

# A graph search algorithm for indoor pursuit/evasion

Athanasios Kehagias<sup>a,\*</sup>, Geoffrey Hollinger<sup>b</sup>, Sanjiv Singh<sup>b</sup>

<sup>a</sup> Aristotle University of Thessaloniki, Faculty of Engineering, Box 464, Division of Mathematics, Department of Math., Phys. and Comp. Science, Thessaloniki, GR 54124, Greece

<sup>b</sup> Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

## ARTICLE INFO

### Article history:

Received 7 October 2008

Received in revised form 14 June 2009

Accepted 19 June 2009

### Keywords:

Pursuit evasion

Graph search

Robotics

## ABSTRACT

Using concepts from both robotics and graph theory, we formulate the problem of indoor pursuit/evasion in terms of searching the nodes of a graph for a mobile evader. We present the *IGNS* (Iterative Greedy Node Search) algorithm, which performs offline *guaranteed search* (i.e. no matter how the evader moves, it will eventually be captured). Furthermore, the algorithm produces an *internal search* (the searchers move only along the edges of the graph; “teleporting” is not used) and exploits non-monotonicity, extended visibility and finite evader speed to reduce the number of searchers required to clear an environment. We present search experiments for several indoor environments, in all of which the algorithm succeeds in *clearing the graph* (i.e. capturing the evader).

© 2009 Elsevier Ltd. All rights reserved.

## 1. Introduction

In this paper, we present a formulation of *pursuit/evasion* as a *graph search* problem and introduce an algorithm to solve this problem.

In broad terms *pursuit/evasion* (PE) involves a team of  $K$  *pursuers* trying to locate/capture a mobile *evader*. Several more specific variations of the problem can be defined, depending on the capabilities of the pursuers and the evader (and the information available to each of them), the environment in which PE takes place, the type of solution sought and so on. In this paper we are concerned with the following version of the PE problem.

1. The pursuit takes place in an indoor environment (e.g., a house, an office building) and the evader cannot leave the environment (i.e., no exits are available).
2. The evader is *adversarial* (i.e., wants to avoid capture), *arbitrarily fast* and has complete knowledge of the environment map and the pursuer locations.
3. The pursuers know the map of the environment and the locations of their teammates. They do *not* know the evader's position unless it is within their *visibility region*.

What is required is a *search schedule* (i.e., a plan for the movement of each pursuer) which will be computed *offline* (i.e., before the PE process starts) and will *guarantee* the eventual capture of the evader (i.e. every possible sequence of evader moves leads to capture). To produce such a schedule, we examine a *discretized* version of the problem by introducing the following assumptions.

1. The PE process evolves in discrete time steps  $t = 1, 2, 3, \dots$  and, at every time step, movement is *sequential* (i.e., first the evader moves and then one pursuer at a time).

\* Corresponding author.

E-mail addresses: [kehagiat@auth.gr](mailto:kehagiat@auth.gr) (Ath. Kehagias), [geoff.hollinger@gmail.com](mailto:geoff.hollinger@gmail.com) (G. Hollinger), [ssingh@cmu.edu](mailto:ssingh@cmu.edu) (S. Singh).

2. The map of the environment is decomposed into convex *cells* and each cell is represented as a *node* of a *finite graph*; the *edges* of the graph correspond to the connectivity of cells in the original environment; it is assumed that, at every time step, pursuers and evader reside in the nodes of the graph.<sup>1</sup>
3. *Capture* takes place (with certainty) when the evader and at least one pursuer reside in the same node.

We present the *IGNS* (Iterative Greedy Node Search) algorithm which guarantees capture by ensuring that the “dirty set” (the set of nodes which *may* contain the evader) decreases progressively until it reduces to the empty set (i.e., the evader is left without any escape moves)<sup>2</sup> (when this happens we say that the graph has been *cleared*). IGNS combines concepts and methods from two distinct research themes: *graph search* and *robotic pursuit/evasion*.

PE on a graph was first studied by Parsons [1] in connection to *cave searching*. Papadimitriou and his collaborators [2,3] placed the problem in the context of graph theory and initiated a vigorous line of research which continues to this day; for recent reviews see [4,5]. In this context, the problem is usually described as “graph search”, but it should be distinguished from both (a) searching a graph for an *immobile* target and (b) pursuing a *visible* mobile evader (“cops-and-robbers” problems [6,7]).

The graph theoretic work mentioned in the previous paragraph is rather abstract. For example, much of this work is concerned with studying the *search number* of a graph (i.e., the minimum number of searchers required to clear a graph) and establishing its relationship to other graph parameters (e.g., pathwidth [8,9], vertex separation [2] etc.). Another topic of interest is to establish the NP-completeness of the various versions of graph search. Several search algorithms have been presented, but they are complex and not straightforward to implement. Simpler algorithms exist for special types of graphs (e.g., for trees [10]) but the relevance of such graphs to robotic PE is rather limited.

In fact, a major difference between robotic pursuit/evasion and graph search concerns *what* is being searched. Because a cave is essentially a collection of tunnels, in his original formulation Parsons assumed the evader to be hiding in the *edges* of the graph; this version of the problem is the one still studied in most of the graph theoretic literature, under the name *edge search*.<sup>3</sup> However, in robotic indoor PE, we are usually interested in searching *rooms*, and it is natural to model these as *nodes* of a graph.

Furthermore, the search schedules studied in the graph theoretic literature allow a pursuer to move between non-adjacent nodes (i.e., not necessarily following the edges of the graph). We call this “*teleporting*” and obviously, it is not appropriate for robotics problems. Only recently has some research been published [10–12] that concentrates on *non-teleporting* search where the pursuers move only along the graph edges (such a search is called *internal* in the graph theoretic literature). In addition, most graph theoretic search algorithms produce *monotonic* searches (i.e., the dirty set decreases at *every* step of the search). In other words, *recontamination* (of edges and/or nodes) is not allowed. In fact it has been proved [13] that, in teleporting edge search, recontamination does not “help” (does not decrease the search number). However we will give examples which show that sometimes recontamination reduces the number of searchers required to clear the nodes of a graph.

Hence we see that the graph theoretic literature on graph search does not fit all aspects of robotic PE. However, we feel that the graph theoretic point of view has much to offer to robotic PE and that this connection has not been sufficiently explored by robotic researchers. We will elaborate on this point in Section 5.

Let us now turn to robotic PE. The indoor version of the problem was first studied by Lavalley and his collaborators [14–16]; this work was continued by several other researchers [17–21].<sup>4</sup> A very extensive exposition of robotic PE appears in the book [24]. While robotics researchers make use of the graph representation, apparently they have not made much use of the graph theoretic research mentioned in the previous paragraphs. Rather, the approach most often used consists of the following steps [14].

1. The indoor environment is discretized into cells, each of which corresponds to a node of an (undirected) *navigation graph*. The discretization is based on *critical visibility events*.
2. A (directed) *information space graph* is obtained from the navigation graph. The information graph essentially is a *state transition diagram* of the search process.
3. Search consists in finding a path in the *information graph* from the starting state to a *clear* state (using, for example, Dijkstra’s algorithm).

