 2 (and also Algorithm 4) also performs node clearing.

Let us define two *search games*: the *edge game* and the *node game*. Both games are played in discrete time $t = 0, 1, 2, ...$ on a graph $G = (N, E)$ by a team of searchers who may perform the following moves.

1. At $t = 0$, K searchers are placed in the same node $u_0 \in N$ (u_0 will be called the *base node*).

2. At every $t \in \{1, 2, ...\}$ any number of searchers may slide from node u to node v.

We can fully define a *search strategy* $\mathbf{S}$ (of finite length t_f) by specifying the base node u_0, the number of searchers K and the moves (edge slides) at times $t = 1, 2, ..., t_f$. We will write a move as

$u \to v$ (where $u, v \in N$ and $uv \in E$) thus specifying the edge being traversed and also the direction of the traversal. Note that, given a graph G, the same search strategy can be used in an edge game and a node game.

The *clearing rules* are different for the edge and the node game .

1. In the edge game all edges start dirty; an edge is cleared when traversed and becomes dirty again (recontaminated) if there is a free path from it to a dirty edge.

2. In the node game all nodes start dirty; a node is cleared when entered and becomes dirty again (recontaminated) if it is unguarded and there is a free path from it to a dirty node.

It will be useful to define clear / dirty nodes in the edge game: we say a node is *e-dirty* (edge-game dirty) if it is unoccupied and adjacent to a dirty edge; otherwise it is *e-clear*. Dirty and clear nodes in the node game will be called *n-dirty* and *n-clear*, to stress the difference from e-dirty / e-clear.[2]

In the edge game we denote by $E_C(t)$ the set of edges clear at time t and by $N_C(t)$ the set of nodes e-clear at time t.

We will study a restricted family of search strategies: they must be such that they satisfy (in the edge game) the following.

1. *Monotonicity*: for all t, we have $E_C(t-1) \subseteq E_C(t)$.

2. *Connectedness*: for all t, $(N_C(t), E_C(t))$ is a connected subgraph of G.

In addition, to avoid certain trivial cases, we will require that at time $t = 1$ the strategy $\mathbf{S}$ clears an edge.

Theorem 1 *For every graph G, if a monotone and connected search strategy $\mathbf{S}$ is edge clearing in the edge game, it is also node clearing in the node game.*

We give a proof sketch of Theorem 1 in the Appendix. We also show in the Appendix that Algorithm 2 and hence our full algorithm (Algorithm 4) always provide node clearing strategies.

It is important to note that the converse of Theorem 1 is not true. Any node search strategy is not necessarily an edge search strategy because the target can hide in the edges even if the nodes are clear. It is also important to note that an optimal edge search strategy is not necessarily an optimal node search strategy.

Algorithm 2 restricts the movement of the searcher to moves that do not recontaminate nodes in the tree. Node recontamination occurs if a searcher leaves a node unguarded and one or more of its adjacent nodes are uncleared (with the exception of the node the searcher is moving into). Node recontamination can improve the search number for connected search on arbitrary graphs (Yang et al., 2004). However, situations in which recontamination helps are exceedingly rare in realistic environments.

[2] It will not be necessary or useful to define the concept of clear / dirty edges in the node game.

Algorithm 3 Randomized depth-first search spanning tree algorithm

```
Input: Graph G(N, E), Start node b ∈ N
V, S, B ← ∅
x ← b
while some edges are in neither S nor B do
  V ← V ∪ x
  R ← ∅
  for all nodes y adjacent to node x do
    if y ∈ V then
      B ← B ∪ e(x, y)
    else if y not already visited from x then
      R ← R ∪ y
    end if
  end for
  if R is empty then
    x ← parent of x
  else
    Choose random node z ∈ R
    Set parent of z to x
    S ← S ∪ e(x, z)
    x ← z
  end if
end while
Output: Set of tree edges S, Set of back edges B
```

4.4 Random Search of Spanning Trees

The algorithm described above for trees does not apply to arbitrary graphs with cycles because the edge labeling is not possible. 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 3, which gives a randomized depth-first search algorithm for finding a spanning tree. This algorithm focuses the search on trees that have back edges incident to few nodes. This intuitively leads to trees that require fewer guards. Using either of these generation techniques, the spanning tree that produces the best search strategy can be utilized for searching the original graph. We compare these methods for spanning tree generation in Section 5.

