Probabilistic Graph-Clear

Andreas Kolling and Stefano Carpin

Abstract— **This paper introduces a probabilistic model for multirobot surveillance applications with limited range and possibly faulty sensors. Sensors are described with a footprint and a false negative probability, i.e. the probability of failing to report a target within their sensing range. The model implements a probabilistic extension to our formerly developed deterministic approach for modeling surveillance tasks in large environments with large robot teams known as Graph-Clear. This extension leads to a new algorithm that allows to answer new design and performance questions, namely 1) how many robots are needed to obtain a certain confidence that the environment is free from intruders, and 2) given a certain number of robots, how should they coordinate their actions to minimize their failure rate.**

I. Introduction

In the recent past we have studied the problem where multiple robots with limited sensing capabilities are used to detect all intruders possibly hidden in a complex environment. This preliminary study led to the formal definition of a new graph theoretic problem dubbed Graph-Clear [4], [5], [6]. Informally speaking, Graph-Clear asks to determine the minimum number of robots needed to spot all intruders, and how these robots should coordinate their actions to achieve this goal. Besides the modeling effort, we have also investigated the computational properties of this problem, showing that in its most general formulation it is NP-hard. For the special case of trees we have also determined efficient polynomial-time algorithms that determine so-called strategies, i.e. a sequence of coordinated actions that will eventually detect all intruders. One of the appealing aspects of Graph-Clear is that, despite its high-level theoretical formulation, and differently from many similar studies that remained purely theoretical, it is possible to easily implement it in contemporary robots. On the other hand, one of the limits of our previous study has been the *deterministic* assumption. To be precise, in the past we have assumed an error-free sensing process. In fact, it was hypothesized that whenever an intruder was within the sensing range of a robot its presence was always reported. In practice, however, no sensor is error-free. In this paper we extend our previous deterministic formulation to include possibly faulty sensors. In particular, we here account for sensors that may give false negatives, i.e. with a certain known probability they may report that no intruder is within their sensing range even if there is one or more. The use of robots equipped with these faulty sensors naturally leads to a number of design and performance questions, like for example the following:

School of Engineering, University of California, Merced, CA, USA

- if after having cleared an environment the robot team reports that no intruders were found, what is the probability that instead n intruders (with n being a positive natural number) successfully managed to remain undetected due to errors in the sensing process?
- given r robots, what is the clearing strategy that minimizes the probability that one or more intruders remain undetected?
- given a certain environment and a target probability p, how many robots are needed in order to be sure that if the team reports no detection, then with probability at least p there are indeed no intruders in the environment?

The main contribution of this paper is twofold. First, the deterministic model is extended into a probabilistic one in order to account for faulty sensors. This modification entails a number of updates in the model components that are fully worked out in this manuscript. Moreover, an appropriate model for faulty sensors is presented. Secondly, in the light of the new model, one of the algorithms we formerly developed for the deterministic case is extended in order to answer the above questions.

The paper is organized as follows. Section II shortly revises related research in the field of multi-robot surveillance. Next, Section III presents the deterministic Graph-Clear problem in informal terms, outlining the elements that need a new definition when transitioning to the probabilistic case. The probabilistic model is then introduced in section IV, and a model for faulty sensors is discussed in V. Finally, the algorithmic core is illustrated in section VI, and conclusions are provided in VII.

II. Related Work

Probabilistic aspects of multi-robot surveillance have been investigated in a variety of forms. Most often the problem of detecting mobile intruders in an environment is known as pursuit-evasion. In [1] Adler et al. defined a *Hunter vs. Rabbit* game on a graph and devised probabilistic strategies to solve it. Only the movement of the agents, hunter and rabbit, is probabilistic and not the actual detection of the rabbit, which always occurs exactly when rabbit and hunter occupy the same vertex. In [2] Isler et al. also consider randomized strategies, albeit in a polygon instead of a graph. In [3] Isler et al. investigate the hunter and rabbit game on a graph with local visibility, i.e. the rabbit can see the hunter when it is on an adjacent vertex. In [14] a probabilistic approach is used in the design of an actual robot team for pursuit-evasion. The approach casts map building and pursuit-evasion into one probabilistic framework and emphasis is put on the design of a distributed architecture. Targets are assumed to move on a grid according to a Markov model. The environment in which the system is tested is fairly open and several heuristic policies for moving the robots are tested in real experiments. A similar probabilistic approach is developed in [10]. Therein the target moves according to a worst-case Markov process. Motion coordination is achieved with an A*-like search on a graph representation of the environment. Robots move on the graph following a heuristic function which encourages movement towards closer nodes with high probabilities for targets being located therein. Deterministic pursuit-evasion problems on a graph were introduced by Parsons in [11]. The problem, also known as graph searching, involves capturing an arbitrarily fast intruder on a graph with a team of agents that can either block the intruder's motion through a vertex, or catch it when moving along an edge. The possibility of an intruder being located somewhere is considered as *contamination* and the agents actions remove contamination until the entire graph is cleared. In [7] LaPaugh showed that optimal strategies for this problem do not require recontamination. In combination with the NP-hardness result from Megiddo et al. [9] this result leads to a proof of NP-completeness for finding the minimum number of agents needed to clear the graph of contamination. The graph-search problem has been studied in many other variations and it also has a relationship to visibility-based pursuit-evasion introduced by Suzuki and Yamashita [13]. Visibility-based pursuit-evasion is concerned with detecting an intruder in a planar environment with an unlimited range sensor. This problem has also been investigated in a number of variations, such as the k-searcher which emits k sensing beams to detect intruders with $k \in \{1 \ldots, \infty\}$ [13], [8]. Sachs et al. present in [12] an online algorithm to solve the visibility-based pursuit-evasion problem for a point pursuer moving in an unknown, simply-connected, piecewise-smooth planar environment. The pursuer is assumed to be equipped only with a sensor that measures depth-discontinuities, and can perform only wall-following or a movements along the perceived depth-discontinuities.