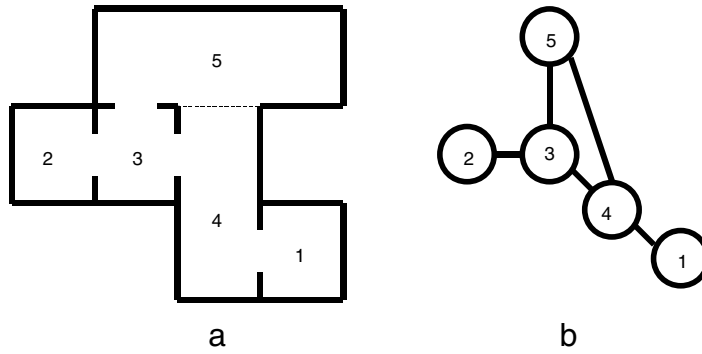
The basic disadvantage of the above approach is that it does not scale well, either with the size of the environment or the number of searchers. Because discretization is based on visibility changes, the navigation graph can involve a large number of nodes, even for simple environments. Furthermore, the size of the information graph grows exponentially with the number of nodes in the navigation graph *and* with the number of searchers. Hence the shortest path computation is viable only for simple environments and one searcher; for more complicated cases, various approximations are used.

<sup>1</sup> This is a naïve way to convert the map to a graph; more sophisticated ways are available but our focus in this paper is on the search algorithm, not on the discretization method.

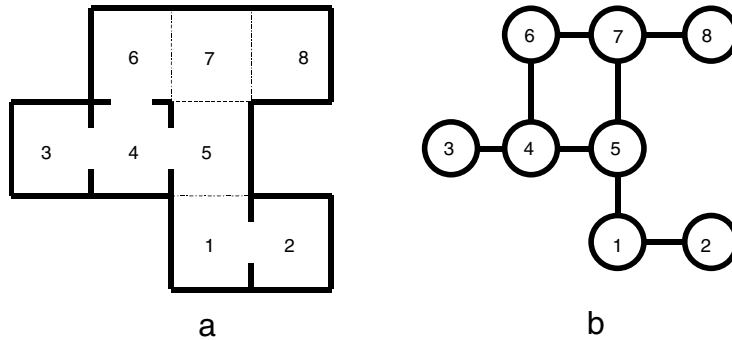
<sup>2</sup> Note that the search schedule is not about the evader per se, but about the dirty set; in fact the search can be executed even in the absence of an evader.

<sup>3</sup> Strangely, in the graph theoretic literature, a variant appears which is named “node search” [2] but again assumes the evader to be hiding in the edges.

<sup>4</sup> All of these papers deal with guaranteed search; “probabilistic” PE, where the evader is assumed to move in a Markovian random manner, has also been studied [19,22,23,21] but will not concern us here.



**Fig. 1.** (a) An indoor environment partitioned into rectangular cells. (b) A graph obtained by corresponding the convex cells of 1(a) to nodes. The graph also includes edges from each node to itself (loops) which are not displayed for simplicity of presentation.



**Fig. 2.** (a) The same environment as in Fig. 1(a), partitioned into different cells. (b) The corresponding graph. Compare with 1(b). The graph again includes non-displayed looping edges.

An approach similar to the one we present here is used in [25] to derive the GSST algorithm; GSST is faster than IGNS and can clear certain environments with a smaller number of searchers but, unlike IGNS, it cannot produce nonmonotonic searches and can only handle *local* visibility and *infinite* evader speed. Kolling and Carpin present another interesting approach to a related problem [26–28]. They deal with what they call *GRAPH-CLEAR*, which requires generating local multi-searcher clearing and guarding strategies and then using this abstraction to formulate a weighted graph search. We leave a more detailed analysis of the connections between GRAPH-CLEAR and our formulation for future work.

In the current paper, we reduce the robotic PE problem to a graph *node* search problem and present the IGNS algorithm, which does not use teleporting and can solve problems involving large indoor environments. Furthermore, IGNS is *not* restricted to *monotonic search*, *local visibility* and *infinite evader speed*; these terms will be further clarified in the next section but we must stress at this point that, as far as we know, *none* of the previously proposed algorithms can handle local visibility and finite evader speed; furthermore most algorithms are restricted to generate monotonic searches; all of these restrictions, which can significantly increase the required number of searchers, are lifted by IGNS.

The paper is organized as follows: in Section 2 we formulate the problem; in Section 3 we present the IGNS algorithm; in Section 4 we present experiments to evaluate the algorithm; in Section 5 we present our conclusions and directions for future research. [Appendix](#) lists certain graphs used in the experiments of Section 4.

## 2. Problem formulation

As already mentioned, we will study PE in indoor environments. Consider the map of one such environment, illustrated in Fig. 1(a). This map can be reduced to a *graph* by discretization. For example, the map of Fig. 1(a) can be reduced to the graph of Fig. 1(b).

A graph  $G = (V, E)$  is described by node set  $V$  and edge set  $E$ . We will always assume the node set has the form  $V = \{1, 2, \dots, N\}$  (and  $|V| = N$ ). The *map-to-graph reduction* is performed by (a) decomposing the environment into convex cells, (b) associating a node with each cell, and (c) inserting edges between nodes which correspond to adjacent cells (an edge also connects each node to itself).

Neither the cell decomposition nor the resulting graph are unique. For example, the map of Fig. 1(a) is reproduced in Fig. 2(a) with a different cell decomposition and the corresponding graph appears in Fig. 2(b).

Cell decompositions of the type described above are quite suitable for robotic pursuit/evasion problems. For example, they correspond to the use of 360° field-of-view sensors as long as the cells in the underlying decomposition are sufficiently

small that a searcher can view the entire cell at once, which would allow for both guarding and clearing the cell. Examples of such sensors are omnidirectional cameras or the combination of rear-facing and forward-facing laser scanners. If the sensors provide a 180° field-of-view (“regular” cameras, laser scanners, and even the human eye—if the searchers are humans), the environment must be discretized as follows: (a) the cells must be sufficiently small such that a searcher on any edge pointing perpendicular to the edge can see the entire cell, and (b) searchers can move between edges while keeping the destination edge in view at all times (determined by robot kinematics/dynamics). Then a “180° search schedule” is generated by having the searchers move between edges while always keeping the destination edge in view. A cell is guarded by a searcher remaining stationary on an edge and pointing its sensor perpendicular to that edge. Note that this does not necessarily require convexity, although convex cells greatly simplify the determination of the two necessary qualities. This extension allows our algorithm to be applied to 180° field-of-view situations.

