2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Fast Guaranteed Search With Unmanned Aerial Vehicles

Andreas Kolling[1], Alexander Kleiner[2], and Piotr Rudol[2]

Abstract— **In this paper we consider the problem of searching for an arbitrarily smart and fast evader in a large environment with a team of unmanned aerial vehicles (UAVs) while providing guarantees of detection. Our emphasis is on the fast execution of efficient search strategies that minimize the number of UAVs and the search time. We present the first approach for computing fast guaranteed search strategies utilizing additional searchers to speed up the execution time and thereby enabling large scale UAV search. In order to scale to very large environments when using UAVs one would either have to overcome the energy limitations of UAVs or pay the cost of utilizing additional UAVs to speed up the search. Our approach is based on coordinating UAVs on sweep lines, covered by the UAV sensors, that move simultaneously through an environment. We present some simulation results that show a significant reduction in execution time when using multiple UAVs and a demonstration of a real system with three AR.Drone 2.0.**

I. Introduction

Guaranteed search is one of the many applications for which recent progress in the development of cheaper and improved unmanned aerial vehicles (UAVs) is leading to new exciting possibilities. Especially for search and rescue in large and dangerous areas, UAVs offer an unparalleled advantage of being highly mobile and truly expendable with a cost of only a few hundred dollars for current consumer platforms. The downside of these platforms, however, is their severe limitation with regard to energy and flight time. Prior work on guaranteed search in robotics has almost exclusively focused on the computation of search strategies that minimize the number of searchers. The recent survey on search and pursuit-evasion in robotics [3] has identified the minimization of time and travel distance for guaranteed search strategies as an important problem that has not received much attention. We concur with this assessment especially when considering UAVs instead of ground, underwater vehicles or pan-tilt-zoom cameras, which suffer less from energy limitations. The goal of this paper is to provide a first step towards the consideration of execution time for search strategies, the computation of fast and efficient strategies, and the demonstration of powerful search capabilities with real systems using teams of UAVs.

One of the few results regarding the minimization of time for search strategies on graphs are found in [2]. Therein it is shown that the computation of the minimum time to search a graph with a team of searchers is strongly NP-Complete even for stars, trees, two-vertex graphs and most other types of graphs except for paths and cycles. In addition, minimizing time has not only received very little attention in robotics, but also the graph searching literature on this topic, see [6], is sparse and we do not have such a large body of research available as for minimizing the number of searchers. As a consequence, our approach in this paper is that of speeding up the execution of given strategies rather than computing time optimal strategies from scratch. This has the advantage of simplifying the problem in addition to being able to use the approach for any strategy, even those modified by a human operator or modified to consider criteria other than the number of searchers, such as terrain difficulty, online adaptation, or probabilistic considerations.

The work presented here is based on extensive prior work building towards real search systems which we review briefly in Section II and Section III. This is followed by our contribution towards parallelizing the execution of strategies in Section IV and experiments in V. Finally, we conclude with a discussion in Section VI.

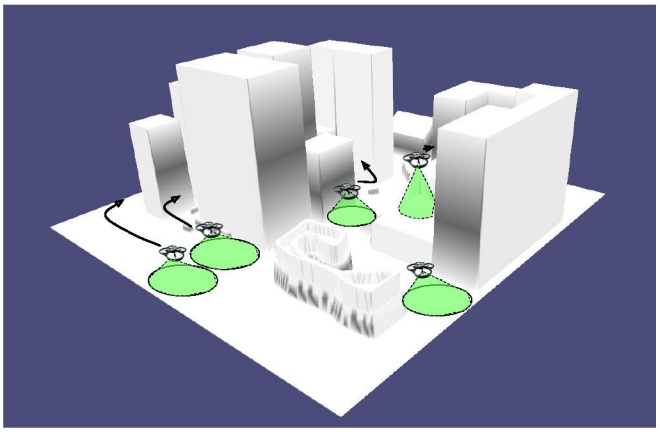

Fig. 1. The motivation for this work is the fast execution of guaranteed search strategies with a team of UAVs in large and complex environments.

II. Related Work

Guaranteed search and pursuit-evasion problems have been an active research area in robotics and other related fields such as graph theory for an extended period of time. From a robotics perspective the survey by Chung *et al.* [3] provides an excellent summary. Fomin *et al.* [6] provide a good overview of the problem from a graph theoretical perspective.

This paper work is based on our prior work in guaranteed search and an extension thereof [15], [16], [11], [9]. In [16],

[1]Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, S1 3JD United Kingdom andreas.kolling@gmail.com

[2]Department of Computer and Information Science, Linköping University, 58738 Linköping, Sweden

This work is partially supported by the Swedish Foundation for Strategic Research and the Excellence Center at Linköping and Lund in Information Technology (ELLIIT), the Swedish Foundation for Strategic Research CUAS Project, and the EU FP7 project SHERPA, grant agreement 600958.