III. Deterministic Graph-Clear: model and algorithm

We formalized the deterministic Graph-Clear problem in [4]. It is based on the concept of a surveillance graph defined as an undirected graph $G = (V, E)$ with vertex set V (with n elements), edge set E (with m elements), and a *weight* function $w : V \cup E \rightarrow \mathbb{N}^+$ ($\mathbb{N}^+$ denotes the set of positive natural numbers). Weights on edges and vertices describe the costs in terms of the number of robots needed to execute certain actions known as *blocking on edges* and *sweeping on vertices*. These abstract actions represent routines that after their execution guarantee that 1) in the case of blocks no intruder crosses the edge undetected or 2) in the case of sweeps no intruder present in a vertex remains undetected. To represent the possibility of intruders being located in G, the concept of *contamination* is used. Initially all of G is contaminated and sweeping and blocking removes contamination. Also, contamination spreads from each vertex or edge to any other vertex or edge to which a path without blocked edges exists. This spreading of contamination is instantaneous and represents a target moving with unbounded speed and complete knowledge of the environment and the actions of the robot team. The abstraction of a graph is used to model complex environments, with vertices representing places, and edges representing connections between places. We recently presented an algorithm that extracts both the graph G and the w function from a grid map representation of the environment and a simple sensor model [5].

The Graph-Clear problem asks to find a sequence of actions, a so-called *strategy*, that removes all contamination from G and requires the least number of robots. To formalize this optimization problem the concepts of action set and cost of actions are introduced. Note that formally an action allows also multiple simultaneous blocks or sweeps.

Definition 1 (Action set and actions): The *action set* of a surveillance graph G is the subset of $\{0, 1\}^{n+m}$ where each element $a = \{a_1, \ldots, a_{n+m}\}$ (called *action*) satisfies the following constraint:

- if $a_i = 1$ with $1 \leq i \leq n$, then $a_{n+j} = 1$ for each edge $e_j \in edges(v_i)$

If $a_i = 1$ with $1 \leq i \leq n$, we say that the action a sweeps vertex v_i, and if $a_{n+j} = 1$ with $1 \leq j \leq m$ we say that action a blocks edge e_j. The action set of G is indicated as $\mathcal{A}(G)$.

The reader should observe that the definition of *action* mandates that while a vertex v is being swept all edges ending on v (indicated as $edges(v)$) must be blocked. This constraint is introduced to model situations where large areas are represented by a single vertex. In such scenario if all edges are not kept blocked during the sweeping, a malicious intruder may recontaminate an area already cleared.

Definition 2 (Sweeping and blocking cots): Let G be a surveillance graph. The *sweeping cost* of a vertex $v \in V$ is $w(v)$, while the *blocking cost* of an edge $e \in E$ is $w(e)$.

Definition 3 (Cost of an action): Let G be a surveillance graph and let $a \in \mathcal{A}(G)$ be an action. The *cost of action a* is:

$$c(a) = \sum_{i=1}^{n} a_i w(v_i) + \sum_{j=1}^{m} a_{n+j} w(e_j)$$

We can now denote the cost of clearing a vertex v while avoiding recontamination by $s(v) := w(v) + \sum_{e \in Edges(v)} w(e)$, i.e. sweeping the vertex while keeping all its edges blocked. This cost definition will be used to extend the deterministic model to the probabilistic variant, in which the cost function is more flexible in as much as it will provide a relationship between the cost and the probability of missing a target. Finally, the concept of *strategy* is needed. Informally speaking, a strategy is a sequence of block and sweep actions that remove all contamination from G[1].