Given a graph  $G = (V, E)$ , the  $N \times N$  adjacency matrix  $A$  of the graph is defined by

$$A_{mn} = \begin{cases} 1 & \text{iff } (m, n) \in E \\ 0 & \text{else.} \end{cases}$$

The  $N \times N$  visibility matrix  $B$  is defined by

$$B_{mn} = \begin{cases} 1 & \text{iff “} m \text{ is visible from } n \text{”;} \\ 0 & \text{else.} \end{cases}$$

The condition “ $m$  is visible from  $n$ ” holds iff every point of cell  $m$  is visible from every point of cell  $n$  (under straight line visibility) and depends on both the map and cell decomposition from which the graph was obtained.<sup>5</sup>

We assume the PE process evolves in discrete time  $t = 1, 2, \dots$ . The  $K$ -long vector of pursuer positions will be denoted by  $\mathbf{x}(t) = [x_1(t), \dots, x_K(t)]$ , where  $x_k(t)$  denotes the position of the  $k$ -th pursuer at time  $t$ . For  $t = 0, 1, 2, \dots$ ,  $\mathbf{x}(t) \in \mathbf{X} = \{1, 2, \dots, N\}^K$ , the space of (pursuer) configurations. If a node might contain the evader, the node is called *dirty*, otherwise *clear*. A node  $n$  is *cleared* when visited by a pursuer or falls inside the pursuer’s visibility region; it remains clear as long as there is no free (from pursuer inspection) path from it to a dirty node. If, at some time, a free path from the clear node  $n$  to a dirty node  $m$  becomes available, then  $n$  is *recontaminated*. The set of all dirty nodes (resp. clear nodes) is the *dirty set*, denoted by  $D$  (resp. *clear set*, denoted by  $C$ ). We will often describe sets by their *indicator vectors*. For example the indicator vector of  $D$  is  $\mathbf{d} = [d_1, d_2, \dots, d_N]$ , where  $d_n = 1$  iff  $n \in D$ , 0 else; similarly  $\mathbf{c}$  is the indicator vector of  $C$ . The *length* of  $\mathbf{d}$  is  $|\mathbf{d}| = \sum_{n=1}^N d_n$  (note that  $|\mathbf{d}| = |D|$ ); similarly for  $\mathbf{c}$ . If the evader is assumed to be arbitrarily fast and to have complete knowledge of the pursuers’ positions, it will only be captured when it has no escape moves left or, in other words, when the *dirty set becomes empty*. Hence the pursuers must move in such a manner that the dirty set is progressively (but not necessarily *monotonically*) reduced, until it becomes the empty set. The case of finite evader speed is treated similarly, as will be seen presently.

The *state vector*  $\mathbf{z}(t) = [x_1(t), \dots, x_K(t), d_1(t), \dots, d_N(t)]$  contains all the essential information available to the pursuers at time  $t$ . Namely, every pursuer knows the location of itself and its teammates and the *possible* locations of the evader. For  $t = 0, 1, 2, \dots$ ,  $\mathbf{z}(t) \in \mathbf{Z} = \mathbf{X} \times \mathbf{D}$ , where  $\mathbf{X} = \{1, \dots, N\}^K$  (the set of all possible *pursuer configurations*),  $\mathbf{D} = \{0, 1\}^N$  (the set of all possible dirty sets) and  $\mathbf{Z}$  is the *state space*. For every  $\mathbf{z} = (\mathbf{x}, \mathbf{d}) \in \mathbf{Z}$ ,  $\mathbf{x} \in \mathbf{X}$  is a configuration vector and  $\mathbf{d} \in \mathbf{D}$  is a dirty set indicator.  $\mathbf{Z}$  elements of the form  $\mathbf{z} = (\mathbf{x}, \mathbf{0})$  are the *clear states*, i.e. the ones for which the clear set  $C = V$ .

The *control vector*  $\mathbf{u}(t) = [u_1(t), \dots, u_K(t)]$ , denotes the next move of the  $k$ -th pursuer, i.e.  $x_k(t+1) = u_k(t)$ . The *PE state evolution equation* has the form

$$\mathbf{z}(t+1) = F(\mathbf{z}(t), \mathbf{u}(t)), \quad (1)$$

i.e. it yields the next state  $\mathbf{z}(t+1)$  in terms of the previous state  $\mathbf{z}(t)$  and the control  $\mathbf{u}(t)$ .

The movement of the evader (or, more precisely, the evolution of the dirty set) is modeled by *max–min matrix multiplication*, denoted by  $*$  and defined as follows: for arbitrary matrices  $P$  (with dimensions  $L \times M$ ) and  $Q$  (with dimensions  $M \times N$ ) the matrix  $R = P * Q$  is defined by

$$R_{ln} = \max_{m=1,2,\dots,M} [\min(P_{lm}, Q_{mn})] \quad \text{for } l \in \{1, 2, \dots, L\} \text{ and } n \in \{1, 2, \dots, N\}.$$

Suppose, for a moment, that the pursuers are removed from the graph and the evader moves with speed  $S$  (i.e., it traverses exactly  $S$  edges per time step). Then, the possible locations of the evader at time  $t+1$  are given by

$$\mathbf{d}(t+1) = \mathbf{d}(t) * \underset{S \text{ times}}{A} * \dots * A. \quad (2)$$

(If the evader is *arbitrarily fast*, then set  $S = N$ , the number of nodes in the graph.) Now suppose a single pursuer is placed at node  $x_1$ . The pursuer blocks certain paths which, if taken by the evader, would result in capture. In particular, the evader will not move into any node which falls in the visibility region of the pursuer, because it would be captured immediately after

<sup>5</sup> The visibility matrix can actually be designed so as to incorporate many additional characteristics of a particular environment (e.g., limited range sensors, asymmetric visibility etc.).

its move. Similarly, the evader will not move out of any node in the visibility region, because it would have been captured immediately *before* the move. Similar rules hold in case more than one pursuer is located in the graph. Let us then introduce the *modified adjacency matrices*  $\bar{A}(\mathbf{x})$ :

$$\bar{A}_{ij}(\mathbf{x}) = \begin{cases} A_{ij} & \text{if } i \notin V(x_k) \text{ and } j \notin V(x_k) \text{ for any } k \in \{1, 2, \dots, K\}; \\ 0 & \text{if } i \in V(x_k) \text{ or } j \in V(x_k) \text{ for some } k \in \{1, 2, \dots, K\}. \end{cases}$$

In words, we obtain  $\bar{A}(\mathbf{x})$  by removing from the graph the nodes in the visibility region of the pursuer (and all their incident edges).

Putting all of the above pieces together, the PE state evolution equation for  $K$  pursuers and evader speed equal to  $M$  is

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{u}(t) \\ \mathbf{d}(t+1) &= \mathbf{d}(t) * \bar{A}(\mathbf{u}(t)) * \dots * \bar{A}(\mathbf{u}(t)) \\ &\quad \text{\scriptsize $S$ times} \end{aligned} \quad (3)$$

subject to:  $(x_k(t), u_k(t)) \in E$  for  $k = 1, 2, \dots, K$ .

We now introduce a *cost function* which is nonnegative and takes the value zero exactly when the evader is captured (or, more precisely, when its location is known with certainty). The cost function is simply the cardinality of the dirty set,  $J = |\mathbf{d}(t)|$ . Indeed  $J = 0$  iff the graph is clear. We write  $J(\mathbf{u}(1), \dots, \mathbf{u}(t))$ : as the pursuers apply a search schedule (described by  $\mathbf{u}(1), \dots, \mathbf{u}(t)$ ) the size of the dirty set changes, so at time  $t$  we have  $J(\mathbf{u}(1), \dots, \mathbf{u}(t)) = |\mathbf{d}(t)|$ .