Any spanning tree will always contain $|S| = |N| - 1$ tree edges and $|B| = |E| - |N| + 1$ non-tree edges. The maximum number of guards necessary is $|B|$, which can grow to be a very high number in graphs with many cycles. The next section describes how implicit coordination and the temporal nature of a search plan can be exploited to reduce this number.

Algorithm 4 Guaranteed search with tree decomposition

```
Input: Graph G(N, E), Start node b
while time is available do
  Find a spanning tree using Algorithm 3
  Label edges of T(N, S) using Algorithm 1
  Generate μ tree searchers
  while graph is not completely cleared do
    Move tree searchers according to Algorithm 2
    if a tree searcher reaches a node c with incident non-tree edge then
      if guard can move without recontaminating then
        Move guard to node c
      else
        Generate new guard and move to node c
        Increment number of guards
      end if
    end if
  end while
  Record η = number of tree searchers plus guards
end while
Output: Search strategy with lowest η
```

4.5 Temporal Task Allocation

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 2. 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 both 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 4 shows how these observations can be taken into consideration during search.

Note that Algorithm 4 can be modified to more conservatively use 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. To modify the algorithm, simply check to see if any tree searcher is waiting on another to move. If this is not the case, any tree searchers moving along negative edge labels may be assigned as guards. These two extensions are used in Section 5.

Situations also arise in which a surplus number of guards exist, but the algorithm does not currently take advantage of these situations. It is important to note that using more tree searchers than the minimum (or dynamically switching guards to tree searchers) will not improve the search number, though it may improve the clearing time.

The temporal task 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 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.

4.6 Example

Figure 2 shows a simple example of a small house environment. Assume the spanning tree in the moddle of Figure 2 was found. This spanning tree (ignoring the non-tree edges) requires two searchers to clear because a searcher must remain in cell four to prevent recontamination. 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. This is also the minimal number of searchers capable of clearing this example.

Even in this simple example, all spanning trees do not yield a minimal (two searcher) strategy. Consider the right spanning tree in Figure 2. 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 does not yield a minimal search strategy. This motivates the generation of random spanning trees to reduce the number of searchers.

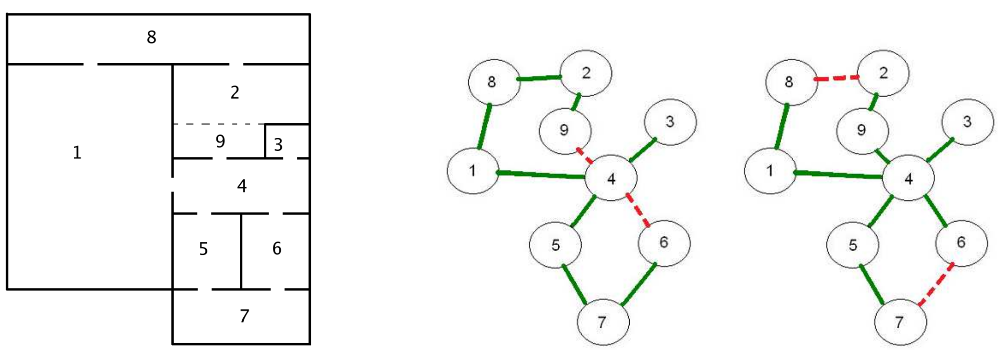

Figure 2: Example discretization of house environment (left) and two example spanning trees of resulting graph (right). Black edges (solid lines) are spanning tree edges, and cyan edges (dashed lines) are non-tree edges. If searchers start in cell three, the middle spanning tree gives a two searcher clearing schedule while the right spanning tree gives a three searcher clearing schedule.

5 Analysis and Results

5.1 Theoretical Analysis

This section gives theoretical analysis regarding the performance of the proposed algorithm and its running time.

The most important characteristic of the search schedules generated by the algorithm is the number of total searchers (guards plus tree searchers) required to clear the graph. Barrière et al. show that the worst-case bound on trees is $\mu \leq \log_2(|N|)$ (Barrière et al., 2002). This is an upper bound on the number of tree searchers needed for a graph with $|N|$ nodes. The proposed algorithm 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 $\eta \leq \log_2(|N|) + |E| - |N| + 1$. It can now be shown that Algorithm 4 will always terminate with a successful clearing schedule (see Theorem 2).