We will use the algorithm from [4] which computes contiguous strategies on trees as a basis for our probabilistic

[1]The formal definition of strategy is hereby omitted for lack of space. Its formalization is not challenging, but requires the formal definition of a variety of terms not relevant for the scope of this paper

algorithm. A contiguous strategy has the additional constraint that, at every step during the execution of the strategy, cleared vertices need to form a connected subgraph. The algorithm to compute such strategies is based on an auxiliary label defined on the edges. Given an edge $e = [v_x, v_y]$ between vertices v_x and v_y, the label $\lambda_{v_x}(e)$ denotes the number of robots needed to clear all vertices one can encounter when crossing edge e while coming from v_x. If v_y is a leaf there are no other neighbors and then

$$\lambda_{v_x}(e) = w(v_y) + w(e). \tag{1}$$

Otherwise consider all neighbors of v_y different from v_x written as $v_1, \dots, v_m$ where $m = degree(v_y) - 1$ and let them be ordered by $\rho_i = \lambda_{v_y}(e_i) - w(e_i)$ s.t. $\rho_i \geq \rho_{i+1}$. Figure 1 shows a visualization of this concept. We now clear v_y and then subsequently all the subtrees roots at each v_i in the reverse order defined by ρ, i.e. the largest index i first. The cost in terms of robots needed for each v_i is then:

$$c(v_i) = \lambda_{v_y}(e_i) + \sum_{l=1}^{i-1} w(e_l). \tag{2}$$

Hence the label becomes the maximum cost encountered in this process:

$$\lambda_{v_x}(e) = \max\{s(v_y), \max_{i=1,\dots,m}\{c(v_i)\}\}. \tag{3}$$

Figure 1 shows a contiguous strategy.

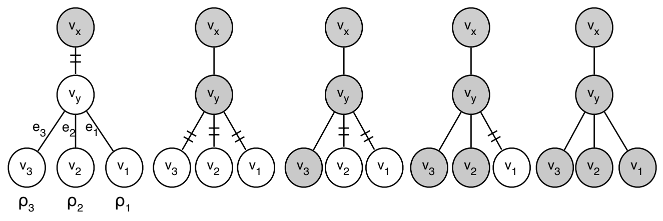

Fig. 1. An illustration of the computation of labels for the algorithm computing contiguous strategies (gray vertices are clear, and edges with double strokes are blocked). On the left we have v_x cleared and v_y and subtrees rooted at v_1, v_2, v_3 contaminated. In each step moving towards the right the next vertex in the ordering is cleared and blocks are released.

The label $\lambda_{v_x}(e)$ captures the maximum number of robots used. From equation 2 it already becomes apparent that these labels are computed recursively. By first computing the labels on all edges towards the leaves, using equation 1 we can bootstrap the computation of all other labels. Once the labels towards leaves are computed there is at least one internal vertex that has all neighbors except one as leaves. This vertex becomes v_y and the non leaf neighbor v_x and all leaf neighbors $v_1, \dots, v_m$. Since $v_1, \dots, v_m$ are all leaves we have $\lambda_{v_y}(e_i)$ already computed and can hence compute $\lambda_{v_x}(e)$ according to equation 3. Once this is computed there will be another vertex v_y that has at least $m = degree(v_y) - 1$ outgoing labels, i.e. $\lambda_{v_y}(e_i)$, computed and we can compute its incoming labels $\lambda_{v_x}(e)$ where $e = [v_x, v_y]$ and so on until every edge has two labels, one for each direction. Once all the labels are computed we can compute the total cost $ag(S_v)$ for clearing the tree with a strategy S_v starting for a vertex v by considering all its neighbors $v_1, \dots, v_m$ where now $m = degree(v)$ and equation 3 in a slightly modified form:

$$ag(S_v) = \max\left\{s(v), \max_{i=1,\dots,m}\{c(v_i)\}\right\}. \tag{4}$$

The precise algorithm and details about contiguous strategies are found in [4]. Similarly, when a block is first installed on an edge, it will be maintained for a certain time, but after it has been released, it will never be reinstantiated again. For these reasons, in the following we can talk about the event *clearing vertex* v without ambiguity, because this event occurs only once. A similar consideration holds for edge blocking.

IV. Probabilistic Model

In order to extend the Graph-Clear formalism from a deterministic to a probabilistic scenario, various concepts need to be accordingly updated or introduced. Before getting into the details we clarify one important point concerning the remaining of the discussion. From now on we concentrate on the case of robots not reporting intruders. We hereby assume that when a robot reports an intruder this event is separately handled, e.g. a human operator is dispatched, a tracking behavior is triggered or the like. Also, while we assumed the possibility of false negatives, we do not assume false positives, i.e. an intruder is detected only when it is really present. For this reason the event *intruder detected* never occurs in the discussion. Our interest in this paper is in drawing conclusions upon a sequence of negative observations reported by the robots.

