The GRAPH-CLEAR problem: definition, theoretical properties and its connections to multirobot aided surveillance

Andreas Kolling and Stefano Carpin
School of Engineering
University of California, Merced
Merced, CA, USA

Abstract— **In this paper we present a novel graph theoretic problem, called GRAPH-CLEAR, useful to model surveillance tasks where multiple robots are used to detect all possible intruders in a given indoor environment. We provide a formal definition of the problem and we investigate its basic theoretical properties, showing that the problem is NP-complete. We then present an algorithm to compute a strategy for the restriction of the problem to trees and present a method how to use this solution in applications. The method is then tested in simple simulations. GRAPH-CLEAR is useful to describe multirobot pursuit evasion games when robots have limited sensing capabilities, i.e. multiple agents are needed to perform basic patrolling operations.**

I. Introduction

One of the commonly envisioned applications of multi-robot systems is in the field of surveillance of large areas, like harbors, airports, etc. [1]. In a world where sensors are getting cheaper and sensor networks are foreseen to be soon deployed on a large scale, one could argue that robots do not really offer an advantage. Instead, there are indeed cases where teams of multiple robots are needed. If intruders are being sought in an area not previously equipped with sensors, or if the intruder maliciously or accidentally neutralizes previously deployed sensors, the possibility to have a set of moving sensors mounted on robots offers clear advantages. For economical reasons it is preferred to deploy as few robots as possible when completing such a task. In this paper we investigate the theoretical foundations of this problem. More specifically, we define a new graph theoretical problem that models many situations where robots are used to patrol complex environments. Our perspective differs from previous work on graph searching and pursuit evasion in as much as we take a practical stand with the goal of deploying a team of robots with restricted sensing capabilities. Hence we need multiple robots to execute actions like *clearing a room* or *guarding the door connecting rooms*. In section II we shortly discuss related work, both from the graph-theoretical point of view and from the robotics perspective. Section III describes the problem we investigate, showing the connection between the abstract theoretical formulation and practical situations. Then in section IV we proof that GRAPH-CLEAR is NP-complete and present an algorithm to compute a strategy for trees in section V. Finally, we present an approach to apply the results in practical scenarios in section VI, followed by future work and the conclusion in section VII.

II. Related Research

Due to space limitations, only selected contributions coming from the robotics and graph-theoretic communities will be discussed here. The first instances of the graph-searching problem were discussed by Parsons [2]. It consists of a graph G with *contaminated* edges. The goal is to decontaminate, i.e. to use agents that can execute certain actions turning contaminated edges into *clear* edges. For the problem known as *edge-search* agents can (1) be placed on a vertex, (2) be removed from a vertex or (3) move along an edge. An edge is cleared when an agent moves along the edge and either all edges connected to the start vertex are clear or another agent is placed on the start vertex. A clear edge is re-contaminated if there exists a path from a contaminated edge to this edge which is not containing vertices with agents. The search number $s(G)$ of the *edge-search* problem is the smallest number of agents with which one can find a sequence of actions, called *strategy*, such that all edges become clear. Megiddo et al. in [3] show that determining $s(G)$ is NP-complete by reducing it to the MIN-CUT INTO EQUAL-SIZED SUBSETS problem. Barriere et al. [4] first considered the edge-search problem with weighted vertices and edges. Therein more than one agent could be needed to clear an edge (by moving along it), or more than one agent could be needed to stay in a vertex and break a path leading to recontamination. They also added the restrictions that searchers cannot be removed from the graph and that at any time all cleared edges form a connected subgraph. They show that the general problem is NP-complete on graphs, but can be solved with linear complexity on trees and give an $O(n)$ algorithm to solve this problem.

In the robotics community visibility-based pursuit evasion problems have been considered for a while. The early work by Suzuki and Yamashita [5] investigates how a pursuer can detect intruders using a *flashlight*, i.e. a beam sensor with unlimited range. In a series of papers LaValle and other collaborators investigated different variations of this problem, for example the case where the pursuer has a flashlight-like omnidirectional sensor and is searching for an intruder in a complex polygonal environment [6]. Subsequently, minimalist strategies for simple pursuers were investigated. In [7] it is shown how a single pursuer equipped with a gap sensor, i.e. a sensor only capable of detecting discontinuities, can detect an unpredictable target that moves with unbounded speed in a simply-connected environment. More recently Gerkey et al. [8] address the visibility based pursuit problem considering a robot with limited field of view and showing also results involving practical experiments.

