Detecting intruders in complex environments with limited range mobile sensors.

Andreas Kolling and Stefano Carpin

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

Summary. This paper examines the problem of detecting intruders in large multiply-connected environments with multiple robots and limited range sensors. We divide the problem into two sub-problems: 1) partitioning of the environment and 2) a new problem called *weighted graph clearing*. We reduce the second to a *weighted tree clearing* and present a solution for finding paths for the robot team that ensure the detection of all intruders with a proven upper bound on the number of robots needed. This is followed by a practical performance analysis.

1 Introduction

Multi-target surveillance by a team of mobile robots as one of the more active and novel research areas in robotics has received particular attention in recent years. Its core problems are at an interesting cross-section between path-planning, map building, machine learning, cooperative, behavioral and distributed robotics, sensor placement and image analysis. We are currently investigating the problem of detecting intruders in large and complicated environments with multiple robots equipped with limited range sensors. The problem is closely related to *pursuit-evasion* and *cooperative multi-robot observation of multiple moving targets*, short CMOMMT, which we are going to discuss shortly in section 2. The main contribution of this paper is to present a starting point for an approach for a team of robots with limited range sensors. We divided the initial problem into two sub-problems: 1) to partition the environment into a graph representation and 2) to clear the graph from all possible intruders with as few robots as possible.

For the remainder of this paper we are going to focus on the later problem. Interestingly, the division has the effect that we can present an algorithm to compute a clearing strategy on the graph without considering the robots' capabilities directly. Hence with some further work the algorithm is extendable to teams of heterogeneous robots. The clearing strategy consists of paths for the robots on the vertices and edges of the graph. When executed it ensures

that any arbitrarily fast moving target present in the environment will be detected. To find a clearing strategy we need to solve a graph theoretical problem which we dub *weighted graph clearing*, introduced in section 3.1. In section 3 we provide a solution for the problem with a proven upper bound followed by some experiments in section 4. Finally, our results and and a outlook on the first sub-problem are to discussed in the section 5.

2 Related Work

The two research areas most closely related to our work are CMOMMT and pursuit-evasion. CMOMMT is related due to its focus on multi-robot teams in large environments with sophisticated cooperation. The term was coined by Parker who produced the first formal definition of the problem and an algorithm in [2]. It focuses on multi-target surveillance in large environments with mobile robots with limited range sensors with the goal to optimized resource usage when resources are not sufficient to surveil all targets. Recent improvements were presented in [1] by introducing a new distributed behavioral approach, called Behavioral-CMOMMT. The improvements are due to an advanced cooperation schemes between robots which boost the overall performance.

Pursuit-evasion is usually tackled from the perspective of the pursuer, i.e. the goal is to try to make impossible for the evader to escape. Most current solutions for this pursuit-evasion problem, however, require unlimited range sensors. In [3] an algorithm for detecting all targets in a subclass of all simply connected environments is presented. The restrictions on the subclasses are relatively few and the performance of the algorithm is proven. It guarantees to find a path to clear the environment of all targets using only a single pursuer with a gap sensor with full field of view and unlimited range. The drawbacks are frequently revisited areas and hence long clearing paths. In [4] and [5] a previous algorithm for a single pursuer is presented in more detail and in [4] interesting bounds on the number of robots required for certain free spaces are established. For simply-connected free spaces a logarithmic bound is obtained, while for a multiply-connected environment a bound in the square root of the number of holes is given. Furthermore, finding the number of robots needed to clear an environment was shown to be NP-hard. Gerkey et al. in [6] extended the previous algorithm to varying field of view.

Pursuit-evasion solutions draws most of their information from a particular method of partitioning to finding a clearing strategy. This will be useful for extensions of this paper when the first sub-problem is discussed and related to the weighted-graph clearing problem. For now we shall present a solution that is independent of any partitioning but yet a powerful method in the *divide and conquer* spirit.