1) Worst case adversary: As previously mentioned, the deterministic Graph-Clear framework has been formulated under a worst case scenario. More precisely, in the deterministic scenario this hypothesis implies targets moving with unbounded speed and with complete knowledge of the environment and of the actions of the robots. Hence, whenever a strategy leaves room for recontamination, it will certainly happen. To maintain this idea, the worst case adversary has to be differently defined when faulty sensors are used. In the probabilistic scenario the worst case adversary still has complete knowledge of the robots' positions, as well as of their sensors error rates. Each of the intruders will try to maintain their *undiscovered* status by crossing blocks or sweeps where the highest error rate occurs. As anticipated, these crossing events must occur, since ultimately all elements in the graph will be swept or blocked. This concept will be clearer after the probabilistic sensor model will be formally specified.

2) Environment: A probabilistic surveillance graph is similar to a deterministic surveillance graph, with the important difference that instead of $w : V \cup E \to \mathbb{N}^+$ we introduce $w : V \cup E \to \mathcal{F}$, where the set $\mathcal{F}$ is defined as follows:

$$\mathcal{F} := \{\ f \mid f : \mathbb{N} \to [0,1], f(0) = 1, \forall r, r' \in \mathbb{N}\ r \geq r' \Rightarrow f(r) \leq f(r')\}.$$

That is, in the probabilistic case each graph element is not associated with a constant cost, but rather with a monotonically decreasing function mapping the natural numbers to the interval $[0, 1]$. Throughout the paper, with a slight abuse of notation, we will write $w_x(r)$ for $w(x)(r)$ for some $x \in V \cup E$ and $r \in \mathbb{N}$. Also, in order to ease the discussion, whenever we write x we mean either a vertex or an edge. What was previously understood as the weight, namely the number of robots needed for a block or sweep, now becomes a function defining the probability of a false negative (i.e. no intruder reported even if there was at least one passing through the sensor footprint during the block or sweep) while executing a block or a sweep using a certain number of robots. According to the intuition, for every x, $w_x(0) = 1$, i.e. if no robot is used then the probability of not reporting any intruder is always 1. Using more robots hence leads to an improvement in the detection capabilities. Moreover, we assume that each vertex and edge are associated with a grid representation that roughly represents their planar layout.

3) Probabilistic Actions and Probabilistic Strategies: As a consequence of the new definition of w, the notion of a strategy also needs to be extended. While it was formerly defined as a sequence of actions which essentially determines which block and sweep operations are executed at which time step, a probabilistic strategy now describes the number of robots allocated to a each sweep or block of an action.

Definition 4 (Probabilistic action): The *probabilistic action set* of a probabilistic surveillance graph G is $\mathbb{N}^{n+m}$ where each element $a = \{a_1, \dots, a_{n+m}\}$ (called *probabilistic action*) has an associated cost $c(a) = \sum_{i=1}^{n+m} a_i$.

The reader should observe the fundamental difference with the deterministic case, where actions are elements of $\{0, 1\}^{n+m}$. In the deterministic scenario a block or a sweep is either executed or not. In the probabilistic case these operations are instead executed using a certain number of robots. Also, due to the way the set $\mathcal{F}$ was defined, we relax the explicit requirement that edge blocks are executed concurrently to vertex sweeps. With the new definition of probabilistic action, if these blocks are not executed than the corresponding function yields a probability of non detection equal to 1, rendering such a strategy useless. The functions w_x are the essential probabilistic element in the graph G.

4) Undetected intruders and false negatives: We describe the number of undetected intruders in the environment with the discrete random variable T, and we write $p(T = i), i \in \mathbb{N}$, for its mass distribution. For each edge or vertex x we will write $p_x(N|t, r)$ for $w_x(r)$, i.e. the probability of a false negative. A negative observation is written as N and the event of a target crossing is written as t. The number of robots that are executing the corresponding sweep or block operation on x is given by r. For notational convenience we will drop the r and write $p_x(N|t)$ assuming that there is a set number of robots. Keeping in mind that each x is blocked or swept once during the execution of a fixed strategy S, let $\bar{N}$ denote a sequence consisting of only negative observations during the execution of S. We are interested in studying the following probability, that is here written using Bayes rule:

$$p(T = i|\bar{N}) = \frac{p(\bar{N}|T = i) \cdot p(T = i)}{p(\bar{N})} \tag{5}$$