III. Problem Description

Before introducing the problem in formal terms we provide its practical grounding in multi-robot aided surveillance. In case one or more robots are used to detect possible intruders in a given indoor environment, it is convenient to associate a graph with the given environment. Specifically, we associate vertices with rooms (i.e. planar areas), and edges to connections between different rooms, i.e. doors, passages, and so on. We assume that robots are equipped with limited range sensors. For example, more than one robot may be needed in order to assure that no intruder traverses a certain door. Similarly, more than one robot may be needed to sweep a room and detect all possible intruders. Having this scenario in mind, we turn to the formal problem statement. In the sequel we use the terms *agent* and *robot* as synonyms. They both indicate the moving device carrying a sensor that can detect intruders.

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Edges and vertices can be *clear* or *contaminated*. A clear vertex or edge is guaranteed to host no intruders, while a contaminated vertex or edge could potentially hide one or more intruders. G is said to be clear when all its vertices and edges are clear. Contaminated vertices or edges can be cleared by applying the *Blocking* and *Clearing* operations that will be shortly introduced. However, a clear vertex v can become contaminated again if there exists a path from v to another contaminated vertex or edge[1]. A similar definition can be given to describe edge recontamination. The following operations can be applied to edges or vertices:

1) *Blocking* - an action applied to an edge that does not allow recontamination of any edge or vertex through a path using the blocked edge or vertex. A blocked edge becomes clear.
2) *Clearing* - an action applied to a vertex that ensures that all intruders are detected, assuming no new intruders enter or leave the vertex. When this operation is applied the vertex becomes clear.

In order to formally introduce the GRAPH-CLEAR problem, we first define the terminology.

*Definition 1 (*Weighted graph): A weighted graph $G = (V, E, w)$ is an undirected graph $G = (V, E)$ endowed with a weight function $w : V \cup E \to \mathbb{N} \setminus \{0\}$. The weight of a vertex $v \in V$, i.e. $w(v)$, is the number of agents needed to perform a clearing action on that vertex, while the weight of an edge $e \in E$, i.e. $w(e)$, is the number of agents necessary to perform a blocking action on that edge.

When using multiple robots in order to clear an environment, we can deploy robots in edges or vertices in order to perform the blocking and clearing actions. The policy we follow when deploying robots is called *strategy* and is captured by the following definition.

*Definition 2 (*Strategy): Let $G = (V, E, w)$ be a weighted graph. A strategy S on G is a function $S : (V \cup E) \times \mathbb{N} \to \mathbb{N}$.

According to the above definition, if $v \in V$, then $S(v, t)$ is the number of agents deployed on vertex v at time t, while $S(e, t)$ is the number of agents deployed at time t on edge e. Associated with each strategy there is a cost, i.e. the number of agents needed in order to implement the strategy.

*Definition 3 (*Cost of a strategy): Let $G = (V, E, w)$ be a weighted graph, and let S be a strategy on G. The cost of S is

$$ag(S) = \max_{t \in \mathbb{N}} \sum_{x \in V \cup E} S(x, t)$$

We can now define the GRAPH-CLEAR problem.

*Definition 4 (*GRAPH-CLEAR problem): Let $G = (V, E, w)$ be a weighted graph with all edges and vertices contaminated. Determine a strategy S on G that clears G and is of minimal cost.

Let us illuminate a key difference to edge-search. In edge-search an edge is cleared when one of the endpoints is guarded and another agent is moved along the edge to the other endpoint. If the first endpoint is not adjacent to any other contaminated edges one can clear the edge by moving only one searcher along it without keeping the first endpoint guarded. If we think of this in practical terms it imposes a severe restriction. Think of the agent as a mobile sensor that moves from one vertex to another. In classical edge-search it has to ensure that no target can possibly enter the first endpoint during the movement along the edge. Furthermore, once arrived at the second endpoint no target should be able to enter the edge. In our variant this restriction is not present, since for such a movement recontamination will occur. The only way for a team of mobile sensors to travel from one vertex to another, without re-contaminating the first endpoint, is by blocking the edge and while it is blocked, clear and block the second endpoint. Only then the block on the edge can be released. This allows the usage of sophisticated methods for the clearing action on the vertex, such as the one presented in [7] using a single mobile gap detector with sufficient range, without perfect control or any localization capabilities, to clear unknown, simply connected, piece-wise smooth and planar environments. We could, e.g. clear the environment in the center of figure 1 with a single mobile gap-detector while three other sensors observe only the grey rectangular regions. Once done, the gap sensor can move into a neighboring environment. It is hard to imagine how one would guard such complex simply connected environments.