[11] Kolling *et al.* introduced a formalism and algorithms to coordinate the motion of robots moving on sweep lines, reusing prior graph-based approaches [15]. The time it takes to execute these strategies was first evaluated in simulations in [9], revealing considerable potential for improvements. We shall discuss the connection to this work in more detail in Section III.

Other work that considers line-based abstractions is [5]. Therein Efrat *et al.* consider an approach where multiple UAVs, each with an unlimited range sensor, are arranged in a single movable polygonal chain operating in a simply connected environment. Their algorithm for computing motion strategies runs in $O(n^3)$, an improved version of the algorithms runs in $O(n^2)$ time [17]. Another similar approach that assumes limited sensing range is presented in [1] where the authors present a system where multiple UAVs trap faster intruders, once they detect them, by forming a surrounding chain. Also closely related to our work, Durham *et al.* presented a distributed algorithm guaranteeing complete coverage of the frontier between cleared and contaminated areas during expansion [4].

To the best of our knowledge no work considering the fast execution of guaranteed search strategies has been pursued in the robotics community. The closest related work is [8] which minimizes distance by improving the average-case performance for non-adversarial motion models.

III. Search Strategies With Lines

In this section we briefly describe the basics of our line-based search approach, first presented in more detail in [16] and [11]. The goal is to find an unknown number of targets in an environment represented by a polygon, grid, or elevation map. For this paper we assume a simply-connected 2D polygon. In [16] it was shown how to apply the resulting strategies to multiply-connected and even 2.5D environments and the same approach can be applied for the fast strategies we compute here. Regarding the targets, we make the same assumptions as is customary for pursuit-evasion problems, namely that targets are evaders moving at unbounded speed, are omniscient and are able to evade the searcher optimally. As a consequence targets can conveniently be represented with contamination which is cleared by the searcher as they proceed to clear the environment. Contamination spreads immediately whenever possible, simulating the fact that omniscient and fast targets will exploit any possibility to move into areas not covered by sensor or obstacles. Contamination simplifies the target model and allows the efficient computation of strategies at least for simply-connected environments [15], [16]. In our line-based approach searchers clear the environment by moving together on sweep lines, necessitated by the limited sensing range. These sweep lines are simply lines between the obstacle boundaries of the environments. As a sensor we assume a circular footprint, such as a downward facing camera of a UAV, with a radius r_s. The environment is given by a simply-connected polygon $P = \{v_1, \ldots, v_n\}$, with n vertices and edges, written $e_i = [v_i, v_{i+1}]$, $i = 1, \ldots, n$. The indices of the obstacles, i.e., the polygon edges, are assumed to be circular and we identify $i + n$ with i.

The original formulation of this approach in form of the Line-Clear problem was given in [11]. Therein any number of sweep lines can be placed in the environment and moved around to clear more of it. To minimize the number of searchers needed to cover these lines with their sensor footprint, however, it is was shown it suffices to consider strategies that move at most one line at any time. This approach was adopted in all subsequent work on line-based search, namely [13], [14], [16]. In order to enable fast line-based search we have to consider the concurrent motion of multiple sweep lines. In the following, we will first describe sequential strategies, such as the one computed in [16] and then show how to parallelize these, possibly using additional searchers.

Suppose we are given a line-based search strategy, e.g. computed by the algorithm from [16]. It was shown therein that such search strategies can be represented by an obstacle index sequence, $o_1, \ldots, o_n$ containing all indices from P. The search strategy proceeds by first setting up a sweep line between obstacles o_1 and o_2, and then subsequently splitting the sweep line on the next obstacle index in the sequence. This is best illustrated with an example, seen in Fig. 2. The corresponding obstacle sequence for the strategy shown therein is $\{o_1 = 2, o_2 = 3, o_3 = 4, o_4 = 17, o_5 = 1, o_6 = 18, o_7 = 22, o_8 = 19, o_9 = 21, o_{10} = 20, o_{11} = 5, o_{12} = 16, o_{13} = 6, o_{14} = 7, o_{15} = 11, o_{16} = 10, o_{17} = 8, o_{18} = 9, o_{19} = 12, o_{20} = 15, o_{21} = 13, o_{22} = 14\}$ computed with the sensing range r_s shown in Fig. 2. The procedure for splitting an existing sweep on a new obstacle is quite simple. For the computation of the strategy a point on the new obstacle that minimizes the cost of the split in terms of the number of searchers was computed. The existing sweep line is simply moved towards this point and then split into two sweep lines. One or both of these sweep lines can be of zero length if the obstacle indices are adjacent. If both are zero, then then no new sweep lines are formed. This procedure is described in more detail in [16].