where for a fixed strategy $p(\bar{N})$ is a normalization constant. Now, $p(T = i)$ is simply the prior target distribution of the number of targets, and the most important part is $p(\bar{N}|T = i)$ which we intend to relate to w_x. It is in this term that the smart target assumptions has significant consequences. We restrict the collection of observations to robots engaged in a sweep or a block. Recall that an undetected target after the entire strategy is executed must have crossed either through a block or been in a vertex during a sweep. The undetected target then crosses from the contaminated part to the cleared part at some x. An individual smart target will choose its crossing point x so that $p_x(N|t)$ is largest. A smart group of i targets acting in a cooperative way will each choose an x s.t. $p(\bar{N}|T = i)$ is maximized. This is an important distinction since the probability for detections may be different for sensors that are more sensitive when multiple targets cross at once. In the general case, for each x we should then specify $p_x(N|T_c = i)$, i.e. the probability of a negative observation given that i targets cross x during its block or sweep. However, it is not practical to specify $p_x(N|T_c = i)$ for all i and doing so also leads to more complications. It may be the case that i targets in the environment achieve the lowest likelihood of detection if split into $i_1 + i_2 = i$ with some crossing on $x_1 \in G$ and some on $x_2 \neq x_1 \in G$ if $p_x(N|T_c = i) < p_{x_1}(N|T_c = i_1) \cdot p_{x_2}(N|T_c = i_2), \forall x \in G$. For simplicity we prefer to describe the sensing in terms of $p_x(N|t) = p_x(N|T_c = 1)$. Therefore we assume that all targets crossing an element x are detected independently. This assumption leads to $p_x(N|T_c = i) = p_x(N|t)^i$. In this case a group of smart targets would all choose to cross at the same x with $p_x(N|t)$ largest, but not necessarily at the same time. Alternatively, we could have chosen e.g. $p_x(N|T_c = i) = p_x(N|t), \forall i$, then a group of smart targets will all choose to cross at x with $p_x(N|t) = p_x(N|T_c = 1)$ largest and they will choose to cross at once. This choice has different implications and the choice should ideally coincide with the actual sensor properties. To summarize, all this taken together we now get that $p(\bar{N}|T = i) = p_{x_{max}}(N|t)^i$ where $x_{max} = argmax_{x \in G}\{p_x(N|t)\}$ from the assumption that targets are detected independently, choose the worst case path and have complete knowledge about actions of the robot team. Before we proceed with an algorithm that computes probabilistic strategies let us shortly discuss how to obtain p_x, or equivalently w_x, from basic sensing on the grid.

V. Modeling faulty sensors

The actual sensors of the robot team are described by their footprint, i.e. their coverage of part of the grid g representing the environment, and their probability for misses which may differ for each cell the sensor covers. They produce observations in discrete time intervals and may at any given interval either return a positive observation, i.e. a flagged target detection or a negative observation, i.e. no target detection. Fig. 2 shows an example of a sensor placed in a grid and shows its coverage. In general, the sensor may have any number of cells covered and any probability associated with any cell, so there is no restriction on the type of sensor except that its covered area is discretized on a grid.

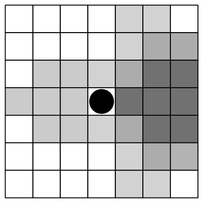

Fig. 2. A grid with a sensor placed in its center as a black circle and with the cells observed by the sensor in grey. A darker grey tone denotes a smaller false negative probability.

From the sensing probabilities on individual cells, we need to obtain probabilities for the blocks and sweeps. The details on how to execute these actions, however, are only given when they are actually implemented in a specific application. In the deterministic model the requirement for the implementations on a block are that it has to guarantee that an intruder attempting to cross the block will be detected. In the probabilistic scenario we can modify this into having the intruder cross through at least one grid cell covered by a sensor so that the probability of detecting it is non-zero. The main difference, however, is that we can increase the probability of detecting the intruder if we utilize more robots than the minimum necessary, hence the monotonically decreasing trend for the function w_x for $x \in G$. Given that we assumed worst case targets, we assume that a target chooses the path with the smallest probability of being detected. The probability for a miss on the edge will then become the probability of a miss of a target on this path. Fig. 3 shows two robots blocking a hallway and the worst case target path as well as four robots blocking and leading to a higher probability of detecting the target.

For an edge e let $C_t = \{g_1, \ldots, g_{n_p}\}$ be a set of grid cells that are covered by the sensors of the block on e and that are traversed by the target on its worst case path through the block. Now the probability of a miss on the block of e becomes $p_e(N|t) = \Pi_{i=1}^{n_p} p_{g_i}(N|t)$, where $p_{g_i}(N|t)$ is the probability of a miss on grid cell g_i. Hence a target can only pass undetected if it is undetected on each cell. Note that the equation assumes independence in the detections. Here $p_{g_i}(N|t)$ is given by $p_{g_i}(N|t) = \Pi_{j=1}^{n_{g_i}} p_{s_j}$ where $\{s_1, \ldots, s_{n_{g_i}}\}$ are the n_{g_i} sensors covering cell g_i. One may increase the probability of detecting the target by increasing the number of robots for the block action. This may not always be possible, depending on the constraints of the environment. We may e.g. not be able to add additional robots to a block due lack of space or we may need many more robots to yield an improvement. The latter case is seen in fig. 4. This is the motivation for having w_x as a general function that is only required to be monotonically decreasing and starting at $w_x(0) = 1$ and otherwise unrestricted, since it can then also capture the above cases.