The following simple equation gives the cost to clear a vertex safely, i.e. the cost to perform a clearing operation on the vertex while blocking all the edges connected to it to avoid immediate recontamination.

$$s(v) = w(v) + \sum_{e \in Edges(v)} w(e). \qquad (1)$$

[1] usually a path is defined as a sequence of vertices, while here we consider also edges connecting two vertices in the path. This slight difference could be formalized, but the notation would become heavier, so we prefer to leave this notion at this intuitive level

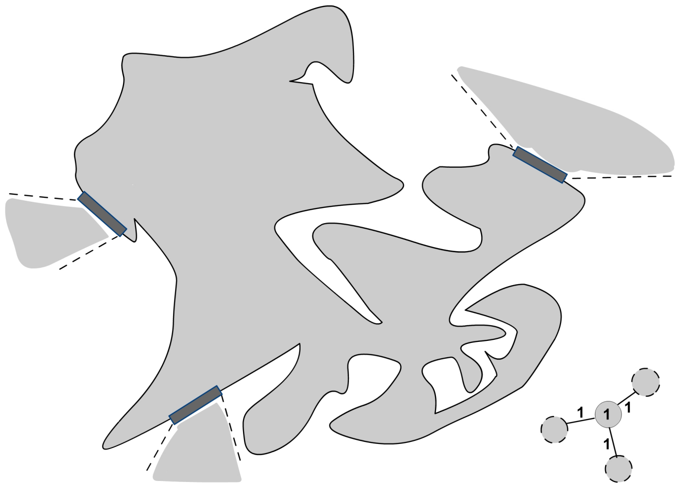

Fig. 1. A complicated environment that can be cleared with a single mobile gap sensor while three other sensors block the rectangular entrances. The corresponding graph with weights is displayed on the lower right. Dotted lines indicate an arbitrary continuation.

Our definition allow us to place agents on multiple vertices in one step. It is worth noting that a strategy clearing no more than one vertex per step needs less agents than one that places agents on more vertices in one step. Hence, since we are looking for strategies of minimal cost, we will restrict our attention to strategies that clear at most one vertex per step. We conclude this section by observing that for a given weighted graph G there can be multiple strategies S of minimal cost. Let $ag(G)$ denote the cost of any optimal strategy for G.

IV. GRAPH-CLEAR IS NP-COMPLETE

The main result of this section is a proof showing that the decision problem associated with the graph clearing problem NP-complete. Formally, we consider the following decision version of the GRAPH-CLEAR problem:

INSTANCE: $G = (V, E, w)$ with $w(x) = 1\ \forall x \in V \cup E$, and a natural number K

QUESTION: is $ag(G) \leq K$?

Theorem 1: GRAPH-CLEAR is NP-complete.

Our proof also relies on a reduction to the MIN-CUT INTO EQUAL-SIZED SUBSETS problem (MCIESS from now on) that is known to be NP-complete [9]. For sake of completeness, we shortly restate the MCIESS problem.

INSTANCE: An undirected graph $G = (V, E)$ with an even number of vertices, and a natural number K.

QUESTION: is there a partition of V into two subsets V_1 and V_2 with $|V_1| = |V_2| = \frac{1}{2}|V|$ such that $|\{u, v\} \in E : u \in V_1, v \in V_2| \leq K$?

The core part of the proof is analogue to the proof of NP-completeness of edge-search on a graph given by Megiddo et al. in [3]. The key difference is that instead of complete graphs we have to use star-shaped graphs, as defined below. We first prove some properties of graph clearing on star-shaped graphs and then proof Theorem 1.

*Definition 5 (*Star shaped graph): A star shaped graph $S_n = (V, E, w)$ is a weighted graph where

- $V = \{v_0, v_1, \ldots, v_n\}$
- there exists a vertex $v_s \in V$ such that $[v_i, v_j] \in E \Leftrightarrow v_i = v_s$ or $v_j = v_s$.
- $w(x) = 1\ \forall x \in V \cup E$.

The vertex v_s is called *center*, while all other vertices are called *singletons*.

*Lemma 1 (*Clearing a star shaped graph): Let S_n be a star shaped graph. Then $ag(S_n) = n + 1$.