Another representation of this strategy that is more convenient for our purposes is as a surveillance tree. Surveillance trees were introduced in [15] and are trees $T = (V, E)$ with a set of vertices V and edges $E \subset V \times V$ and a weight function $w : V \times E :\rightarrow \mathbb{N}$ that associates a cost to each vertex and edge. This cost represents the number of robots needed to clear a vertex and block an edge. Hence, the surveillance tree can represent the cost of the block lines in the environment and the cost of the split by associating blocks to edges and splits to vertices. We construct such a surveillance tree by creating a node in V for every o_s and using $c(o_s)$, the cost of splitting a sweep line on obstacle o_s at step s in the obstacle sequence, as its weight. The edges are given by the progression of the sweep lines and their weights $b(o_s)$ for the cost of the blocking sweep line prior to the split. This simple construction is illustrated in Fig. 3 for the example environment from Fig. 2.

The representation as a surveillance tree describes the

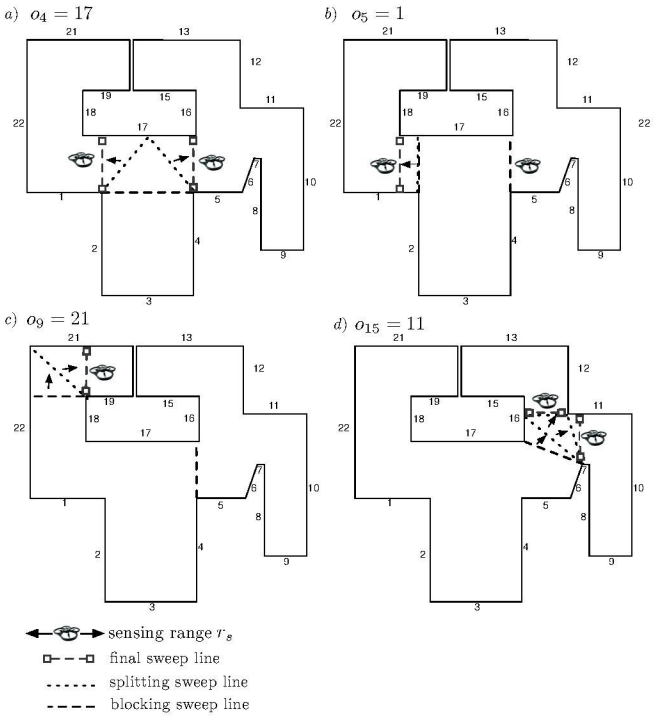

Fig. 2. An example of a sweep line strategy with a) showing the first split of a blocking sweep line between obstacles 2 and 4 splitting on obstacle $17 = o_4$. Subsequently the left and right side of the split move to a shorter final sweep line for this split. In b) the final sweep line from on the left is further split onto $o_5 = 1$ into two sweep lines, one of which is zero, since $o_5 = 1$ is adjacent to $o_1 = 2$ and moved to a final blocking sweep line between 17 and 1. In c) the search strategy continues to the left side until it is cleared and the sweep line on the left disappears. In d) when splitting on o_{15} another two non-zero sweep lines are created.

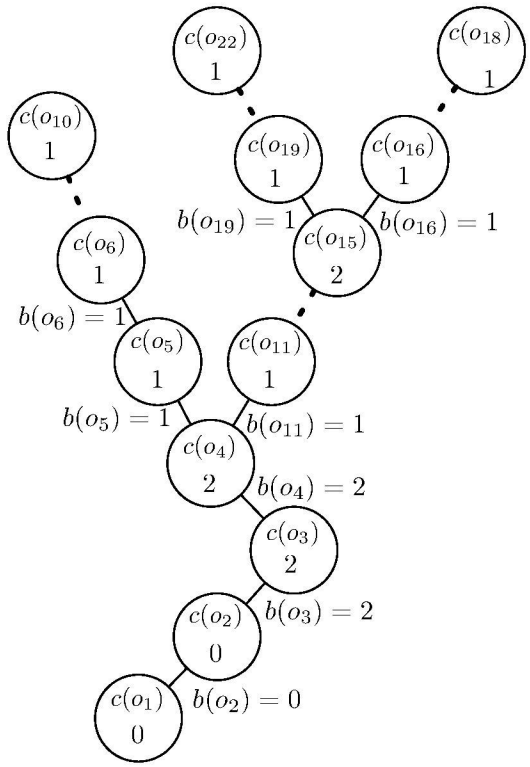

Fig. 3. A surveillance tree representing the line-based search strategy shown in 2. The cost of the vertices are shown inside the vertices and the cost for the edges are shown to the side of the edges. Dashed lines indicate a continuation of the tree, omitting some nodes for a more concise presentation.

temporal dependencies between the sweep lines, which is important for parallelization. At every split into two non-zero sweep lines, represented by a node with two children in the surveillance tree, there is an opportunity to continue both lines in parallel.