Theorem 2 *Algorithm 4 is guaranteed to terminate on an arbitrary graph $G(N, E)$ with a successful clearing schedule with at most $\eta \leq \log_2(|N|) + |E| - |N| + 1$ searchers.*

Proof Let T_s be the spanning tree generated for G. This generates exactly $|E| - |N| + 1$ non-tree edges in G. The spanning tree requires at most $\log_2(|N|)$ tree searchers to clear (Barrière et al., 2002). If Algorithm 4 must generate a new guard whenever a guard is needed for a non-tree edge, it must generate $|E| - |N| + 1$ guards. This guards all non-tree edges allowing the spanning tree to be cleared. ■

The maximum number of searchers in Theorem 2 can be considered 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 on all graphs examined. With this in mind, the proposed algorithm can be considered an "anytime algorithm" for guaranteed search. 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 algorithm is available, though perhaps one requiring a large number of searchers.

The initial step of generating a spanning tree before guaranteed search serves as a "locker room agreement" (Emery-Montemerlo, 2005) between the searchers. The searchers agree to utilize a shared spanning tree to allocate the tasks of guards and tree searchers. This agreement serves to simplify the problem into one requiring only implicit coordination, the temporal task allocation step.

The running time of every component of the algorithm is linear in the number of nodes $|N|$. Finding a spanning tree using depth-first search only requires visiting each node once to label its edges and is thus $O(|N|)$. As described by Barrière et al., edge labeling and determining search schedules on trees can be done in linear time (Barrière et al., 2002). The search schedules of the guards can also be determined in linear time simply by following the tree searchers along the tree (or with slightly more computation using A^* between guard points). Thus, the computational complexity of the algorithm is $O(\alpha|N|)$, where α is the number of randomly generated spanning trees. In the experiments below, we were able to generate, label, and determine a search strategy for 1500 spanning trees in one second on the 70 cell museum graph.

The computational complexity of the proposed algorithm is limited by the number of random spanning trees that need to be generated. For large graphs, an exponential number of spanning trees are possible. Thus, determining a spanning tree with the minimal number of searchers may be intractable. Our algorithm leverages the fact that many spanning trees will yield good search schedules (though not necessarily minimal ones).

5.2 Simulated Results

We tested our anytime guaranteed search algorithm in simulation on the two complex environments shown in Figure 1. The office environment has two major cycles, and the 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 office can be cleared with three searchers and the museum with five.[3]

We implemented and tested three methods for generating spanning trees. They are described below. Our software is able to generate, label, and search approximately 1500 spanning trees per second using any of the above methods. This demonstrates the low computational complexity of our algorithm.

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.

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 (Section 4): Randomized depth-first search to generate spanning trees. This does not sample the entire space of trees, but it biases sampling towards trees requiring a small number of guards.

Table 1 shows results from the exhaustive search on the office map. Twenty-eight distinct trees yield a minimal (three) searcher schedule on this map. The table also shows the number of trees examined using uniform sampling before finding all minimal trees. Uniform sampling is able to generate all search strategies in less than four seconds. Since depth-first sampling does not search the the entire space of trees, it is only able to find two of the 28 minimal trees. Figure 4 shows a histogram of the number of searchers required for both exhaustive search and depth-first sampling. The histogram demonstrates both the limited number of trees sampled by depth-first sampling and the improved search numbers of those trees. Thus, depth-first sampling heuristically trades off sampling completeness for empirical efficiency.

[3] 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.

Table 2 shows a comparison of uniform sampling and depth-first sampling given a fixed operating time (one minute). Since depth-first sampling focuses on trees requiring fewer guards, it is able to find over sixty times more minimal search strategies than uniform sampling. Furthermore, Figure 4 displays histograms of searchers required to clear trees generated by both uniform sampling and depth-first sampling in the museum. Depth-first sampling clearly generates trees requiring fewer searchers.

Figure 3 illustrates the anytime behavior of our guaranteed search algorithm. This figure shows the best spanning tree found by depth-first sampling and uniform sampling as the number of generated trees is increased. Similarly, Table 3 compares the time to generate a single minimal spanning tree. In both environments, depth-first sampling generates solutions with fewer searchers much more quickly than uniform sampling. This improvement stems from depth-first sampling's bias towards trees requiring few guards.