Proof: Let $v_0 = v_s$. According to Eq. 1 one needs at least $n+1$ agents to clear the v_s, hence this is a lower bound. Consider the following strategy: first clear v_1 blocking $[v_s, v_1]$; then clear v_2 blocking $[v_s, v_1]$ and $[v_s, v_2]$; next clear v_3 blocking $[v_s, v_1]$, $[v_s, v_2]$ and $[v_s, v_3]$ and so on. Hence the last singleton to be cleared needs n blocks and 1 agent to clear it. Once v_n has been cleared, v_s can be cleared with $n + 1$ agents (n to block the n edges and one to clear it). Hence $ag(S_n) = n + 1$.

*Lemma 2 (*Connectors of stars): Let $S_n = (V, E, w)$ be a star shaped graph with $n \geq 2$, let v_g be a new vertex and let $S_n^c = (V \cup \{v_g\}, E \cup \{[v_c, v_g]\}$, where $v_c \in V$ is a singleton of S_n. Call the v_c a connector of S_n^c. Then $ag(S_n) = ag(S_n^c)$.

Proof: Clearing v_g first requires 2 agents, then clearing v_c requires 3 agents[2]. To maintain v_g and v_c cleared we need to keep edge $[v_s, v_c]$ blocked with weight one. Clearing the remaining vertices is equivalent to clearing S_{n-1}. Hence $ag(S_n) = ag(S_n^c)$. □

*Lemma 3 (*Connecting stars to graphs): Let $S_n = (V, E, w)$ be a star shaped graph $n \geq 2$ and let $G = (V^{'}, E^{'}, w')$ be any graph. Consider the instance of $S_n^c = \{V'', E'', w''\}$ built as follows:

- $V'' = V \cup V^{'}$
- $E'' = E \cup E^{'} \cup \{[v_c, v_g]\}$ where $v_c \in V$ is a singleton of S_n, v_g is any vertex in G
- $w''([v_c, v_g]) = 1$, $w''(x) = w(x)\ \forall x \in V \cup E$, $w''(x) = w'(x)\ \forall x \in V' \cup E'$

Call the v_c a connector of S_n. Then $\max(ag(S_n), ag(G)) \leq ag(S_n^c) \leq \max(ag(S_n), ag(G)) + 1$.

Proof: First clear S_n by clearing all singletons first and then the center. After clearing the center a total of $n + 1$ agents were needed and only the block to v_c has to remain. Then 3 agents suffice to clear the path from the center of S_n to v_g, leaving the edge $[v_c, v_g]$ blocked with one agent. Clearing G will now require exactly $ag(G)$ agents. Hence $ag(S_n^c) \leq max(ag(S_n), ag(G)) + 1$. Clearly $ag(G)$ and $ag(S_n)$ are lower bounds.□

Figure 2 shows a star shaped graph without and with connector.

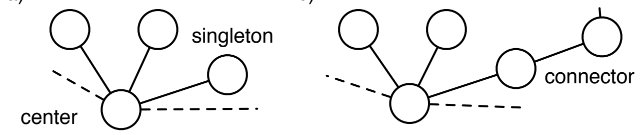

Fig. 2. a) A star shaped graph and b) a star shaped graph with connector.

[2] note that $3 \leq ag(S_n)$ because we assumed $n \geq 2$

*Lemma 4 (*Connecting multiple copies of stars): Let $k \in \mathbb{N}$ and C_i, $1 \leq i \leq m$ be m copies of S_n with $n > k \cdot m$. For each pair C_i, C_j, $j \neq i$ add k edges between singletons to turn them into connectors. Call the resulting graph H. Then $ag(H) = n + 1 + k\lfloor\frac{m-1}{2}\rfloor \cdot (m - 1 - \lfloor\frac{m-1}{2}\rfloor)$

Proof: Trivially, to clear the first C_{n_0} we need at least $n+1$ agents. The following procedure clears C_{n_0} with $n+1$ agents. We start by clearing all singletons in sequence and blocking their edges to the center. Then we block all edges of the center and clear it. This requires $n+1$ agents. Now we can release all blocks on edges between center and singletons while retaining k agents to block the edges connecting the center to the connectors. The remaining contamination in C_{n_0} is now precisely in the connectors whose edges are still blocked. For each connector clear the path until the center of the neighboring C_j. Each time we clear such a path we have to leave precisely one agent blocking the center of the neighboring C_j from its connector leading to C_i. After clearing C_{n_0} we hence reduce the task of clearing all other C_is from clearing an instance of S_n to clearing an instance of S_{n-1}, since the one edge to C_{n_0} will constantly be blocked until the center of C_i is also cleared. At step i let b_i be the number of agents needed for blocks between clear and contaminated C_i's and a_i be the number of agents needed to clear C_{n_i}. Then

$$a_i = n + 1 - i \cdot k \tag{2}$$

$$b_i = i \cdot k(m - i) \tag{3}$$

Now, $ag(H) = \max_{0 \leq i \leq m-1}\{a_i + b_i\}$. This maximum occurs at $i = \lfloor\frac{m-1}{2}\rfloor$ and therefore:

$$ag(H) = n + 1 + k\lfloor\frac{m-1}{2}\rfloor \cdot (m - 1 - \lfloor\frac{m-1}{2}\rfloor) \tag{4}$$