3 The Algorithm

To prepare the grounds for the algorithm we will define our notion of environment, regions, adjacency and contamination. An *environment*, denoted by S, is a bounded multiply connected closed subset of $\mathbb{R}^2$. A *region* is a simply connected subset of S. Let r_1 and r_2 be two regions and let $\bar{r}_1$ and $\bar{r}_2$ denote their closures. If $\bar{r}_1 \cap \bar{r}_2 \neq \emptyset$, then we say that r_1 is *adjacent* to r_2 and vice versa. Let $S^g = \{r_1, \ldots, r_l\}$ be a partition of S with l regions. Let A_{r_i} be the set of all adjacent regions to r_i. Define a binary function for each region r_i on its adjacent regions and across time: $b_{r_i} : A_{r_i} \times \mathbb{R} \to \{0, 1\}$. Let $r_j \in A_{r_i}$. If $b_{r_i}(r_j, t) = 1$ we shall say that the adjacent region r_j is *blocked* w.r.t. r_i at time t. Define another binary function: $c : S^g \times \mathbb{R} \to \{0, 1\}$. If $c(r, t) = 1$, then we shall call region r *contaminated* at time t, otherwise *clear*. Let $A^c_{r_i}$ be the set of all contaminated adjacent regions to r_i. Let c have the following property:

$$\forall \epsilon > 0 : c(r, t_c + \epsilon) = 0 \iff c(r, t_c) = 0 \wedge b \mid_{A^c_{r_i}} (t + \epsilon) = 1 \tag{1}$$

Now, if $c(r, t_c) = 0$ and $\exists \epsilon > 0$ s.t. $\forall t \in (t_c, t_c + \epsilon], c(r, t) = 1$, then we say that that r is *re-contaminated* after time t_c.

These definitions give us a very practical notation for making statements about the state of regions. In particular, it follows that a region becomes re-contaminated when it has a contaminated adjacent region that is not blocked. We are now going to define two procedures, called behaviors, which set the function c and b, i.e. enable us to clear and block.

Definition 1 (Sweep and Block). *A behavior is a set of instructions assigned to one or more robots. The two behaviors are defined as follows:*

1. *Sweep: a routine that detects all invaders in a region assuming no invaders enter or leave the region. After the execution of a sweep on a region r at time t we set $c(r, t) = 0$, i.e. not contaminated*[1]*.*
2. *Block: a routine that ensures that no invaders enter from or exit to adjacent regions. For a region r a block is executed on an adjacent region r_a in A_r. A block on r_a is defined as covering the boundary of r intersected with r_a with sensors, i.e. $\delta\bar{r} \cap \bar{r}_a \subset sensor\_coverage$, where $sensor\_coverage$ is the area covered by all available sensors of all robots.*

Furthermore, let us denote a sweep as safe if and only if during the execution of the sweep we have $b \mid_{A^c_{r_i}} = 1$. It follows that after a safe sweep a region remains clear until a block to a contaminated region is released.

All the previous can be understood as the glue that connects the two subproblems, partitioning and finding the clearing strategy. Partitioning returns the graph representation of the environment into which one can encode the

[1] therefore we assume that when a robot detects and intruder, the intruder is neutralized and cannot operate anymore

number of robots needed for the behaviors on the edges, the blocks, and the vertices, the sweeps, by adding a weight to them. The values of the weigths are specific to the robots capabilities and methods used for sweeping and blocking. Depending on the partitioning, very simple methods can be used. We assume a homogeneous robot team from here on, which makes the interpretation of these weights easier and the notation for the following simpler to follow. We now define the main problem that we are going to investigate, i.e. *weighted graph clearing*. The task is to find a sequence of blocks and sweeps on the edges and vertices of an entirely contaminated graph such that the graph becomes cleared, i.e not contaminated, with the least number of robots. This sequence of blocks and sweeps will be our clearing strategy to detect all intruders.