Finally, we introduce *distance of pursuer configuration from dirty set*, denoted by  $\text{dist}(\mathbf{x}, \mathbf{d})$  (where  $\mathbf{x}$  is a pursuer configuration and  $\mathbf{d}$  the indicator vector of a dirty set). Recall that the distance of nodes  $m, n$  in a graph, is the length of the shortest path joining  $m$  and  $n$ ; denote this by  $\text{dist}(m, n)$ . Second, define  $\text{dist}(U, W)$  (the distance between *sets* of nodes  $U$  and  $W$ ) as follows

$$\text{dist}(U, W) = \sum_{m \in U} \sum_{n \in W} \text{dist}(m, n).$$

Now, given  $\mathbf{x}$  and  $\mathbf{d}$ , denote by  $X$  the set of nodes occupied by the pursuers (corresponding to  $\mathbf{x}$ ) and by  $D$  the dirty set (corresponding to  $\mathbf{d}$ ) and define  $\text{dist}(\mathbf{x}, \mathbf{d}) = \text{dist}(X, D)$ .

### 3. The IGNS algorithm

Suppose that the pursuers start at an initial state  $\mathbf{z}_{\text{in}} = (\mathbf{x}_{\text{in}}, \mathbf{d}_{\text{in}})$ . We want to move the pursuers through states  $\mathbf{z} = (\mathbf{x}, \mathbf{d})$  so that the 1's in the  $\mathbf{d}$  part progressively decrease (the dirty set shrinks) until eventually we reach one of the clear states  $\mathbf{z}_{\text{fin}} = (\mathbf{x}_{\text{fin}}, \mathbf{d}_{\text{fin}})$ , where  $\mathbf{d}_{\text{fin}} = \mathbf{0}$ . This can be seen as a *shortest path* problem, in the *directed* state transition graph of the PE process. Theoretically we could find a shortest path using, for example, Dijkstra's algorithm. But this approach is computationally intractable even for moderate values of  $N$  and  $K$ . For example, with  $N = 10$  and  $K = 2$  we have  $|\mathbf{Z}| = 10^2 \times 2^{10} = 102\,400$ . Building the state transition graph is computationally taxing; finding a shortest path is computationally prohibitive. The IGNS algorithm reduces the problem to a manageable size by performing an *iterative* search of  $\mathbf{Z}$ , examining, in every iteration, the *successors* of already examined states; at the same time *pruning* is used to reduce computational burden. A pseudocode listing of IGNS is presented in the next page, for the case of two searchers; extension to  $K$  searchers is straightforward.

The following notations are used in the pseudocode representation:

1. the *path vector*  $\mathbf{p}^{(n_1, n_2)}$  has the form  $(i_1 i_2 \dots i_T, j_1 j_2 \dots j_T)$  where  $i_1 i_2 \dots i_T$  is the path of the first pursuer, and  $j_1 j_2 \dots j_T$  is the path of the second pursuer;
2. the components of  $\mathbf{p}^{(n_1, n_2)}$  are denoted by  $\mathbf{p}_1^{(n_1, n_2)} = i_1 i_2 \dots i_T$  and  $\mathbf{p}_2^{(n_1, n_2)} = j_1 j_2 \dots j_T$ ;
3. the *cost* associated with configuration  $(n_1, n_2)$  is denoted by  $v^{(n_1, n_2)}$ ;
4. for every set of nodes  $A$ ,  $\text{Ind}(A)$  denotes the indicator vector of  $A$ .

The following remarks explain the operation of the algorithm. The *active list*  $Q$  is the set of *candidate* solutions and contains triples of the form  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$ ;  $\mathbf{p}$  is the path followed by the pursuers to reach final configuration  $\mathbf{x}$  and dirty set  $\mathbf{d}$ . At every iteration of the algorithm, and for every triple  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  we extend the pursuer configuration  $\mathbf{x}$  (considering for each pursuer every possible move from its current position) and compute the new pursuer configuration and new dirty set (and also an expanded path). This results in a new triple  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$ , which *may* be stored in  $Q$ —because some *pruning* is also applied, as will now be explained.

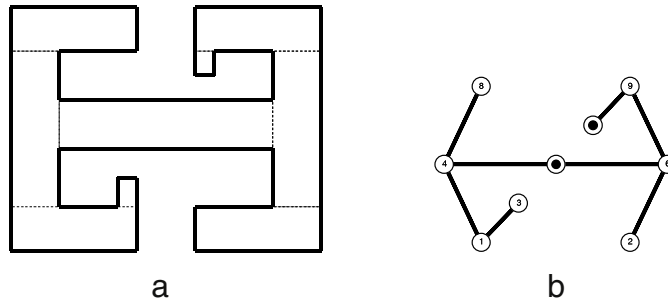
The basic pruning device we use is that stored triples  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  are *indexed* by their  $\mathbf{d}$ 's; in other words at most one triple with a particular  $\mathbf{d}$  is stored. Hence, if an  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$  with a previously unseen  $\hat{\mathbf{d}}$  is obtained, it will be stored in  $Q$ . If, on the other hand some  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  is already stored and now we reach  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$  with  $\hat{\mathbf{d}} = \mathbf{d}$ , then one of the two triples will be stored and the other discarded. Which triple to store is decided by a *heuristic* criterion, which involves the distance function: if  $\text{dist}(\mathbf{x}, \mathbf{d}) > \text{dist}(\hat{\mathbf{x}}, \hat{\mathbf{d}})$  then we replace  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  by  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$ ; else we retain  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  and drop  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$ . The rationale is that, between two states which have the same dirty set, the one in which the pursuers are closer to the dirty nodes is preferable.

---

**Input:** Graph  $G = (V, E)$ , visibility matrix  $B$ , start nodes  $i, j$ , evader speed  $S$ , no. of iterations  $M$ .

end for

Note that, while the cost of a specific configuration is a monotonic function of algorithm iteration, *it does not have to be a monotonic function of path length* (i.e., *recontamination* is possible). The algorithm terminates when some configuration  $(n_1, n_2)$  achieves zero cost ( $v^{(n_1, n_2)} = 0$ ) (or when  $M$ , the maximum number of iterations, is reached).



**Fig. 3.** (a) Lavalley-1 floorplan. The discretization cells are indicated with dashed lines. (b) The graph resulting from the floorplan and discretization of Fig. 3(a). Dots indicate the final positions of the two pursuers.

Some additional heuristics can be incorporated in IGNS. The first two heuristics are intended to reduce computation time by further pruning the list of stored triples  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$ ; the third heuristic reduces the probability of the algorithm getting trapped into nonclearing paths.

1. The algorithm parameter  $L_0$  is *maximum list size*; if the number of elements of  $Q$  grows to more than  $L_0$  elements, we retain only the  $L_0$  best elements (“best” meaning the ones of lowest cost).
2. The algorithm parameter  $v_0$  is *cost tolerance*; we only store triples with cost  $v$  no greater than  $v_{\text{Best}} + v_0$  (since  $v_{\text{Best}}$  changes during execution, previously stored triples may be discarded).
3. Given triples  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  and  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$  with  $\mathbf{d} = \hat{\mathbf{d}}$  and  $\text{dist}(\mathbf{x}, \mathbf{d}) = \text{dist}(\hat{\mathbf{x}}, \hat{\mathbf{d}})$  we will replace  $(\mathbf{x}, \mathbf{d}, \mathbf{p})$  with  $(\hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{p}})$  if  $|\hat{\mathbf{d}}| < |\mathbf{d}| + \varepsilon$ , where  $\varepsilon$  is a zero mean, normally distributed random variable with standard deviation  $\sigma$  (and  $\sigma$  is a parameter of the algorithm).<sup>6</sup>

The “full” IGNS, which we have used in the experiments of the next section incorporates the above described heuristics (a pseudocode version is not given here, for economy of space).