□

With the previous lemmas the key ideas of the proof of NP-completeness for edge-search by Meggido et al. [3] can be adapted to conclude the proof of Theorem 1.

V. Tree clearing

Since GRAPH-CLEAR is NP-complete, let us turn to the restriction of the problem to trees. In [10] we presented an algorithm that computes a strategy S in the special case that G is a tree with a proven upper bound but still with $ag(G) \leq ag(S)$. For the case of trees write T instead of G. Let us restate the algorithm in a more general form with a notation similar to Barriere's notation in [4] where an algorithm to compute a contiguous strategy for weighted edge-search on trees is presented. Let $T = (V, E, w)$ be a tree with a weight function $w : V \cup E \rightarrow \mathbb{N} \setminus \{0\}$. For any edge $e = [v_x, v_y]$ let there be two labels λ_{v_x} and λ_{v_y}. These labels will represent the number of agents needed to clear the subtree rooted in v_y when coming from v_x and vice versa. A label is computed as follows: if v_y is a leaf, then $\lambda_{v_x}(e) = s(v_y) = w(v_y) + w(e)$. Otherwise consider all neighbors of v_y other than v_x. Let these be $v_2, \ldots, v_m$ with $m = degree(v_y)$. Let $e_i = [v_y, v_i]$ and $\rho_i := \lambda_{v_y}(e_i) - w(e_i)$. W.l.o.g. the neighbors are ordered s.t. $\rho_i \geq \rho_{i+1}$. Now define the cost when clearing the subtree at v_i by $c(v_i) := \lambda_{v_y}(e_i) + \sum_{2 \leq l < i} w(e_l)$, i.e. we have to use agents to block all edges to previously cleared subtrees and then use agents to clear the subtree v_i. Now let:

$$\lambda_{v_x}(e) = \max\{s(v_y), \max_{i=2,\ldots,m}\{c(v_i)\}\}. \tag{5}$$

The order defined by ρ_i minimizes this term. It is straight forward to compute the labels in $O(n)$. In [4] Barriere provides details how this can be done in sequential time $O(n)$ and distributively with $O(n)$ messages. Once the labels are computed we can derive a strategy. Consider vertex v and all its neighbors $v_1, \ldots, v_m$ with $m = degree(v)$. The number of agents needed for a strategy $S(v)$ which starts clearing the tree from v is

$$ag(v) = \max\left\{s(v), \max_{i=1,\ldots,m}\{c_{ag}(v_i)\}\right\},$$

where $c_{ag}(v_i) = \lambda_v(e_i) + \sum_{1 \leq l < i} w(e_i)$. It is executed by following the edges in their order, always blocking edges between cleared and contaminated parts of the tree. Once all subtrees of a vertex are cleared the vertex itself will be cleared by blocking all edges and then executing a clearing action on it. Figure 3 shows how such a strategy.

Let us now consider a contiguous version of our algorithm, i.e. one which produces a clearing strategy in which all cleared vertices will always form a connected subtree at every clearing step. To obtain a contiguous strategy we clear and block v_y before we clear all its subtrees. This also means that when we start clearing the first subtree all edges to contaminated subtrees are blocked. It may also become easier to clear the subtree rooted on v_i since the edge to v_y is already blocked. Hence our labels will have to differ slightly. So we define $s^c_{v_x}(v_y) = s(v_y) - w([v_y, v_x])$, i.e. the cost of clearing v_y is reduced by $w([v_y, v_x])$, since this edge is blocked when we enter the subtree rooted at v_y. We shall use the superscript c to denote the contiguous variants of our labels and functions. We also need to order our subtrees differently, i.e. we clear the subtrees in order $v_m, \ldots, v_2$. The contiguous labels are now computed as follows: if v_y is a leaf, then $\lambda^c_{v_x}(e) = w(v_y)$. Otherwise let $c^c(v_i) := \lambda^c_{v_y}(e_i) + \sum_{2 \leq l \leq i} w(e_l)$. $\lambda^c_{v_x}(e) := \max\left\{s^c_{v_x}(v_y), \max_{i=2,\ldots,m}\{c^c(v_i)\}\right\}$. The number of agents needed to clear the tree starting from vertex v