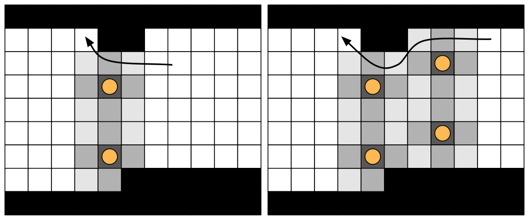

Fig. 3. An illustration of the computation of detection probabilities for blocks through the worst case path a target can take through a block.

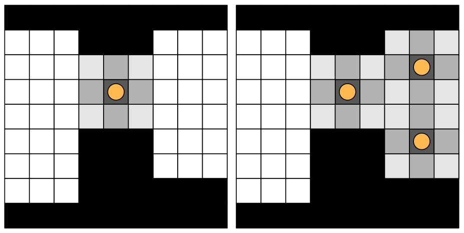

Fig. 4. The basic block in this figure only needs one robot, while the first reinforcement leading to an improvement in the detection capability needs two additional robots. In order to get this fact the reader should consider that intruders may also move diagonally on the grid.

An analogue case can be made for a sweeping routine, even though its derivation is slightly more tedious to describe and hereby omitted. Either way, we assume a sweep routine also gives a final w_x, i.e. in this case the probability that a negative observation is made when targets are present in the corresponding vertex v_i and r robots are used.

VI. Probabilistic Graph-Clear

In this section the deterministic Graph-Clear algorithm is extended to a variant that considers the probabilistic nature of the sensors. We focus on the conservative scenario with worst case adversarial targets and the model introduced in the previous section. Recall that for each edge and vertex we are given a monotonically decreasing function $w_x : \mathbb{N} \to [0, 1]$, which gives us a miss probability $p_x(N|t, r)$ when using r robots for the block or sweep on x. In the light of the model just developed, the last two questions raised in the introduction can then be reformulated as follows:

1) Given a desired $p(T = 0|\bar{N}) \in [0, 1]$ how many robots are needed?
2) Given r robots what is the strategy producing the highest $p(T = 0|\bar{N})$?

Recall the computation in the deterministic algorithm for Graph-Clear of a label $\lambda_{v_x}(e)$ on an edge $e = [v_x, v_y]$. The label $\lambda_{v_x}(e)$ is the number of robots needed to clear all vertices beyond e when coming from v_x towards v_y. At this point we are considering the neighboring vertices $v_1, \ldots, v_m$ of v_y different from v_x and compute $\lambda_{v_x}(e) = \max\{s(v_y), \max_{i=1,\ldots,m}\{c(v_i)\}\}$.

For the probabilistic variant we can employ a similar reasoning, except that we now need to get a function $\lambda^e_{v_x}(r)$ instead of just a label, analogue to having a function w_x instead of just a constant weight on a vertex or edge. $\lambda^e_{v_x}(r)$ will return the probability of failing to report one or more

Algorithm 1 $Compute\_all\_lambda\_functions(T, B)$

```
1: Set all label functions to 0 and initialize empty queue $O$
2: $O.enqueue(leaves(G))$
3: while not $O.empty()$ do
4:     $v_y \leftarrow O.dequeue()$
5:     $Compute\_lambda\_function(v_x, v_y, B)$
6:     $a \leftarrow$ number of neighbors of $v_x$ s.t. $\lambda_{v_x}^{[v_x,v]}$ is computed
7:     if $a = degree(v_x) - 1$ then
8:         $O.enqueue(v_x)$
9:     else if $a = degree(v_x)$ then
10:        for all $v \in neighbors(v_x)$ s.t. $\lambda_v^{[v,v_x]}$ not computed do
11:            $O.enqueue(v_x)$
12:        end for
13:    end if
14: end while
```

targets for all clearing steps when moving from v_x towards v_y and clearing all neighboring subtrees with r robots. Due to the assumption about target movement the overall probability for a miss will be the maximum of all probabilities of misses during any of the steps. Let us illustrate how to construct $\lambda_{v_x}^e(r)$. For each w_x we have a minimum number of robots required to get a miss probability of less than 1. Formally we have for each $x \in G$ a r_{min,w_x} with

$$r_{min,w_x} = \min\{r \in \mathbb{N} \mid w_x(r) < 1\}.$$

Using this r_{min,w_x} as the deterministic weight $w(x)$ on each $x \in G$ for the deterministic Graph-Clear algorithm we can compute $\lambda_{v_x}(e)$ as before using equation 3. Now, obviously $\lambda_{v_x}^e(r) = 1, \forall r < \lambda_{v_x}(e)$, i.e. if we use less robots than $\lambda_{v_x}(e)$ we have at least one block or sweep that does not have sufficiently many robots, and the miss probability will be 1. Let us now describe the procedure that computes $\lambda_{v_x}^e(r)$. We will consider