Figure 5 shows example spanning trees yielding minimal search schedules in the office and museum. The minimal number of searchers is three for the office and five for the museum (determined by inspection). Videos of our guaranteed search results are available online at the following URL.

`http://www.frc.ri.cmu.edu/~gholling/videos/`

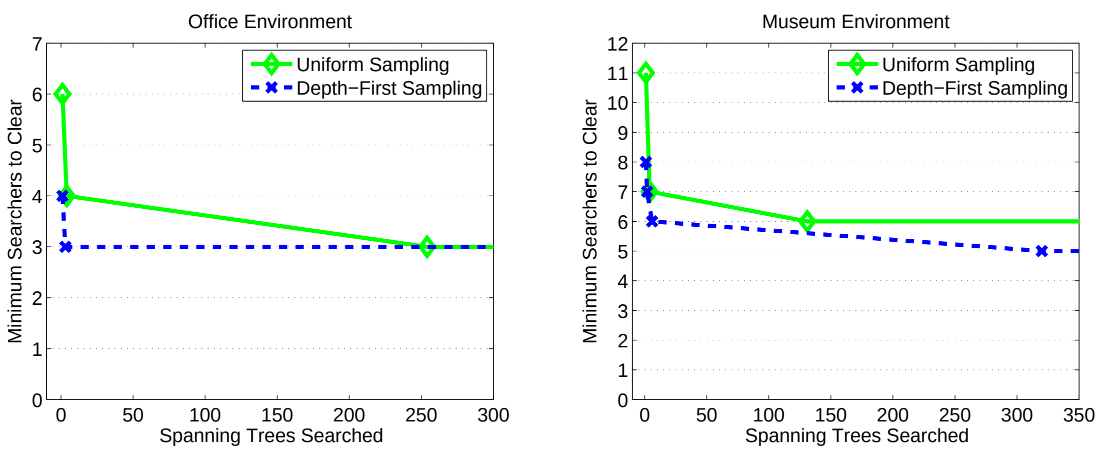

Figure 3: Demonstration of anytime behavior of guaranteed search with spanning trees in the office (left) and museum (right). In both environments, depth-first sampling is able to more quickly generate a spanning tree with few searchers. The exact number of trees searched before finding minimal search schedules are given in Table 3.

