School of Engineering Technical Report TR-2009-001
University of California, Merced – November 2009

On Weighted Edge-Searching

Andreas Kolling and Stefano Carpin

University of California
School of Engineering
Merced,CA, USA
http://robotics.ucmerced.edu

Summary. In this document we address some complications regarding the weighted edge-searching problem. This variant of edge-searching extends the original problem by considering situations where multiple searchers may be required to clear a single edge or guard a single vertex. We show that previous work on this topic overlooked a fundamental problem that arises from the addition of weights. As a consequence, a previously developed algorithm that was thought to compute optimal solutions on trees is in fact not optimal. We describe at which points the proofs are incorrect, provide a counterexample, and point out how to address the problem.

1 Introduction

Edge-searching has been investigated in manifold variations in the literature since it was first proposed in [4]. In general, edge-searching is concerned with the detection of omniscient and arbitrarily fast intruders in a graph. A team of searchers tries to find all such intruders by moving through the graph. The potential presence of an undetected intruder is represented by the concept of *contamination*, and the team of searchers removes this contamination thereby clearing the graph and detecting all intruders. An entirely cleared graph subsequently corresponds to the detection of all intruders. In edge-searching contamination resides in edges and can be cleared with one searcher moving along the edge. Searchers in vertices block contamination from spreading towards other edges. This operation is denoted as *guarding* a vertex. The goal in edge-searching is to coordinate all searchers so that the graph can be cleared with the least number of searchers. An annotated bibliography giving an overview of the variations on graph-searching is available in [2]. The purpose of this document is to present and discuss particular results from [1], in which a weighted variant of edge-searching is introduced. This variant additionally allows for multiple searchers to be required to guard a vertex or clear an edge. This extension has significant implications which, as we shall show, have not yet been addressed sufficiently. More precisely, some results presented in [1] regarding optimal algorithms for weighted edge-searching in

trees appears to be not correct. In fact, we show a counterexample in which the algorithm from [1] is suboptimal. We first proceed by presenting the basic notation for weighted edge-searching in Section 2. This is followed by Section 3 which revises results from [1] and presents a counterexample. Finally, we conclude with a pointer towards a remedy of the problem in Section 4.

2 Notation for Weighted Edge-Searching

We recall some of the notation and definitions from [1]. The weighted edge-searching problem is defined on a weighted graph $G = (V, E)$ with a weight function $\omega : V \cup E \to \mathbb{Z}^+$. The graph G is initially fully contaminated. The weight on edges and vertices describes the number of searchers required to clear an edge or guard a vertex, which avoids contamination from spreading into cleared parts. Additionally, it is required that:

$$\omega(x) \geq \omega(e) \text{ for any edge } e \in E \text{ incident to } x \in V. \tag{1}$$

X_i denotes the set of all clear edges at time i (edges are also called links in [1].) Contamination spreads through all vertices that are not considered guarded, i.e. all those with less than $\omega(x)$ searchers placed on them. An edge is cleared by moving $\omega(e)$ searchers along it. The weight function ω can be interpreted as a cost function that determines the cost in terms of the number of searchers needed for successful clearing and guarding actions. Recontamination is the contamination of a previously clear edge. In [1] the following definition of a search step is given.

Definition 1. *A search step of a contiguous search in a weighted network is one of the following two operations (1) place a team of $q \geq 1$ searchers on a node, and (2) move a team of $q \geq 1$ searchers along a link, both operations being performed under the constraint that the set X_i of all clear links after step i must be connected, for any i.*

A search strategy is a sequence of such contiguous search steps that clear all links. The contiguous search numbers is denoted by $cs(G)$ and is the minimum number of searchers needed to solve the contiguous search problem in G, i.e. clear all initially contaminated edges by following a contiguos search strategy. We shall refer to such strategies that clear G with $cs(G)$ as optimal.

3 Search Strategies on Trees

As a result of Section 3 in [1] it follows that in trees there always exists an optimal contiguous strategy that is also monotone, i.e. it never recontaminates any edge. Therefore, an optimal contiguous and monotone strategy is also an optimal contiguous strategy. Based on this result, Section 4 in [1] rightfully

focuses on the design of an algorithms for trees that attempts to compute optimal contiguous strategies that are monotone. In this section $T = (V, E)$ denotes a weighted tree and the goal is to find an optimal contiguous strategy starting at some node $x \in V$, called *homebase*. The minimal number of searchers to clear T starting at x is written $cs_x(T)$.
The part that is most relevant for this document is Lemma 6 from [1] for which the proof is flawed. We first describe the proof as presented in [1] highlighting where the problem is, and we then show that the statement of the lemma is false by giving a counterexample. Consequently, the algorithm presented in Section 4 is suboptimal since Lemma 6 is the basis on which further results in that section depend.
For the claim of Lemma 6 let T_x denote the tree T rooted at $x \in V$. Write $T_x[y]$ for the subtree of T_x rooted in y and let $cs(T_x[y])$ be the contiguous search number of $T_x[y]$ from y[1].