IGNS is not complete (the algorithm will not always return a solution) but it is correct (if it does return a solution, it will be a graph clearing solution). The time complexity is  $O(M \cdot L_0 \cdot K \cdot J_{\max})$ , where  $M$  is the maximum number of algorithm iterations,  $L_0$  is the maximum size of the active list of configurations,  $K$  is the number of searchers and  $J_{\max}$  is the maximum number of edges incident to a node. Note that the complexity is linear in the number of searchers despite the exponential expansion of the state-space with each additional searcher. This is made possible because of the serial expansion of the state space (as described above) and allows for scalability to large environments and large numbers of searchers.

#### 4. Experiments

In this section, we present several indoor PE experiments. Each of Sections 4.1–4.5 deals with a specific floorplan, which is searched by IGNS (enhanced with the heuristics mentioned in Section 3).<sup>7</sup> In Section 4.6, using several floorplans, we compare some previously presented algorithms to IGNS; we conclude with a discussion of our results in Section 4.7.

##### 4.1. Floorplan no. 1 (Lavalley-1)

The first floorplan we use has been previously used by Lavalley et al. [16]. It is illustrated in Fig. 3(a) along with a possible discretization which yields *local visibility*; the resulting graph appears in Fig. 3(b). The search algorithm with two searchers starting at node 1 (and  $L_0 = 12$ ,  $v_0 = 0$ ,  $\sigma = 1$ ) yields the following search schedule which clears the graph in 13 moves (computed in 0.13 s).

$$\mathbf{p}_{\text{Best}} = \begin{bmatrix} 1 & 1 & 4 & 4 & 4 & 4 & 8 & 8 & 4 & 5 & 5 & 5 & 5 & 5 \\ 1 & 3 & 3 & 1 & 4 & 5 & 5 & 6 & 6 & 6 & 2 & 6 & 9 & 7 \end{bmatrix}.$$

The graph of Fig. 3(b) has search number 2, but the floorplan of Fig. 3(a) can actually be cleared by a *single* searcher. To see this, we plot the same floorplan with a different discretization in Fig. 4(a). The new discretization yields *extended* (straight-line) visibility. The resulting graph appears in Fig. 4(b).

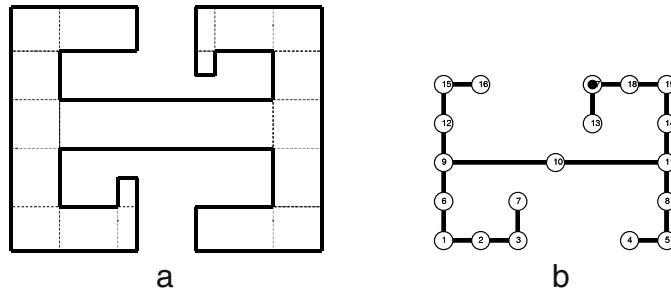
It is not immediately obvious but, using the assumption extended visibility, the graph of Fig. 4(b) can be cleared by *one* searcher. Assuming the searcher starts at node 7 (and using  $L_0 = 40$ ,  $v_0 = 5$ ,  $\sigma = 1$ ) our algorithm finds the following clearing schedule of length 19 (in 0.39 s).

$$\mathbf{p}_{\text{Best}} = [7 \ 3 \ 2 \ 1 \ 6 \ 9 \ 12 \ 15 \ 12 \ 9 \ 10 \ 11 \ 8 \ 5 \ 8 \ 11 \ 14 \ 19 \ 18 \ 17].$$

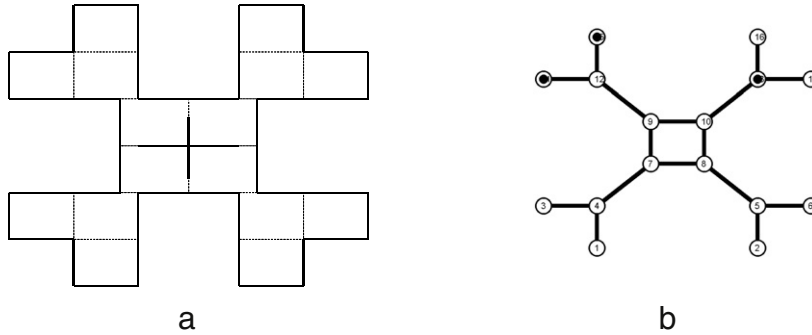
<sup>6</sup> This idea is similar to the stochastic selection of search schedules in Gerkey's Parish algorithm [29].

<sup>7</sup> The algorithm is implemented in Matlab 7.0 and all the experiments reported here were run on a Windows PC with Intel Core Duo CPU E7500 running at 2.93 GHz with 3 GB RAM.





**Fig. 4.** (a) Lavalley-1 floorplan with an alternative, finer discretization. Note that this discretization affords *extended*, straight-line visibility. (b) The graph resulting from the discretization of Fig. 4(a).



**Fig. 5.** (a) Square-Y floorplan and a discretization (cells denoted by dashed lines) which yields local visibility. (b) The graph obtained from the discretization of Fig. 5(a).

Note the use of a large-size list  $Q$  ( $L_0 = 40$ ) and the storage of many suboptimal values ( $v_0 = 5$ ); these choices are necessary because the algorithm must explore a large of state sequences until it finds a clearing path. Hence, we see that appropriate discretization of a floorplan (and use of extended visibility) may reduce the numbers of searchers used to clear the floorplan. Note that for both graphs of this example we assume unbounded (essentially infinite) evader speed.

#### 4.2. Floorplan no. 2 (Square-Y)

In this experiment we use the floorplan of Fig. 5(a). We use the discretization of Fig. 5(a) which yields the graph of Fig. 5(b) (and local visibility). For this graph (in 1.69 s of computation) we find a three searcher clearing schedule of length 51 (parameter values are  $L_0 = 30$ ,  $v_0 = 2$ ,  $\sigma = 1$ ). The *monotone search number* of the graph is *four*; here nonmonotonicity is essential in reducing the required number of searchers by 25%. In this example we again assume unbounded evader speed.

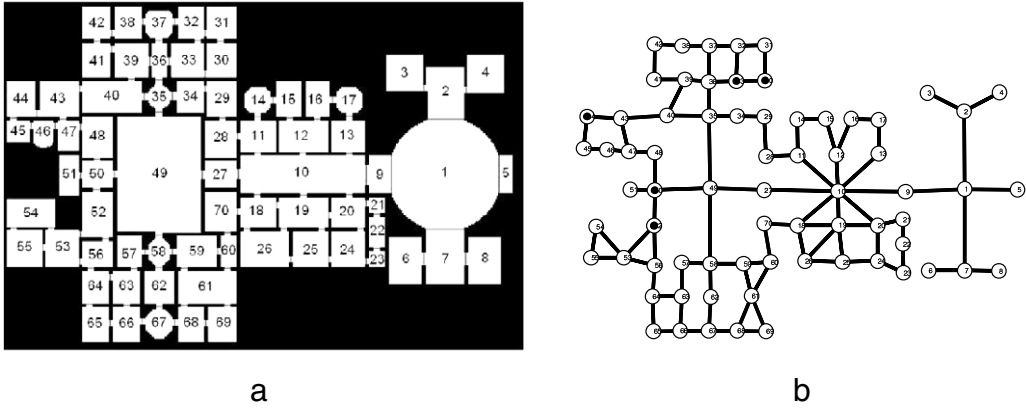
#### 4.3. Floorplan no. 3 (Museum)

In this experiment we use a real-world floor plan, illustrated in Fig. 6(a) (it is the floorplan of a “museum”, namely the National Art Gallery at Washington, DC—this floorplan has been previously used in [19,25]). The discretization used is also illustrated in Fig. 6(a) (it yields local visibility) and the resulting graph in Fig. 6(b). This is a highly complex graph with 69 nodes and approximately  $5.3 \times 10^{14}$  cycles. It is not obvious what the search number of this graph is (a clearing search schedule with 5 searchers has been found in [25]). We have been able to clear the graph with 6 searchers (parameter values are  $L_0 = 5$ ,  $v_0 = 0$ ,  $\sigma = 1$ ); the clearing schedule has length 120 and was computed in 15.82 s. In this example we again assume unbounded evader speed.