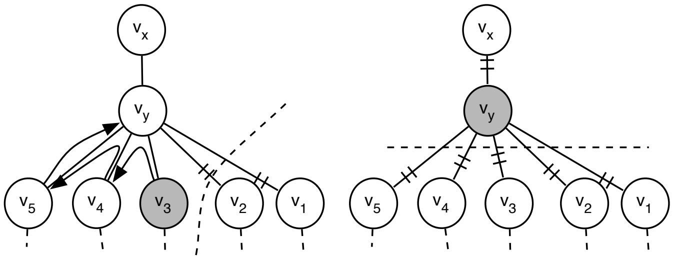

Fig. 3. This figure shows how a noncontiguous strategy on a tree can be executed. The dashed line separates clear and contaminated vertices. Edges are represented by solid lines and blocked when crossed through twice. The grey color indicates the subtree or vertex that is being cleared at the presented step and arrows indicate movement of clearing robots.

now is:

$$ag^c(v) = \max\left\{s(v), \max_{i=1,\dots,m}\{c^c_{ag}\}\right\}.$$

where $ag^c_{ag} = \lambda^c_v(v_i) + \sum_{1\leq l\leq i} w(e_i)$. The following compares the contiguous variant to the noncontiguous one.

Lemma 5: Let $T = (V, E, w)$ be a tree with a weight function $w : V \cup E \to \mathbb{N}/\{0\}$. Then $ag(v) \leq ag^c(v), \forall v \in V$.

Proof: Consider the computation of $\lambda^c_{v_x}(e)$ and $\lambda_{v_x}(e)$ with the same notation as above. If v_y is a leaf, trivially $\lambda^c_{v_x}(e) + w(e) = \lambda_{v_x}(e)$. If not, then consider it neighbors v_i different from v_x. If all v_i are leaves, then either $\lambda^c_{v_x}(e) = \lambda_{v_x}(e)$ or $\lambda^c_{v_x}(e) + w(e) = \lambda_{v_x}(e)$. Now if we have one v_j for which all children are leaves, then by above $\lambda^c_{v_j}(e) = \lambda_{v_j}(e)$ or $\lambda_{v_j}(e) - \lambda^c_{v_j}(e) = w(e_j)$. Therefore $-w(e_i) \leq \lambda^c_{v_x}(e) - \lambda_{v_x}(e) \leq w(e)$. Extending this argument inductively gives us the simple fact that the contiguous label can at most be better by $w(e)$. Since this weight is added in $ag^c(v)$ we get $ag(v) \leq ag^c(v)$. □

Indeed this lemma says more. With sufficient depth the noncontiguous label can become better than multiple edge weights. We can also derive a bound on the maximum difference between the contiguous and noncontiguous version in terms of the maximum edge-weight and depth of the tree starting from the final root chosen for clearing the tree. Hence the difference can become significant for deeper trees. This shows that the trade-off for a contiguous strategy is that one has to block more edges throughout the tree.

Interestingly a combination of the contiguous and noncontiguous version can still be better than either one. This is easiest to see by an example. Consider v_x and edge $e = [v_x, v_y]$ and v_y also connected to v_2, v_3, v_4, v_5, v_6. Let $\lambda_{v_y}(e_i) = a, 2 \leq i \leq 6$, where $a > 3$ is some constant and let $w(v_y) = w(e) = w(e_2) = w(e_3) = 1, w(e_4) = 2, w(e_5) = w(e_6) = 3$. Now $\lambda_{v_x}(e) = a + 7$. Consider an alternative of clearing the subtrees by clearing v_2, v_3, v_4, v_5, v_y and v_6 in this order. We need $a + 4$ for clearing the first four, then $s(v_y) = 12$ for clearing v_y, leaving $w(e)+w(e_6) = 4$ agents to block contaminated parts from cleared parts. Clearing v_6 will require at most a agents. Hence to maximum number of agents needed to clear the subtree rooted in v_y coming from v_x with this new strategy is $a+4$. Similarly, we can construct examples in which clearing v_y before clearing some of the subtrees requires more agents. This shows that our previous algorithm is indeed not s.t. the strategy S it produces satisfies $ag(G) = ag(S)$, even if we start the strategy at vertex v s.t. $ag(v)$ is minimal.