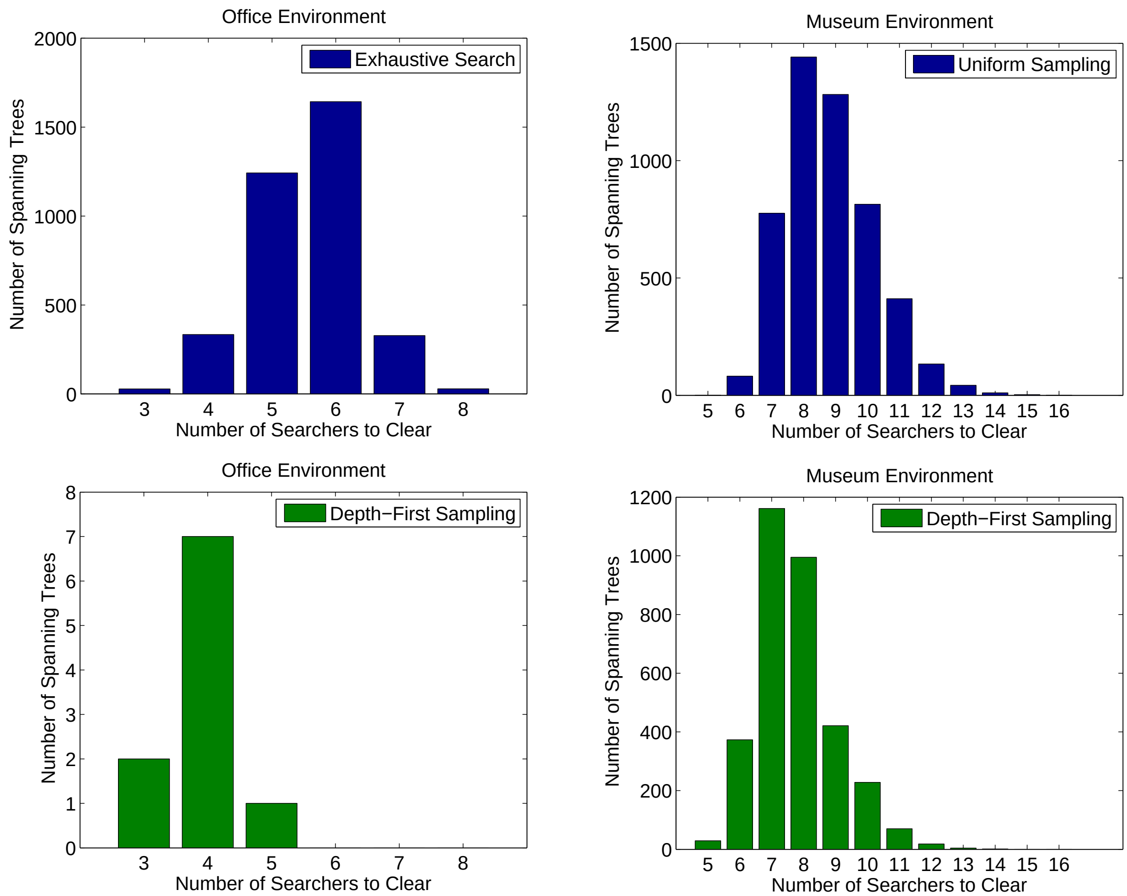

Figure 4: Histograms of number of searchers on different spanning trees for the office (left) and museum (right). In the office, exhaustive search is compared to depth-first sampling. Depth-first sampling greatly limits the space of spanning trees searched but generates trees that require fewer searches. In the museum, uniform sampling of 30,000 trees is substituted for exhaustive search. Again, depth-first sampling generates spanning trees requiring fewer searchers.

Table 1: Comparison of exhaustive search versus randomly generated spanning trees in the office. Uniform sampling covers the entire space of spanning trees. Depth-first sampling covers a limited space and is only able to find some of the minimal spanning trees.

	Time (seconds)	Trees searched	Minimal trees found
Exhaustive Search	1.1	3604	28
Uniform Sampling	3.7	10,981	28
Depth-First Sampling	0.01	13	2

Table 2: Comparison of two methods for randomly generating spanning trees in the museum. Depth-first sampling is able to find sixty times the number of minimal spanning trees as uniform sampling given a fixed running time.

	Time (seconds)	Trees searched	Minimal trees found
Uniform Sampling	60.0	99,357	7
Depth-First Sampling	60.0	106,129	481

Table 3: Comparison of two methods for randomly generating spanning trees in the office and museum. Depth-first sampling more quickly finds its first minimal spanning tree than uniform sampling in both environments.

Office	Time (seconds)	Trees searched	Minimal trees found
Uniform Sampling	0.1	253	1
Depth-First Sampling	0.006	2	1
Museum	Time (seconds)	Trees searched	Minimal trees found
Uniform Sampling	16.7	25,932	1
Depth-First Sampling	0.2	319	1