3.1 Weighted graph clearing

We are given a graph with weighted edges and vertices, denoting costs for blocks and sweeps. The goal is to find a sequence of behaviors to clear the graph using the least number of robots. As the first step we attempt to reduce the complexity of the problem by reducing the graph to a tree by removing occurring cycles. Cycles can be eliminated by executing a permanent block on one edge for each cycle. The choice of such edge will influence the tree structure and hence our final strategy. This is a topic that requires further investigation, as discussed in section 5. Recall the partition S^g in form of a planar graph. Let the weights on the edges and vertices be denoted by $w(e_i)$ on edges and $w(r_i)$ on vertices and all be non-zero. The weight of a vertex is the number of robots needed to complete a sweep in the associated region, while the weight of an edge is the number of robots required to block it. To transform S^g into a tree we compute the minimum-spanning-tree (MST) with the inverse of the w-weight of the edges. Let B be the set of all removed edges to get the MST. The costs for blocking all edges in B and transforming S^g into a tree is $b(B) = \sum_{e_i \in B} w(e_i)$ and as such the minimum possible. From now on this cost is constant and will henceforth be ignored.

3.2 Weigthed tree clearing

Recall that a safe sweep requires all adjacent regions to be blocked. It is clear that we have to block contaminated and cleared regions alike, since the latter may get re-contaminated. Hence for a safe sweep for a region r_i we need $s(r_i) = sum_{e_j \in Edges(r_i)} w(e_j) + w(r_i)$ robots. After a safe sweep all blocks on edges to cleared regions can be released. To remove the blocks to contaminated regions we have to clear these first. This suggests a recursive clearing of the regions. To capture this we define an additional weight on each edge, $p(e_j)$, referred to as p-weight. For clarity: the p-weight for edges is the number of robots needed to clear the region accessible via this edge and all its sub-regions into the respective direction. Once we have the p-weight we

still have to choose at each vertex which edges to visit first. Let us define the order by $p(e_j) - w(e_j)$ with the highest value first.

The following algorithm computes the p-weight and implicitly selects a vertex as the root which is the starting point for the clearing. For each region r, let $Edges = \{e_{n_1}, \ldots, e_{n_{m(r)}}\}$ denote the set of edges.

1. For all edges e_j that are edges to a leaf r_i set $p(e_j) = w(e_j) + w(r_i)$.
2. For all vertices r_i that have its p-weight assigned for all but one edge, denoted by e_{n_1}, do the following:
   a) Sort all $m-1$ assigned edges w.r.t to $p(e_j) - w(e_j)$ with the highest value first. Let the order be $e_{n_2}, \ldots, e_{n_m}$.
   b) Determine $h := \max_{2 \leq k \leq m}(p(e_{n_i}) + \sum_{1<l<k} w(e_{n_l}))$
   c) For the unassigned edge e_{n_1} of r_i set $p(e_{n_1}) = \max(s(r_i), h)$.
3. Repeat from step 2 until all edges have p assigned.

The root selected is one of the vertices connected with the last assigned edge and will be in the center of the longest path from a leaf to another leaf. Uniqueness is resolved trivially by a random selection of valid roots as done by the algorithm. One could use a procedure that computes the p-weight in both directions, so one does not have to choose a root and is independent of directions until the final deployment of the robots. This would merely complicate the notation, but since the theoretical qualities are identical we chose to present a simpler version of the algorithm. To clarify the notation: edges are noted w.r.t to a vertex r and e_{n_1} is always the edge from vertex r to the root while the order of the remaining edges w.r.t. $p(e_j) - w(e_j)$ is $e_{n_2}, \ldots, e_{n_m}$. We can now state the main theorem, namely the bound for the maximum number of robots needed to clear a tree.

Theorem 1. *Let $h_{max} := \max_{\forall r}(s(r)) > 2$. Let d be the length of the longest path in the tree S^g and $d^* = \lceil d/2 \rceil$. The maximum p-weight of an edge is bound by*

$$\max_{e \in Edges(S^g)}(p(e)) \leq h_{max} + (d^* - 1) \cdot (h_{max} - 3) \tag{2}$$