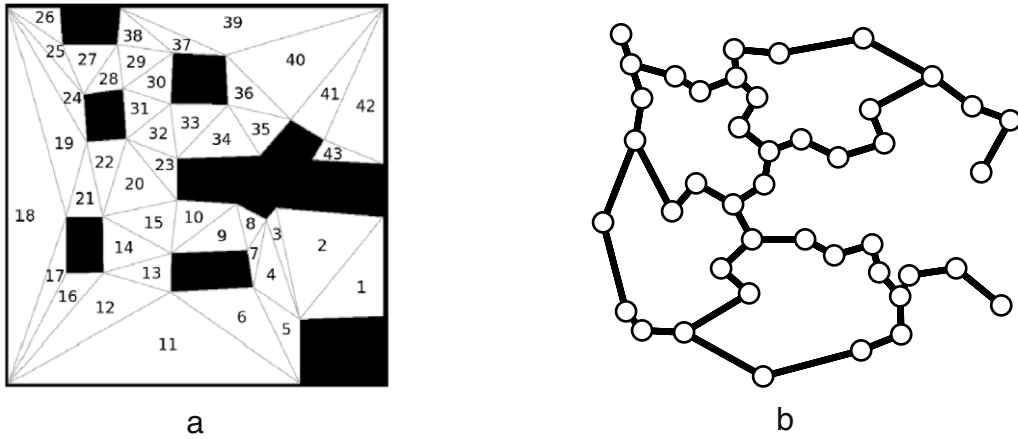
#### 4.4. Floorplan no. 4 (Cave)

The map used in this experiment involves an outdoor environment strewn with irregular obstacles (this map has been previously used in [30] where a four searcher schedule has been found by the Parish algorithm). The discretization used is also illustrated in Fig. 7(a); it is based on a constrained Delaunay triangulation [31]. The resulting graph is shown in Fig. 7(b) (note that, as a result of the discretization by triangles, every node in the graph has degree at most three). We have been able to clear the graph with 3 searchers (using  $L_0 = 1$ ,  $v_0 = 200$ ,  $\sigma = 1$ ) in 4.71 s. In this example we again assume local visibility and unbounded evader speed.





**Fig. 6.** (a) The floorplan of a real world museum. The discretization (which yields local visibility) is denoted by cell numbers. (b) The graph obtained from the discretization of Fig. 6(a).



**Fig. 7.** (a) Cave: An outdoor environment strewn with irregular obstacles. The discretization (which yields local visibility) is denoted by cell numbers. (b) The graph obtained from the discretization of Fig. 7(a).

#### 4.5. Floorplan no. 5 (Tree-1)

This experiment demonstrates the benefit of exploiting (when applicable) finite evader speed. The floor plan illustrated in Fig. 8(a) is discretized as shown in 8(b). In this example we assume extended (straight line) visibility and (unlike the previous examples) *unit speed* for the evader (i.e., he can traverse one edge per time step). Under these conditions (and using  $L_0 = 1$ ,  $v_0 = 10$ ,  $\sigma = 0$ ) we find the following clearing schedule with *one* searcher (in 0.16 s).

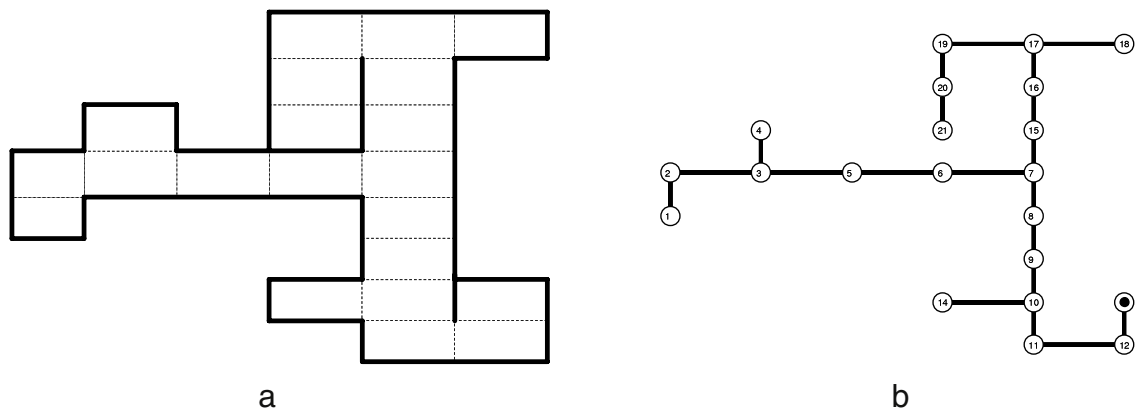
$$\mathbf{p}_{\text{Best}} = [1 \ 2 \ 3 \ 5 \ 6 \ 7 \ 15 \ 16 \ 17 \ 19 \ 17 \ 16 \ 15 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12].$$

Assuming extended visibility and *unbounded* evader speed yields two-searcher clearing at best; if, in addition, monotonicity is required, then three searchers are required to clear the environment.

#### 4.6. Comparative experiments

In this section, we compare IGNS to several other previously proposed graph clearing (pursuit/evasion) algorithms. In particular, we use the FHPE + SA algorithm [19] modified for deterministic search<sup>8</sup> and the GSST algorithm [25]; we also report partial results from the Parish algorithm (since we do not have an implementation of Parish, we simply report results from [29,30]). We perform the comparison on a total of twelve graphs: the first six are the graphs presented in Sections 4.1–4.5; the remaining six graphs are obtained from floorplans described in Appendix. In Table 1 we list the results of the

<sup>8</sup> The FHPE + SA algorithm was designed for probabilistic search. To modify it to perform graph search, we simply replace the finite-horizon probabilistic optimization with a finite-horizon optimization of the number of cells cleared. Having the searchers sequentially perform this optimization on the receding horizon leads to a “piggy-backing” effect that clears the environment.



**Fig. 8.** (a) Tree-1 floorplan. The discretization (which yields local visibility) is denoted by cell numbers. (b) The graph obtained from the discretization of Fig. 8(a).

**Table 1**

Graph search experiments: The minimum number of searchers attained by several algorithms.