Claim (Lemma 6 from [1]). Let $y_1, y_2, \ldots, y_k$ be the $k \geq 2$ children of y in T_x, and assume, w.l.o.g., that $cs(T_x[y_i]) \geq cs(T_x[y_{i+1}])$[2] for all $i < k$. Then

$$cs(T_x[y]) = \max\{cs(T_x[y_1]), cs(T_x[y_2]) + \omega(y)\}. \tag{2}$$

The proof from [1] is as follows. We present the proof verbatim with our comments added as footnotes:

> Clearly $cs(T_x[y]) \geq cs(T_x[y_1])$, otherwise $T_x[y_1]$ cannot be cleared. If $cs(T_x[y_1]) > cs(T_x[y_2]) + \omega(y)$, then $cs(T_x[y_1])$ searchers suffice to clear $T_x[y]$ by visiting y_1 last among the children of x[3], and by letting $\omega(y)$ searchers occupying node y while the other subtrees are visited. Indeed, from Eq. 1, $\omega(\{y, y_i\}) \leq \omega(y_i) \leq cs(T_x[y_i])$ for every i.
> Hence let $cs(T_x[y_1]) < cs(T_x[y_2]) + \omega(y)$. Let β be a contiguous search strategy which uses $b = cs(T_x[y_2]) + \omega(y) - 1$ searchers to clear $T_x[y]$. If $T_x[y_2]$ is cleared before $T_x[y_1]$, while the $cs(T_x[y_2])$ searchers are clearing $T_x[y_2]$, y will be occupied by at most $\omega(y) - 1$ searchers, and incident to $\{y, y_1\}$ [4]. Thus β does not satisfy the condition of Theorem 1[5]. Similarly, if $T_x[y_1]$ is cleared before $T_x[y_2]$, then while the $cs(T_x[y_1])$ searchers are clearing $T_x[y_1]$, at most $\omega(y)-1$ searcher will be at y since by definitions $cs(T_x[y_1]) \geq cs(T_x[y_2])$, and $b = cs(T_x[y_2]) + \omega(y) - 1$

[1] Here we use the notation as from [1], but writing $cs_y(T_x[y])$ would be more consistent.
[2] The term $cs(T_x[y_{i+1}])$ appears as $cs(T[y_{i+1}])$ in [1] which is a typo.
[3] This should be y, but this is only a typo.
[4] Here a first problem appears. There is no reason why $T_x[y_1]$ cannot be partially cleared, i.e. $T_x[y_1]$ is not fully cleared, but the edge incident to y is not contaminated.
[5] Theorem 1 shows that a contiguous strategy that is optimal and monotone starting at x_0 exists.

[6]. Since y is incident to $\{y, y_2\}$ which is contaminated, y becomes unsafe, and β is hence not monotone. Therefore, strictly more than $cs(T_x[y_2]) + \omega(y) - 1$ searchers are required for a monotone strategy if $cs(T_x[y_1]) < cs(T_x[y_2]) + \omega(y)$. One the other hand, $cs(T_x[y_2]) + \omega(y)$ searchers clearly suffice to clear $T_x[y]$ by visiting y_1 last among the children of y, since at least $\omega(y)$ will stay at y making it safe while the other subtrees are visited. Links incident to y are cleared since, again from Eq. 1, $\omega(\{y, y_i\}) \leq \omega(y_i) \leq cs(T_x[y_i])$ for every i.$\square$

The important fact to notice is that $T_x[y_1]$ and $T_x[y_2]$ can be cleared partially which is not considered in the original argument. For weighted edge-search this turns out to be important. Let us first give the intuition for this before we present a counterexample to the claim of Lemma 6. Instead of using $\omega(y)$ searchers in y one can prevent recontamination by partially clearing one of the subtrees $T_x[y_1]$ or $T_x[y_2]$ and possibly reach a state in which searchers on a vertex $z \in V$ within one of these subtrees such that $\omega(z) < \omega(y)$ suffice to prevent recontamination. Then one can proceed to fully clear the other subtree but only using $\omega(z)$ plus the searchers needed for the other subtree. Notice that from the ordering it is not clear which subtree should be cleared first given this additional possibility of moving the guarded vertices into the subtree by partially clearing it. Notice also that instead of one vertex z in one of the subtrees we may have multiple vertices $z_1, z_2, \ldots, z_m$ s.t. $\sum_{i=1}^{m} \omega(z_i) < \omega(y)$. It is sufficient to say that the fact that we can split up the $\omega(y)$ searchers from y causes this complication. This argument would not be apply if $\omega(y) = 1$ and hence only applies to the general weighted case. The consequence is that weighted edge-searching is considerably more complicated than non-weighted edge-searching. Let us now present the counterexample.