For weighted edge-search Barriere proves that the presented algorithm from [4] indeed produces an optimal contiguous strategy. It may well be that we can use a similar approach to show the analogue for our contiguous variant. But since we know that the non-contiguous version outperforms the contiguous we are rather interested in investigating the case when the strategy is not contiguous.

Let us now formalize how an improved strategy, based on the two examples shown above, may look like. For vertices v_x, v_y as before and $v_2, \dots, v_m$ partitioned into two sets of vertices V_1 and V_2 define: $h = \max_{v_i\in V_1}\{c(v_i)\}, h^c = \max_{v_i\in V_2}\{c^c(v_i)\} + w(e), h^u = \max\{h, h^c\}, \lambda^u = \max\{s(v), h^u\}$.

Note that this assumes that we do not have $w(e)$ blocked when we enter v_y from v_x, i.e. we assume that v_y is one of the vertices in V_1 w.r.t. label $\lambda_{v_z}([v_z, v_x])$ for some other neighbor of v_z. Despite these details that merely complicate notation we have one main challenge that remains to be solved. Given the above find a partitioning V_1, V_2 that minimizes h^u. We know from our previous results that within V_1 and V_2 the optimal order of vertices is ρ_i, i.e. strictly speaking $-\rho$ for V_2.

Investigating this new problem and attempting to find an algorithm that produces the optimal noncontiguous strategy on a tree remains subject for further work.

VI. Application and simulations

A partition of an environment in a realistic application is usually a graph. Since the general problem on graphs is NP-complete, we need to develop heuristics or approximations in order to obtain practical solutions. In [10] we first proposed to calculate the minimum spanning tree (MST) on a graph and then block all edges s.t. only MST edges remain. Let us denote all edges not belonging to the MST as cycle edges. Figure 4 shows an example of a simple environment with a given partitioning and the resulting noncontiguous strategy computed for the MST. As originally proposed, the cycle edges would be blocked throughout the execution of the strategy for the MST. Clearly, we can do a lot better. Once the MST is computed we only need to block cycle edges that connect a contaminated and a cleared vertex during the clearing process. We shall call this approach dynamic cycle blocking and the former constant cycle blocking. To validate the new approach we designed the following simple experiments.

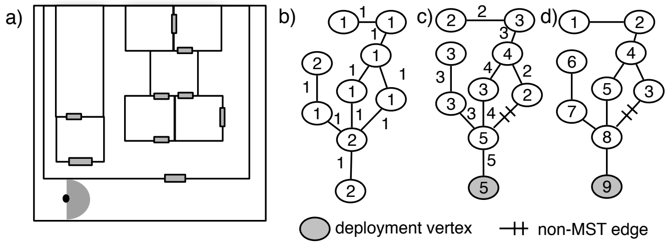

Fig. 4. Part a) shows an environment with doors as grey rectangles and walls as black lines. A sample robot is shown as a black dot with the area covered by its sensor shown as a grey half circle. Part b) shows the graph representation with weights on edges and vertices corresponding to how many robots alike the sample robot are needed. Part c) shows the MST and the noncontiguous labels w.r.t the direction from the deployment vertex. The number of agents needed to clear a vertex are written on the vertex. Part d) shows the order of clearing starting from 1.

A. Experimental set up

Random graphs with a given number of vertices and edges are created with random weights in the range 1 to 12 for

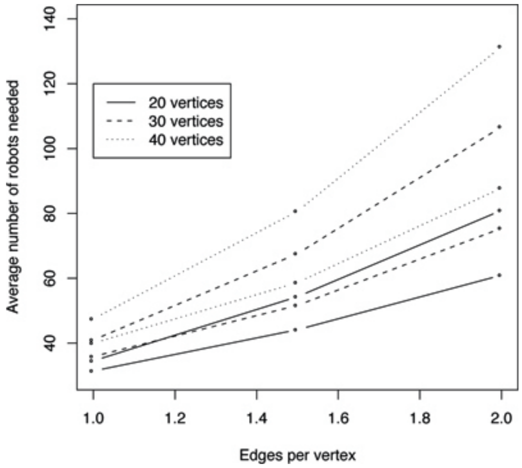

Fig. 5. Results of the experiments with 9 sets of parameters. The upper lines for each number of vertices is always the cost for the constant cycle blocking strategy and the lower for the dynamic cycle blocking strategy.

TABLE I

Reduction of the number of robots needed when using dynamic cycle blocking expressed in terms of the percentage of the number of agents needed to block all cycles at once.