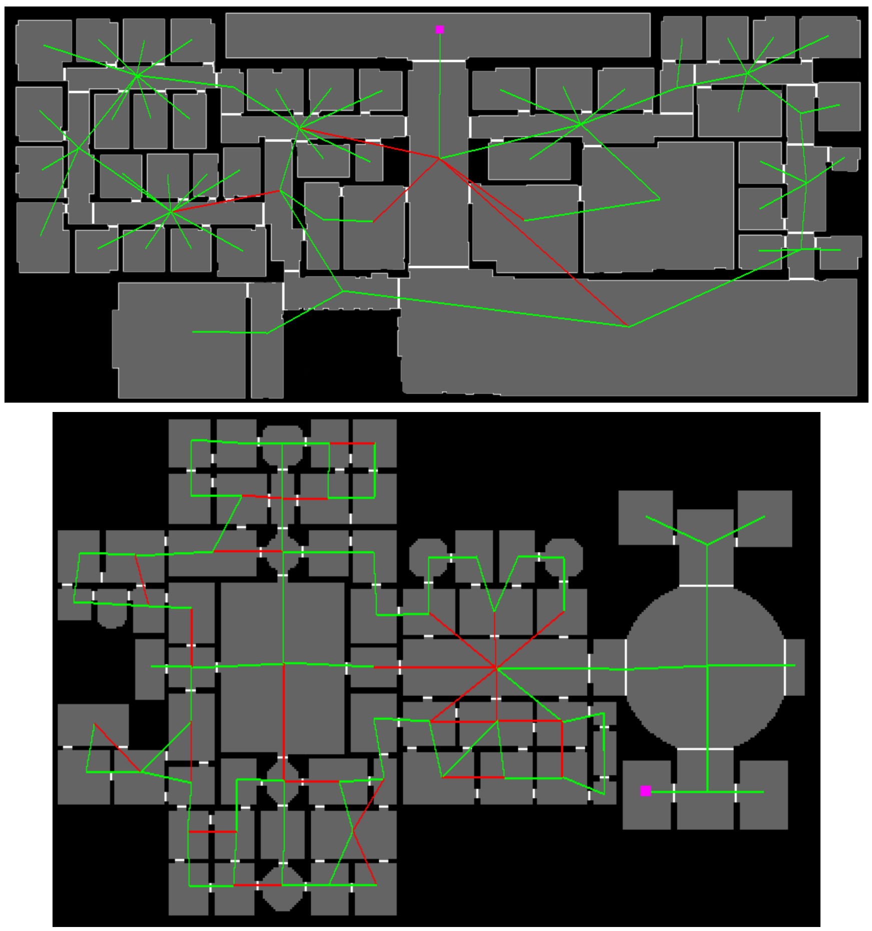

Figure 5: Example spanning trees found for the office (top) and museum (bottom) that yield minimal search schedules. The office spanning tree yields a search strategy with three searchers, and the museum tree yields one with five. Green edges denote edges in the spanning tree, and red edges denote edges that need to be guarded during search. The square denotes the searchers' starting position.

6 Conclusions and Future Work

This technical report has presented an anytime guaranteed search algorithm applicable to complex physical environments. We have shown that simple implicit coordination can generate poor guaranteed search schedules, and an algorithm has been proposed that utilizes a shared pre-processing step before implicit coordination. The proposed algorithm decomposes the environment into an arbitrary graph and then generates spanning trees of that graph. Stationary searchers guard the non-tree edges in the graph allowing the remaining searchers to solve the much easier problem of guaranteed search on trees. We have derived an upper bound on the number of searchers required by the algorithm, and we have demonstrated how implicitly coordinated temporal task allocation can greatly reduce this number. The pre-processing step can be stopped at any time yielding a complete search strategy on the best spanning tree generated thus far. The proposed algorithm runs in time linear in the number of nodes in the graph, which makes it applicable to very large graphical representations of physical environments.

Our proposed algorithm provides search schedules with the minimal number of searchers in two complex, realistic environments. On a large map with many cycles, a minimal search strategy was found in merely 0.2 seconds. We have also demonstrated a depth-first randomized sampling strategy for tree generation that biases sampling towards trees requiring a small number of searchers. These results show that implicit coordination with shared pre-processing can provide solutions to a multi-robot planning problem requiring tightly coupled coordination.

Our algorithm also 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 (Klok, 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.

Our findings open up several interesting avenues for future work. In the current paper, trees were selected randomly using one of two methods. Alternatively, informed heuristics could be used to intelligently generate spanning trees with a high likelihood of yielding low-searcher strategies. The development of such heuristics is left to future work. Furthermore, 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 of bounding the minimum number of searchers from below. The development of such a technique would lead to a bounded approximation algorithm for guaranteed search. Such a technique also opens up the possibility of pruning the space of spanning trees in a branch-and-bound manner.

7 Acknowledgments

The authors gratefully acknowledge Joseph Djugash and Ben Grocholsky for their insightful comments. This work is partially funded by the National Science Foundation under Grant No. IIS-0426945. Further thanks to Intel Research Pittsburgh.

References

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

Barrière, L., Fraigniaud, P., Santoro, N., and Thilikos, D. (2003). Searching is Not Jumping. *Graph-Theoretic Concepts in Computer Science*, 2880:34–45.

Char, J. (1968). Generation of Trees, Two-Trees, and Storage of Master Forests. *IEEE Trans. Circuits Theory*, 15(3):228–238.