Proof: The proof proceeds by identifying the worst case for a non-leaf vertex and then starting at the worst case for leaves, construct the worst case tree. Now, consider any vertex r that is not a leaf. Write m for $m(r)$ and E' for $Edges(r) \backslash \{e_{n_1}, e_{n_m}\}$. We are assigning $p(e_{n_1})$ using $\{e_{n_2}, \ldots, e_{n_m}\}$ with $p(e_{n_i})$ assigned for $i > 1$. W.l.o.g. assume that $m \geq 3$, otherwise we have $p(e_{n_1}) = max(p(e_{n_2}), s(r))$. Since $s(r) \leq h_{max}$ and $s(r) = \sum_{e_j \in Edges(r)} w(e_j) + w(r)$ we get:

$$\sum_{e_i \in Edges(r) \backslash \{e_{n_1}\}} w(e_i) \leq h_{max} - 2.$$

Consider the worst case for the last edge e_{n_m} to be cleared from r. All other edges have to be blocked with cost $\sum_{e_i \in E'} w(e_i)$ while we clear sub-regions beyond e_{n_m} with cost $p(e_{n_m})$. Due to the ordering of edges we know that

$p(e_{n_m}) - w(e_{n_m})$ is smaller than for other edges. The worst case costs occur when the p weight of e_{n_m} is as big as possible while at the same time $\sum_{e_i \in E'} w(e_i)$ is large. Trivially we have:

$$\sum_{e_i \in E'} w(e_i) \leq h_{max} - 3$$

For $p(e_{n_m})$ we know:

$$p(e_{n_m}) \leq p(e_{n_i}) - w(e_{n_i}) + w(e_{n_m})$$

due to the ordering. Hence it follows that the worst case occurs when:

$$p(e_{n_m}) = max(p(e_{n_i})),\ w(e_{n_i}) = 1,\ m = h_{max} - 2$$

So we get $p(e_{n_i}) = p(e_{n_j}), \forall j, i > 1$. A quick proof by contradiction shows that the last edge in the worst case is the worst edge. Now the bound for $p(e_{n_1})$ if $p(e_{n_m}) > 2$ becomes:

$$p(e_{n_1}) \leq p(e_{n_m}) + h_{max} - 3.$$

(Note: if $p(e_{n_m}) = 2$ then we may get $p(e_{n_m}) + h_{max} - 3 = h_{max} - 1$ and violate that possibly $p(e_{n_1}) = s(r) = h_{max}$)

We can now use this to derive a bound for the entire tree. We start with the leaves at worst case h_{max} for a safe sweep, so all p-weights at the first stage are h_{max}. Now we assume the worst case for all other vertices. This gives us:

$$\max_{e \in Edges(S^g)} \{p(e)\} \leq h_{max} + (d^* - 1) \cdot (h_{max} - 3)$$

□

The number of robots needed can easily be computed by adding a "virtual" edge to the root vertex and computing its p-weight. Figure 1 illustrates a worst case example with $d = 6, h_{max} = 6$ and the resulting size of the robot team of 16. The actual exploration proceeds by choosing the subtree with the highest $p(e_i) - w(e_i)$ first, then once it is explored blocking it and continuing with the new one in order until the last subtree is cleared. The example from figure 1 is generic and establishes the bound as tight for any d and h_{max}. It, however, only occurs in exactly such an example in the sense that we need all vertices and leaves to have exactly the structure for the worst case as indicated in the proof and figure. When seeking the optimal root vertex for the deployment one can use the same principal method and calculate p-weights in both directions, append a virtual edge to each vertex and then choose the vertex with the smallest p-weight for this virtual edge.

4 Investigation of Performance

To show the practicability of the algorithm we ran the recursive sweep on randomly generated weighted trees. Trees with 20, 40, 60, 80 and 100 vertices,

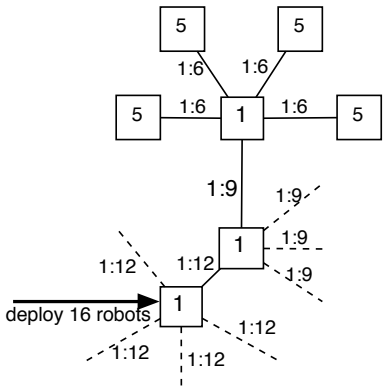

Fig. 1. This is the worst case example for $d = 6, h_{max} = 6, p* = 6+2\cdot3 = 12$. Blocking weight w is indicate left and p-weight right of : on the edge. Each vertex has its weight in its center.