$$F_{v_x,e} := \{\lambda_{v_y}^{e_1}, \ldots, \lambda_{v_y}^{e_m}, w_{e_1}, \ldots, w_{e_m}, w_{v_y}\}$$

a set of functions instead of fixed weights and labels, the key difference to the deterministic case. For each of these $f \in F$ we will have an auxiliary term r_f which denotes the current argument for f, i.e. how many robots are allocated to the respective edge for blocking the subtree or for clearing it. Also, let $deterministic\_contiguous\_label(G, v_x, e)$ be a function that computes the deterministic label $\lambda_{v_x}(e)$ on G using the current r_f. Hence, it computes $\lambda_{v_x}(e)$ according to equation 3 but with $w(e_i) = r_{w_{e_i}}$ and similarly for $\lambda_{v_y}(e_i) = r_{\lambda_{v_y}^{e_i}}$. Intuitively, this computes the current cost for clearing the subtree rooted at v_y given a certain cost assignment for each f.

Algorithms 1 and 2 shows the entire procedure to compute $\lambda_{v_x}^e(r)$. Algorithm 1 can be used to compute all label functions by first considering all leaves and then moving upwards through the tree and process all non-leaves that have all neighbors but one with label functions already computed.

Algorithm 2 $Compute\_lambda\_function(v_x, v_y, B)$

```
1: if $degree(v_y) == 1$ then
2:     $\lambda_{v_x}^e \leftarrow w_{v_y}$
3: else
4:     Locate all neighbors $v_1, \ldots, v_m$ and create $F_{v_x,e}$.
5:     Initialize $r_f$ to $r_{f,min}$ for all $f \in F$
6:     $deterministic\_contiguous\_label(v_x, e)$
7:     Set $\lambda_{v_x}^e(r) = 1, \forall r < \lambda_{v_x}(e)$
8:     for $r \leftarrow \lambda_{v_x}(e)$ to $B$ do
9:         $repeatable \leftarrow true$
10:        while $repeatable$ do
11:            $f_{max} \leftarrow argmax_{f \in F_{v_x,e}} f(r_f)$
12:            $addbots \leftarrow argmin_{i \in \mathbb{N}}(f_{max}(r_{f_{max}} + i) < f_{max}(r_{f_{max}}))$
13:            $r_{f_{max}} \leftarrow r_{f_{max}} + addbots$
14:            $deterministic\_contiguous\_label(v_x, e)$
15:            if $\lambda_{v_x}(e) > r$ then
16:                $r_{f_{max}} \leftarrow r_{f_{max}} - addbots$
17:                $repeatable \leftarrow false$
18:                $\lambda_{v_x}^e(r) \leftarrow f_{max}(r_{f_{max}})$
19:            end if
20:        end while
21:    end for
22: end if
```

This part of the procedure identical to the computation of deterministic labels on a tree. Algorithm 2 shows this in detail. Its complexity is $O(e \cdot B \cdot d^2 \cdot \log(d))$. Here $e = |E|$, B is a bound given as input and d is the maximum vertex degree. The complexity results from line 14 in algorithm 2 which is itself $O(d \log(d))$ and executed within the while loop $O(d \cdot B)$ times. Finally, algorithm 2 is called $O(e)$ times in the queuing in algorithm 2.

To illustrate the algorithm let us consider two examples. First, let v_y be a leaf. In this case $F_{v_x,e} = \{w_{v_y}\}$ and $\lambda_{v_x}^e(r) = w_{v_y}(r)$. Secondly, consider the slightly more complicated example in fig. 5. All $\lambda_{v_y}^{e_i}$ are available since v_1, v_2, v_3 are just leaves. The algorithm then computes $\lambda_{v_x}^e(r)$ by first trying the smallest possible values $r_{f,min}$ for each function involved and then allocates additional robots to those functions that determine the overall maximum to reduce the probability that a target can cross undetected until the overall cost is larger than allowed. The necessity for checking whether we can improve the functions that determine the maximum is that the maximum cost during the clearing may occur when clearing v_1, v_2, v_3 or v_y, since each vertex is cleared at a different step in the strategy. Let us assume the maximum cost that determines the deterministic label occurs when clearing vertex v_2. The function that determines the maximum for the probability of letting a target through may, however, be w_{v_3}. If we use less robots than the maximum cost at v_2 while clearing v_3 then we could try to use more to decrease $w_{v_3}(r_{w_{v_3}})$ without increasing the overall number of robots needed as long as it is still below what we need during the clearing step for v_2.