Dendris, N., Kirousis, L., and Thilikos, D. (1994). Fugitive-Search Games on Graphs and Related Parameters. In *Proc. 20th Int'l Workshop on Graph-Theoretic Concepts in Computer Science*, pages 331–342.

Emery-Montemerlo, R. (2005). *Game-Theoretic Control for Robot Teams*. PhD thesis, Carnegie Mellon University.

Flocchini, P., Huang, M., and Luccio, F. (2007). Decontamination of Chordal Rings and Tori using Mobile Agents. *Int'l Journal of Foundations of Computer Science*, 18(3):547–564.

Flocchini, P., Huang, M., and Luccio, F. (2008). Decontamination of Hypercubes by Mobile Agents. *Networks*.

Flocchini, P., Nayak, A., and Schulz, A. (2005). Cleaning an Arbitrary Regular Network with Mobile Agents. In *Proc. Int. Conf. on Distributed Computing and Internet Technology*, pages 132–142.

Fomin, F., Fraigniaud, P., and Thilikos, D. (2004). The Price of Connectedness in Expansions. Technical report, UPC Barcelona, LSI-04-28-R.

Fraigniaud, P. and Nisse, N. (2006). Connected Treewidth and Connected Graph Searching. In *Proc. 7th Latin American Symp. on Theoretical Informatics*.

Gerkey, B., Thrun, S., and Gordon, G. (2005). Parallel Stochastic Hill-Climbing with Small Teams. In *Proc. 3rd Int'l NRL Workshop on Multi-Robot Systems*.

Guibas, L., Latombe, J., LaValle, S., Lin, D., and Motwani, R. (1999). Visibility-Based Pursuit-Evasion in a Polygonal Environment. *Int'l Journal of Computational Geometry and Applications*, 9(5):471–494.

Hollinger, G. and Singh, S. (2008). Proofs and Experiments in Scalable, Near-Optimal Search with Multiple Robots. In *Proc. Robotics: Science and Systems Conf.*

Kalra, N. (2006). *A Market-Based Framework for Tightly-Coupled Planned Coordination in Multirobot Teams*. PhD thesis, Carnegie Mellon University.

Kehagias, A., Hollinger, G., and Singh, S. (2008). A Graph Search Algorithm for Indoor Pursuit / Evasion. Technical report, Robotics Institute, Carnegie Mellon University, CMU-RI-TR-08-38.

Klok, T. (1994). *Treewidth: Computations and Approximations*. Springer Berlin / Heidelberg.

Kumar, V., Rus, D., and Singh, S. (2004). Robot and Sensor Networks for First Responders. *Pervasive Computing*, pages 24–33.

LaValle, S., Lin, D., Guibas, L., Latombe, J., and Motwani, R. (1997). Finding an Unpredictable Target in a Workspace with Obstacles. In *Proc. Int'l Conf. on Robotics and Automation*.

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

Megiddo, N., Hakimi, S., Garey, M., Johnson, D., and Papadimitriou, C. (1988). The Complexity of Searching a Graph. *Journal of the ACM*, 35:18–44.

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

Singh, J. S. and Wagh, M. D. (1987). Robot Path Planning using Intersecting Convex Shapes: Analysis and Simulation. *IEEE Journal of Robotics and Automation*, RA-3(2).

Wilson, D. (1996). Generating Random Spanning Trees More Quickly than the Cover Time. In *Proc. 28th ACM Symp. on Theory of Computing*, pages 296–303.

Yang, B., Dyer, D., and Alspach, B. (2004). Sweeping Graphs with Large Clique Number. In *Proc. 5th Int'l Symp. on Algorithms and Computation*, pages 908–920.

8 Appendix: Proof of Theorem 1

Theorem 1 *For every graph G, if a monotone and connected search strategy* **S** *is edge clearing in the edge game, it is also node clearing in the node game.*

Proof This Appendix gives a sketch of the proof of Theorem 1. The proof uses the following facts (the proofs of which are straightforward but tedious and are omitted).

Fact 1 The strategy **S** is monotonic not only with respect to the edges but also with respect to the nodes, i.e. for every t we have $N_C(t-1) \subseteq N_C(t)$.

Fact 2 Let the t-th move of **S** be $u \to v$. Then either the clear graph remains the same or it is expanded by the node v and the edge uv.