Edges per vertex	1	1.5	3
20 Vertices	47.74%	41.69%	40.40%
30 Vertices	55.49%	45.09%	42.85%
40 Vertices	62.24%	47.00%	45.32%

vertices and 1 to 6 for edges. Once the graph is constructed we compute the MST and pick the root in the center of the longest path in the tree and compute all labels into the direction of the root vertex. The labels and the total weight of all cycle edges give the cost for the constant cycle blocking. To determine the costs for the dynamic cycle blocking we execute the strategy and add/remove cycle blocks as necessary. We defined 9 sets of parameters, 3 sets with 20 vertices with 20, 30 and 40 edges, 3 sets with 30 vertices with 30, 45, and 60 edges and 3 sets 40 vertices with 40, 60 and 80 edges. For the experiments we constructed 1000 graphs for each set of parameters.

B. Experimental results and discussion

Figure 5 shows the number of robots needed for clearing the graph for each set of parameters. It is apparent that the costs for constant cycle blocking are considerably higher and that the difference between constant and dynamic cycle blocking becomes greater when more edges are present. Table VI-B shows what percentage of the total costs for all cycle blocks we are saving when blocking cycles dynamically. While these experiments are very limited in scope and validity they still serve as a strong indicator that the dynamic cycle blocking can lead to a significant improvement, in particular as the number of vertices grows.

VII. Conclusion and Future Work

We presented the novel problem GRAPH-CLEAR that emerged at the cross section between multirobot surveillance applications and graph-theory. We proved that it is NP-complete and provided heuristics for finding strategies based on a reduction of the problem to trees by blocking cycles. The non-contiguous variant of the algorithm to compute strategies in trees was shown to not be optimal and a new problem for finding an optimal strategy was proposed. The presented methods were tested in simple simulations comparing the two approaches for blocking cycles, constant and dynamically, and validating the latter.

The results in this paper serve mainly as a theoretical foundation. Our ultimate goal is to deploy a large, possibly heterogeneous, team of robots with limited range sensors in a large unknown, possibly dynamic, and complex environment to carry out exploration and surveillance tasks. This line of research started in [11] and continued in [12]. While the experimental part in this paper is rather limited there is now a sound theoretical basis. The major remaining issues to be address now are the following: 1) Partitioning of the environment to enable good clearing strategies. 2) Develop an online variant. 3) Develop a distributed variant.

Furthermore, also the GRAPH-CLEAR problem has a few more unanswered questions. One might be able to proof that the contiguous version is optimal, i.e. it produces a contiguous strategy that uses at most $S(T)$ agents. But for the most part we are interested in the non-contiguous variant since it needs less agents. It remains to find an optimal algorithm for the non-contiguous variant in trees.

References

[1] L. E. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous robots*, vol. 12, pp. 231–255, 2002.

[2] T. Parsons, "Pursuit-evasion in a graph," *In Y. Alavi and D. R. Lick, editors, Theory and Application of Graphs*, pp. 426–441, 1976.

[3] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, "The complexity of searching a graph," *J. ACM*, vol. 35, no. 1, pp. 18–44, 1988.

[4] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, "Capture of an intruder by mobile agents," in *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 2002, pp. 200–209.

[5] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM Journal on Computing*, vol. 21, no. 5, pp. 863–888, 1992.

[6] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry and Applications*, vol. 9, no. 4/5, 1999.

[7] S. Sachs, S. Rajko, and S. M. LaValle, "Visibility-based pursuit-evasion in an unknown planar environment," *International Journal of Robotics Research*, vol. 23, no. 1, pp. 3–26, Jan. 2004.

[8] B. P. Gerkey, S. Thrun, and G. Gordon, "Visibility-based pursuit-evasion with limited field of view," *International Journal of Robotics Research*, vol. 25, no. 4, pp. 299–315, 2006.

[9] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of np-completeness." *W.H. Freeman and Company*, 1979.

[10] A. Kolling and S. Carpin, "Detecting intruders in complex environments with limited range mobile sensors," in *Robot Motion and Control 2007, LNCIS 360*, K. Kozlowski, Ed. Springer-Verlag London Limited 2007, 2007, pp. 417–426.

[11] ——, "Multirobot cooperation for surveillance of multiple moving targets - a new behavioral approach," *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pp. 1311– 1316, 2006.

[12] ——, "Cooperative observation of multiple moving targets: an algorithm and its formalization," *International Journal of Robotics Research (accepted for publication)*, 2007.