Name	Visib.	Ev.Speed	IGNS	FHPE	GSST	Parish
Lavalle-1a	Local	$\infty$	<b>2</b>	<b>2</b>	<b>2</b>	–
Lavalle-1b	Extended	$\infty$	<b>1</b>	2	2	–
Square-Y	Local	$\infty$	<b>3</b>	4	4	–
Museum	Local	$\infty$	6	7	<b>5</b>	–
Cave	Local	$\infty$	<b>3</b>	<b>3</b>	<b>3</b>	4
Tree-1	Extended	1	<b>1</b>	3	3	–
Lavalle-2	Local	$\infty$	<b>3</b>	4	<b>3</b>	–
Office	Local	$\infty$	4	6	<b>3</b>	–
Hallway	Local	$\infty$	<b>2</b>	<b>2</b>	<b>2</b>	<b>2</b>
Gates	Local	$\infty$	<b>3</b>	4	<b>3</b>	5
Tree-2	Extended	1	<b>1</b>	2	2	–
Tree-3	Extended	1	<b>1</b>	2	2	–

comparison. The first column shows the graphs used, the second column the visibility assumption, the third column the evader speed and the remaining columns list the minimum number of searchers achieved by each algorithm. The minimum for each graph (over all algorithms used) is printed in bold face. It can be seen that for every graph (with the exception of “office” and “museum”) IGNS achieves the minimum number of searchers and in many cases significantly outperforms the remaining algorithms. An important reason for this is that our algorithm is able to leverage several assumptions (nonmonotonic search, extended visibility, finite evader speed) which do not fit in the framework of the other algorithms. It is worth repeating that, as far as we know, no other currently available algorithm can incorporate extended visibility and finite evader speed; when these assumptions hold, they can be used (by our algorithm) to significantly reduce the number of required searchers.

An important issue in the comparison of pursuit/evasion algorithms is time performance. Two types of time comparison are possible: *computation time* (i.e., the time required to *compute* a clearing schedule) and *length* of the clearing schedule (i.e., the time required to actually *clear* the graph). However we argue that neither of these comparisons is particularly meaningful in the current context (and hence we omit them) for the following reasons. Regarding execution time, IGNS is currently implemented in Matlab; hence it is significantly slower than, for example, GSST which is implemented in C. We expect that a C implementation would speed up our algorithm significantly, and its linear scalability in all parameters will ensure that it remains tractable in large environments. Regarding length of computed schedules, it is expected (and generally holds true) that, for a given graph, schedules using fewer searchers will require a larger number of moves; hence IGNS, which generally achieves lower search numbers, will also generally produce longer search schedules than the competing algorithms. Let us also stress that the execution time of our algorithm is in the order of seconds, even for complex graphs such as the office and the museum.

#### 4.7. Discussion

From the above experiments we see that that IGNS has cleared every floorplan studied, in most cases using the minimum possible number of searchers. The computation of the clearing schedules usually takes a few seconds or less. While we do not have an exact method to determine optimal parameter values, the general approach we have used is to start with small values of  $L_0$ ,  $v_0$ ,  $\sigma$  and increase these only in case a clearing schedule cannot be found with the smaller values. This approach usually yields a clearing schedule with a few trials. Note that higher values of  $L_0$ ,  $v_0$  require longer (but still in

the order of seconds) computation times. In general, the experiments indicate that the role of discretization can be quite significant: recall the example of Section 4.1 where an appropriate discretization reduces the search number. The choice of discretization and algorithm parameters must be further investigated to establish principled approaches to these issues; the matter will be further discussed in Section 5.

An additional fact indicated by our experiments is the usefulness of assumptions such as non-monotonicity, extended visibility and finite evader speed in reducing the number of searchers required to clear the environment. To our knowledge, IGNS is the first algorithm to exploit these assumptions, which opens the door to improved search algorithms and further theoretical analysis.

Finally, IGNS does not address the issue of *time optimality*; in other words no effort is made to find the shortest possible clearing schedule. This is an issue we plan to address in the future.

## 5. Conclusion

We have presented an approach to robotic indoor PE which is based on (a) discretization of the indoor environment and conversion to a graph, (b) search of the graph. Let us emphasize that the two components (discretization and graph search) are quite independent. We have used two discretization methods: a naive discretization into rectangular nonoverlapping cells and a more sophisticated one using Delaunay triangulation. More importantly, we have introduced IGNS, a greedy iterative algorithm, to perform graph search. *IGNS exploits non-monotonicity, extended visibility and finite evader speed to reduce the number of searchers required to clear an environment.* While the algorithm is not guaranteed to always find a solution, it has done so in all the example problems we have presented, including some quite complex ones. IGNS can be used in conjunction with *any* discretization method. For example, it is compatible to the Lavalley discretization [16,14,15] (which is based on *critical visibility events*).

We believe IGNS is superior to the algorithms previously proposed in the robotic literature [17,16,14,15,21] for guaranteed search; our algorithm has better scaling properties than the rest and hence can clear more complex floorplans (such as the museum of Section 4.3 and the office building of Section 4.6 and). The one algorithm which we find competitive to IGNS is GSST [25], which performs guaranteed search using spanning trees. GSST can also clear complex environments; it will occasionally require fewer searchers than IGNS, but it does not allow for recontamination, cannot handle variable evader speed, and does not allow for extended visibility.

Let us note that IGNS can also be applied to a randomly moving evader, simply by replacing the cost function  $J$  by the *entropy* of the probability distribution of the evader on the nodes of the dirty set.

IGNS is also preferable to the algorithms appearing in the graph theoretic literature, at least as far as robotics problems are concerned. Indeed, as already remarked in the Introduction, graph theoretic search algorithms suffer from several limitations from the from the robotics point of view. They search edges (rather than nodes), they produce monotonic and teleporting search schedules, they cannot handle variable evader speed (they always assume the evader is arbitrarily fast) and assume “local” visibility. IGNS addresses all of these shortcomings.

We conclude this paper by listing some research directions which we intend to pursue in the future.

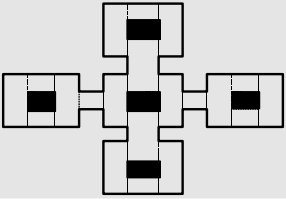
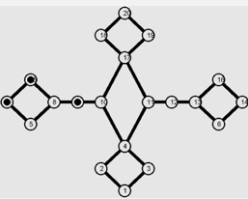

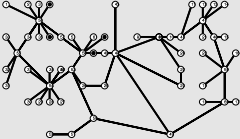
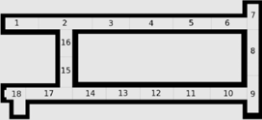
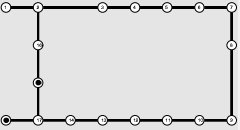
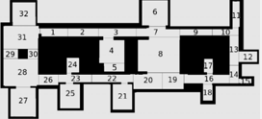
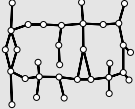
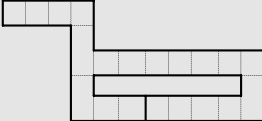

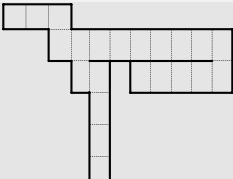
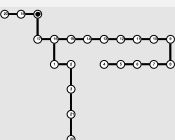
1. We are interested in the *dynamic* adjustment of the number of searchers  $K$ . A simple way to do this would be to start with a single searcher and use it until no further reduction of the dirty set is possible, at which time an additional searcher could be introduced. In this manner, searchers could be introduced, one by one, until the graph is cleared. More sophisticated schemes for controlling the number of searchers may also be devised.
2. Discovering *shortest length* clearing schedules is of obvious importance.
3. We have treated discretization in an *ad hoc* manner, using hand-crafted discretizations. Optimal or, at any rate, principled discretization of the floorplan is of great interest, since appropriate discretization can reduce the number of pursuers necessary to clear the graph.
4. On a more theoretical note, practically all the graph theoretic work done so far considers the problem of the evader hiding in the *edges* of the graph. We have essentially introduced a *new* graph search problem, where the evader is assumed to be hiding in the *nodes* of the graph. Very little is known about the theoretical properties of this problem; we have investigated some of these in a separate publication [32] (for example, we have shown that node search is NP-complete and equivalent to a variant of edge search known as *mixed search*) and we plan to continue this research line in the future.