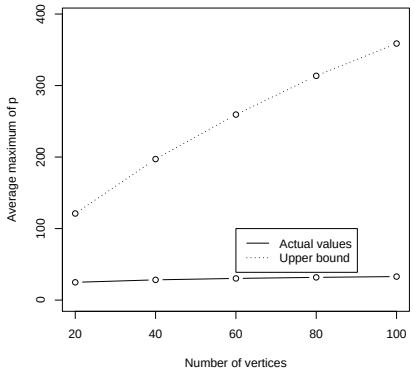

Fig. 2. A comparison of the average upper bound across 1000 weighted trees and actual maximum p-weight for varying number of vertices.

random edges, a random weight for a vertex between 1 and 12 and a random weight for edges between 1 and 6 were generated. For each number of vertices a forest of 1000 trees was created. The average values for the maximum p-weight, d^* and h_{max} are presented in table 4. Figure 2 compares the upper bound computed from d^* and h_{max} with the average maximum value of p.

n Vertices	max($p(e)$)	d*	h_{max}
20	24.865	5.325	25.176
40	28.300	7.784	27.949
60	30.299	9.638	29.607
80	31.781	11.077	31.039
100	32.885	12.306	31.909

Table 1. Results of the experiments. Values are averaged across 1000 random trees. The max($p(e)$) is the largest p-weight occuring in the tree, without any virtual edges attached, while h_{max} is the largest s-weight of the vertices. Note that if the root has the largest s-weight it may be that $h_{max} > max(p(e))$, which is often the case with smaller trees since the root on average has the most edges.

5 Discussion and Conclusion

We presented a first approach for a complex problem and started at dissecting it into the two sub-problems that can be paraphrased into the two major

challenges of *understanding the environment* and then *coordinating the team effort*. We presented a solution to coordinate the team effort with a central instance by solving the weighted graph clearing problem. The benefit is that we can now partition an environment into many simple regions for which one can easily find a sweeping routine for limited range sensors. Particularly suitable are environments such as buildings for which we would suggest a partitioning into regions that are rooms and hallways. The sweeping of a room can then be done by aligning the robots, and then the sensors, on a line. On our path towards a fully working system there are two sets of questions arising. The first set are those regarding the theoretical part, i.e. solving the graph clearing problem without using the heuristic of the constant cycle blocks. First of all, choosing the cycle blocks defines the tree structure and hence influences the resulting strategy. We might get a better strategy if we choose the block w.r.t. to this criteria. Secondly, once we have cleared two regions connected by an edge that has a cycle block it becomes redundant. We could free the block and gain resources and we would want to free these additional resources exactly when we need them, namely when we enter regions with high weights. The second set of questions are those regarding the partitioning. One needs to determine good methods for partitioning depending on the environment and capabilities of the robots. Also we need to conceive criteria for partitions that lead to good strategies on the graph. A first hint of the theorem is that deep trees should be avoided. Understanding the theoretical properties of the weighted graph clearing problem first, might already give us good insight on how to start creating good partitions. Then we can work on devising good partition strategies in coordination with sweeping and blocking methods for particular sensors and close the loop to a working system.

References

1. A. Kolling, S. Carpin, "Multirobot Cooperation for Surveillance of Multiple Moving Targets - A New Behavioral Approach," *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 1311–1316
2. L. E. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," *Autonomous robots*, 12(3):231–255, 2002.
3. S. Sachs, S. Rajko, S. M. LaValle. "Visibility-Based Pursuit-Evasion in an Unknown Planar Environment," *The International Journal of Robotics Research*, 23(1):3-26, 2004
4. L. J. Guibas, J. Latombe, S. M. LaValle, D. Lin, R. Motwani, "Visibility-Based Pursuit-Evasion in a Polygonal Environment," *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999
5. S. M. LaValle, D. Lin, L. J. Guibas, J. Latombe, R. Motwani, , "Finding an Unpredictable Target in a Workspace with Obstacles," *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pp. 737–724
6. B. P. Gerkey, S. Thrun, G. Gordon. "Visibility-based pursuit-evasion with limited field of view," *The International Journal on Robotics Research*, 25(4):299-316, 2006.