Consider the subtree $T_x[y]$ in Fig. 1. Clearly, the subtrees $T_y[y_1]$ and $T_y[y_2]$ need 7 and 5 searchers respectively. With $\omega(y) = 5$ Eq. 2 claims that $cs(T_x[y]) = 10$. Yet in Fig. 1 we show a valid sequence of search steps that clears the entire tree but uses only 8 searchers, all initially placed on y. This strategy is contiguous and monotone.

Since the counterexample shows that Lemma 6 in [1] is indeed a false claim it follows that the algorithm in Section 4 is not optimal. The same counterexample applies there. There is also another minor conceptual problem in the presented algorithm from Section 4 in [1]. According to the definition of recontamination, adapted from non-weighted edge-searching, the weight of a leaf vertex is not relevant for the number of searchers that need to be sent into it. This is because a leaf vertex never needs to be guarded. Contamination resided in edges and once the edge is cleared there is no possibility of recontamination left since the leaf has only one edge. Let vertex y be a leaf incident to x with edge e. Then only $\omega(e)$ are required to be moved across e towards y even if $\omega(e) < \omega(y)$, since y does not need to be guarded after e

[6] Similarly as before, it is possible that one subtree is partially cleared before the other is fully cleared.

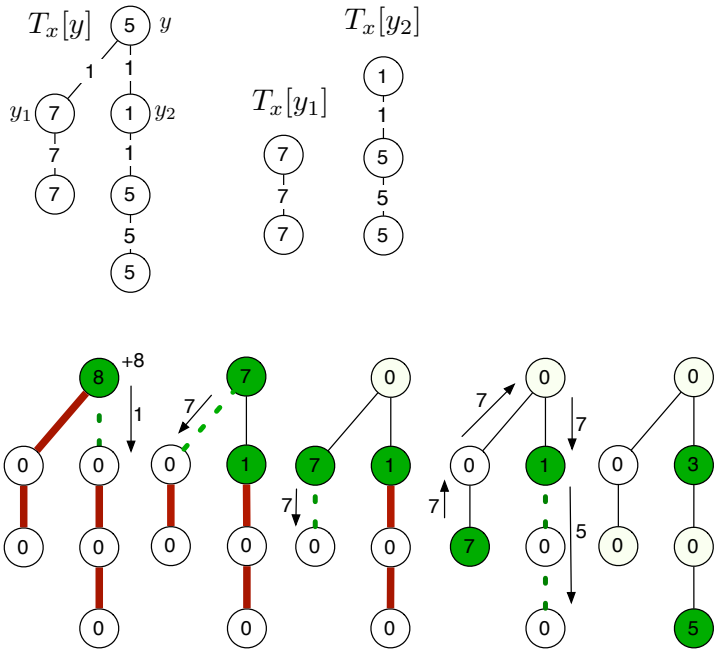

Fig. 1. A subtree $T_x[y]$ is shown on the upper left with its subtree $T_x[y_1]$ and $T_x[y_2]$. Below we show a contiguous and monotone search strategy that clears all initially contaminated edges. First 8 searchers are placed on y and one is moved across $\{y, y_2\}$ to clear it. Then 7 searchers clear $T_x[y_1]$. Contaminated edges are shown as thick lines and thin lines are cleared edges. The number of searchers occupying a vertex are written within it while the number of searcher about to move across an edge are marked with an arrow. The edge that is about to be cleared by a move along it is drawn as a dashed line.

is cleared. Correcting the algorithm from $\lambda_x(e) = \omega(y)$ to $\lambda_x(e) = \omega(e)$ will solve this problem. This is again a complication arising from weights, since in the non-weighted case the searcher traversing e will also automatically guard y.

4 Conclusion

From the above it follows that the problem of finding an optimal algorithm for computing contiguous strategies in trees remains open. The fundamental problem that such an algorithm has to address is how to keep track of the guarding cost of subtrees and how various guarding costs can be achieved. This may involve multiple times entering and exiting partially cleared subtree. It turns out a similar problem for Graph-Clear, another graph-searching problem, has been addressed in [3]. Therein the concept of cut sets and cut

sequences are introduced. These cut sets and cut sequences keep track of the intermediate states that the subtrees can have and the cost to achieve these states as well as the costs to maintain the state without recontamination (referred to as *blocking* cost.) The same approach as used and developed in [3] can be adapted for weighted edge-searching and lead to an optimal algorithm on trees. Only the computation of the clearing cost and blocking cost need to be modified. A detailed presentation of such an algorithm is, however, subject to further work.

References

1. L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proc. of the fourteenth annual ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, New York, NY, USA, 2002. ACM Press.
2. F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.
3. A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 2009. accepted for publication.
4. 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.