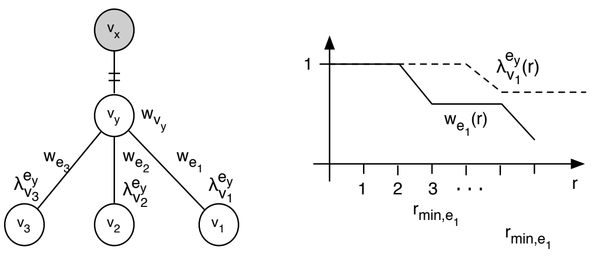

Fig. 5. An example of algorithm 2

Once all lambda functions are computed up to using B robots, both questions we formulated can be answered. For each possible starting vertex $v \in G$ we can compute a probabilistic strategy by only considering the lambda functions computed for the neighbors. To determine the number of robots needed for a certain $p(T = 0)$ we set all auxiliary terms r_f for the lambda functions, w_v and all w_x on the edges of v s.t. each function achieves a value small enough s.t. we get the desired $P(T = 0)$ from equation 5 by having the appropriate $P(\bar{N}|T = i)$. Note that we may not obtain $p(T = 0)$ exactly since the lambda functions are discrete, but we get the next possible value smaller or equal. Finally, to obtain the strategy we proceed exactly as for the deterministic variant utilizing equation 4 to obtain the final cost for a strategy starting at v. Now we do this for every starting vertex and select the best as a desired starting point and return the needed number of robots. To determine the best $p(T = 0)$ reachable with a certain number of robots we simply compute the lambda function of a virtual edge from a new vertex v to a vertex $v_i \in G$. Fig. 6 illustrates this graphically. This lambda function can now be read and the value for the available number of robots can be determined. Doing this for every $v_i \in G$ and selecting the best starting vertex yields the answer to our the last question.

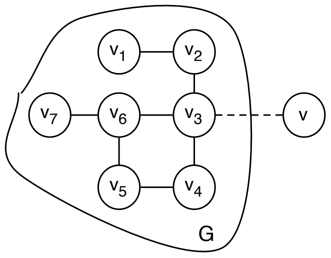

Fig. 6. Adding a virtual edge to compute the cost of clearing G starting from v_3. The lambda function on the virtual edge, represented as a dotted line will represent the probability of clearing everything beyond that edge.

VII. Discussion and Conclusion

In this paper we have presented a probabilistic extension to the Graph-Clear formalism we formerly introduced under deterministic assumptions. The probabilistic extension allows to model faulty sensors that may return false negatives, i.e. they may fail to detect an intruder crossing their sensing range. This failure rate is characterized by a probability distribution that can be easily computed once specific sensors are considered. The introduction of this function is the main change to the deterministic model and entails a different concept of a solution strategy for the problem at hand. Building upon this change in the model, a formerly developed algorithm for the deterministic case has been extended to answer performance and design questions introduced in the introduction. It is worth outlining that while extending the formalism towards the probabilistic scenario we have retained the hypothesis of worse case adversaries, i.e. intruders that have complete knowledge of the environment, of the position of the robots and of their error rates. The results presented in this manuscript have to be therefore accordingly interpreted, namely they describe the system performance when facing the smartest possible set of intruders. The benefit of the probabilistic extension is a more accurate reasoning about the inherent uncertainty in sensing and reflects the reality of robotic applications more accurately than the deterministic approach. More work in this direction is envisioned and we could also consider uncertainty in the robots motion when computing the miss rates for blocks and sweeps, e.g. robots may with a small probability not execute a block accurately due to faulty information about their location and give a target the opportunity to pass through undetected.

References

[1] M. Adler, H. Racke, N. Sivadasan, C. Sohler, and B. Vocking. Randomized pursuit-evasion in graphs. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, 2002.

[2] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion in a polygonal environment. In *IEEE Transactions on Robotics*, volume 21, pages 875 – 884, 2005.

[3] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with local visibility. *SIAM J. Discret. Math.*, 20(1):26–41, 2006.

[4] A. Kolling and S. Carpin. The graph-clear problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Proc. of IEEE/RSJ Intl. Conf. On Int. Robots and Systems*, pages 1003–1008, 2007.

[5] A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *Proc. of IEEE/RSJ Intl. Conf. On Int. Robots and Systems*, pages 2323–2328, 2008.

[6] A. Kolling and S. Carpin. Multi-robot surveillance: an improved algorithm for the graph-clear problem. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, pages 2360–2365, 2008.

[7] A. S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.

[8] S. LaValle, D. Lin, L. Guibas, J. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 737–742, 1997.

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

[10] M. Moors, T. Röhling, and D. Schulz. A probabilistic approach to coordinated multi-robot indoor surveillance. In *Proc IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 3447–3452, 2005.

[11] T.D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Appl. of Graphs*, volume 642, pages 426–441. Springer Berlin / Heidelberg, 1976.

[12] S. Sachs, S. Rajko, and S. M. LaValle. Visibility-based pursuit-evasion in an unknown planar environment. *Int. J. Robotics Research*, 23(1):3–26, January 2004.

[13] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. on Computing*, 21(5):863–888, 1992.

[14] R. Vidal, O. Shakernia, H. Kin, D. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662–669, 2002.