## Appendix. Floorplans for the comparison of PE algorithms

In this appendix we list the additional environments used in Section 4.6. The environments along with their discretizations and the corresponding graphs appear in Table 2. These environments have been obtained from the following sources: Lavalley-2 is from [24]; Office is the first floor of the Newell–Simon building in the Carnegie Mellon University campus; Hallway and Gates are from [30]; Tree-2 and Tree-3 are two artificial floorplans created by us.

**Table 2**

Additional environments appearing in Section 4.6, along with the corresponding graphs.

Floorplan	Graph	Name
		Lavalle-2
		Office
		Hallway
		Gates
		Tree-2
		Tree-3

## References

- [1] T.D. Parsons, Pursuit–evasion in a graph, in: *Theory and Applications of Graphs*, Springer, 1976, pp. 426–441.
- [2] L. Kirousis, C. Papadimitriou, Searching and pebbling, *Theoretical Computer Science* 47 (1986) 205–218.
- [3] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, C.H. Papadimitriou, The complexity of searching a graph, *Journal of the ACM* 35 (1988) 18–44.
- [4] B. Alspach, Searching and sweeping graphs: A brief survey, *Le Matematiche* 59 (2004) 5–37.
- [5] F.V. Fomin, D.M. Thilikos, An annotated bibliography on guaranteed graph searching, *Theoretical Computer Science* 399 (2008) 236–245.
- [6] G. Hahn, G. MacGillivray, A note on  $k$ -cop,  $l$ -robber games on graphs, *Discrete Mathematics* 306 (2006) 2492–2497.
- [7] R. Nowakowski, P. Winkler, Vertex-to-vertex pursuit in a graph, *Discrete Mathematics* 43 (1983) 235–239.
- [8] H.L. Bodlaender, A tourist guide through treewidth, *Acta Cybernetica* 11 (1993) 1–21.
- [9] H.L. Bodlaender, Efficient and constructive algorithms for the pathwidth and treewidth of graphs, *Journal of Algorithms* 21 (1996) 358–402.
- [10] L. Barriere, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an intruder by mobile agents, in: *Proc. of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA 2002, 2002, pp. 200–209.
- [11] L. Barriere, P. Fraigniaud, N. Santoro, D. Thilikos, Searching is not jumping, in: *Graph-Theoretic Concepts in Computer Science*, in: *Lecture Notes in Computer Science*, vol. 2880, 2003, pp. 34–45.
- [12] L. Barriere, P. Fraigniaud, N. Santoro, D. Thilikos, Connected and internal graph searching, in: *Proc. of 29th Workshop on Graph Theoretic Concepts in Computer Science*, 2003.
- [13] A.S. LaPaugh, Recontamination does not help to search a graph, *Journal of the ACM* 40 (1993) 224–245.
- [14] L.J. Guibas, J.-C. Latombe, S.M. LaValle, D. Lin, R. Motwani, Visibility-based pursuit–evasion in a polygonal environment, *Lecture Notes in Computer Science* 1272 (1997) 17–30.
- [15] L.J. Guibas, J.-C. Latombe, S.M. LaValle, D. Lin, R. Motwani, Visibility-based pursuit–evasion in a polygonal environment, *International Journal of Computational Geometry and Applications* 9 (1999) 471–494.
- [16] S.M. LaValle, D. Lin, L.J. Guibas, J.-C. Latombe, R. Motwani, Finding an unpredictable target in a workspace with obstacles, in: *Proceedings IEEE International Conference on Robotics and Automation*, 1997, pp. 737–742.
- [17] B.P. Gerkey, S. Thrun, G. Gordon, Visibility-based pursuit–evasion with limited field of view, in: *Proc. of the National Conf. on Artificial Intelligence*, 2004, pp. 20–27.
- [18] B.P. Gerkey, S. Thrun, G. Gordon, Visibility-based pursuit–evasion with limited field of view, *International Journal of Robotics Research* 25 (2006) 299–316.

- [19] G. Hollinger, Ath. Kehagias, S. Singh, Probabilistic strategies for pursuit in cluttered environments with multiple robots, in: *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, 2007.
- [20] D. Pellier, H. Fiorino, Coordinated exploration of unknown labyrinthine environments applied to the pursuit–evasion problem, in: *Proc. of AAMAS*, 2005, pp. 895–902.
- [21] A. Sarmiento, R. Murrieta, S.A. Hutchinson, An efficient strategy for rapidly finding an object in a polygonal world, in: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 1153–1158.
- [22] G. Hollinger, S. Singh, Proofs and experiments in scalable, near-optimal search with multiple robots, in: *Proceedings of the Robotics Science and Systems Conference*, 2008.
- [23] V. Isler, S. Kannan, S. Khanna, Randomized pursuit–evasion in a polygonal environment, *IEEE Transactions on Robotics* 5 (2005) 864–875.
- [24] S.M. LaValle, *Planning Algorithms*, Cambridge University Press, 2008.
- [25] G. Hollinger, Ath. Kehagias, S. Singh, D. Ferguson, S. Srinivasa, Anytime Guaranteed Search using Spanning Trees, Carnegie Mellon University, Robotics Institute, Technical Report CMU-RI-TR-08-36, 2008.
- [26] A. Kolling, S. Carpin, The GRAPH-CLEAR problem: Definition, theoretical properties and its connections to multirobot aided surveillance, in: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 1003–1008.
- [27] A. Kolling, S. Carpin, Multi-robot surveillance: An improved algorithm for the GRAPH-CLEAR problem, in: *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008, pp. 2360–2365.
- [28] A. Kolling, S. Carpin, Extracting surveillance graphs from robot maps, in: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2323–2328.
- [29] B.P. Gerkey, S. Thrun, G. Gordon, Parallel stochastic hill-climbing with small teams, in: *Multi-Robot Systems: From Swarms to Intelligent Automata*, vol. III, 2005, pp. 65–77.
- [30] B.P. Gerkey, Pursuit–evasion with teams of robots. <http://ai.stanford.edu/~gerkey/research/pe/>.
- [31] J.R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation, *Computational Geometry: Theory and Applications* 22 (2002) 21–74.
- [32] Ath. Kehagias, G. Hollinger, A. Gelastopoulos, Searching the Nodes of a Graph: Theory and Algorithms, [arXiv:0905.3359v1](https://arxiv.org/abs/0905.3359v1) [cs.DM], 2009.