Fact 3 For every t, the set of e-clear nodes $N_C(t)$ is exactly the set of nodes visited by a searcher up to time t.

Fact 4 If at any time t node u is e-clear, node v e-dirty, and there exists an edge uv, then u is guarded at time t.

Now let **S** be an edge clearing strategy of length t_f. We will show inductively that $N_C(t)$ (the set of e-clear nodes at time t) and $\widehat{N}_C(t)$ (the set of n-clear nodes at time t) are equal.

This is certainly true at time $t = 0$, when $N_C(0) = \{u_0\} = \widehat{N}_C(0)$. Suppose it is also true at time t. More specifically, suppose that

$$N_C(t) = \{u_1, ..., u_K, v_1, ..., v_L\} = \widehat{N}_C(t)$$
$$N_D(t) = \{w_1, ..., w_M\} = \widehat{N}_D(t)$$

where the w_m nodes are dirty, the u_k nodes are clear and unlinked to w_m nodes, and the v_l nodes are clear and linked to w_m nodes (and hence also guarded, from Fact 4). Here, clear means both e-clear and n-clear.

At time $t+1$ an edge is traversed. The starting node is either one of the u_k's or one of the v_l's.

1. If it is one of the u_k's, it is not linked to a w_m node. Hence no new edge or node is e-cleared; also, from Fact 1, no node becomes e-dirty. Hence $N_C(t+1) = N_C(t)$. Also, since no w_m node is entered, no previously n-dirty node is n-cleared; and since the v_l's remain guarded, no node becomes n-dirty. Hence $\widehat{N}_C(t+1) = \widehat{N}_C(t)$. And so $N_C(t+1) = \widehat{N}_C(t+1)$.

2. If the starting node is one of the v_l's, and it contained more than one searcher, then all of the v_l's remain guarded, so there is no free path between $\widehat{N}_C(t)$ and $\widehat{N}_D(t)$. So all the nodes of $\widehat{N}_C(t)$ remain both e-clean and n-clean. If the ending node is one of the w_m's, then (Facts 2 and 3)

$$N_C(t+1) = N_C(t) \cup \{w_m\} = \widehat{N}_C(t) \cup \{w_m\} = \widehat{N}_C(t+1);$$

otherwise

$$N_C(t+1) = N_C(t) = \widehat{N}_C(t) = \widehat{N}_C(t+1).$$

3. Next suppose the starting node is one of the v_l's (without loss of generality let it be v_1), the ending node is one of the w_m's (without loss of generality let it be w_1) and v_1 contains a single searcher (so after the move it becomes unguarded). The ending node is added to both $N_C(t+1)$ and $\widehat{N}_C(t+1)$. No previously e-clear node can become e-dirty, because (Fact 1) the searching strategy is node-monotonic. Also, it is not possible for a previously n-clear node to become n-dirty: the only possible path from one of $w_2, ..., w_M$ to one of the previously n-clear nodes will have to pass through v_1 and include one previously clear edge (v_1 cannot be isolated from the *connected* clear graph $G_C(t)$); but the search strategy was assumed edge-monotonic and so no edge recontamination is possible.

4. The final case to examine is when the starting node is one of the v_l's, the ending node is one of the u_k's or v_l's and v_l is left unguarded. We omit a detailed treatment, because it is similar to the previous one.

Hence we conclude that $N_C(t+1) = \widehat{N}_C(t+1)$. Proceeding inductively, we finally conclude that, if **S** is an edge clearing strategy of length t_f, then $N_C(t_f) = N = \widehat{N}_C(t_f)$, i.e. **S** is a node clearing strategy as well. ■

Corollary 1 *For every tree T, Algorithm 2 generates a node clearing strategy.*

Proof As proved by Barriere et al. (Barrière et al., 2002), the search strategy generated by Algorithm 2 is monotone, connected, and edge clearing for every tree. Hence, from Theorem 1, it is also a node clearing strategy for every tree. ■

Theorem 3 *For every graph G, Algorithm 4 generates a node clearing strategy.*

Proof Take an arbitrary graph G, find a spanning tree T, and apply Algorithm 4. Since T contains all the nodes of G, the edge clearing strategy of Algorithm 2 will clear all the nodes of T and hence also of G, *provided no node recontamination occurs.* Preventing recontamination is exactly what the use of guards ensures in Algorithm 4. ■