IV. Parallelization of Strategies

In this section we discuss how to speed up the execution of a given line-based strategy. A purely sequential execution of the obstacle sequence, as in [16], assigns searchers to the current step based on their distance to the sweep line. Once all searchers reach the line they cover it with sensors and push it forward. In the meantime all blocking sweep lines that may exist are covered by searchers that remain stationary on these lines. We can interpret this sequential execution as putting a delay on the sweep lines after each split. So before a sweep line splits on another obstacle o_s at step s it has remained stationary at its blocking locations for some delay $\delta_s \geq 0$. The delay in a sequential execution is given by the time it takes until all steps $i < s$ have been executed and the assigned robots, which may have been at different locations of previously moved sweep lines have arrived at o_s. This case is seen in Fig. 4 for the strategy in 2. Here the searchers not only have to wait for previous steps to finish but also for searchers traveling between steps for step 15.

It is easy to notice that some of these delays can be stripped out without increasing the cost. This is shown in Fig. 5 which shows the same strategy permitting simultaneous motion of more than one sweep line without requiring additional robots. Notice that due to the increased delay δ_{15} not much is gained in terms of overall speed of execution in the first steps, but the last steps are considerably faster. While this case trivially illustrates the shortcoming of preventing simultaneous motion whenever a blocking line can be moved further at no additional cost, the main problem will become the simultaneous motion of lines whenever the cost of motion increase. We will then require additional searchers in order to improve the parallel execution. In fact, even this simple example can already benefit from an additional searcher to remove the delay δ_{15} entirely, as seen in Fig. 6.

The above example illustrates one key problem. Unless resources, i.e., additional searchers, are committed right from the start for every sweep line so that they can arrive when needed we will get delays introduced by moving resources between the sweep lines, as seen by comparing Fig. 5 and Fig. 6. Only if robots on lines move significantly slower then we can have robots be reused without losing time due to long travel distances between different sweep lines in far away parts of the environment. Therefore, the fastest possible execution of a strategy is achieved by using the number of robots given by $c^p(o_1)$, the parallelized cost computed

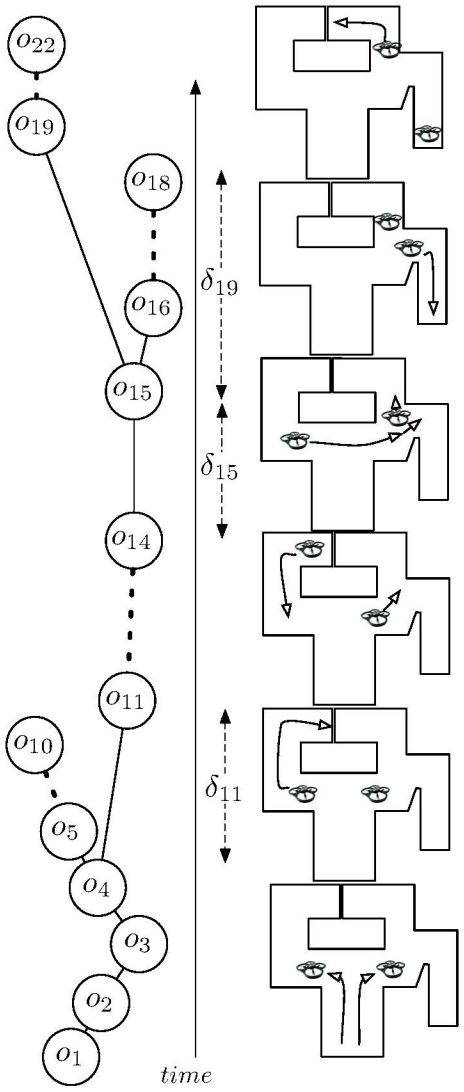

Fig. 4. The execution of a strategy as time progresses, illustrated by placing the surveillance tree on a timeline. The delays δ_{11} and δ_{19} result from the sequential execution with steps 11 and 19 having to wait until the previous steps are finished. The delay δ_{15}, however, is due to the travel time that the searcher needs to be able to contribution to step 15 while having previously executed steps 5 to 10.

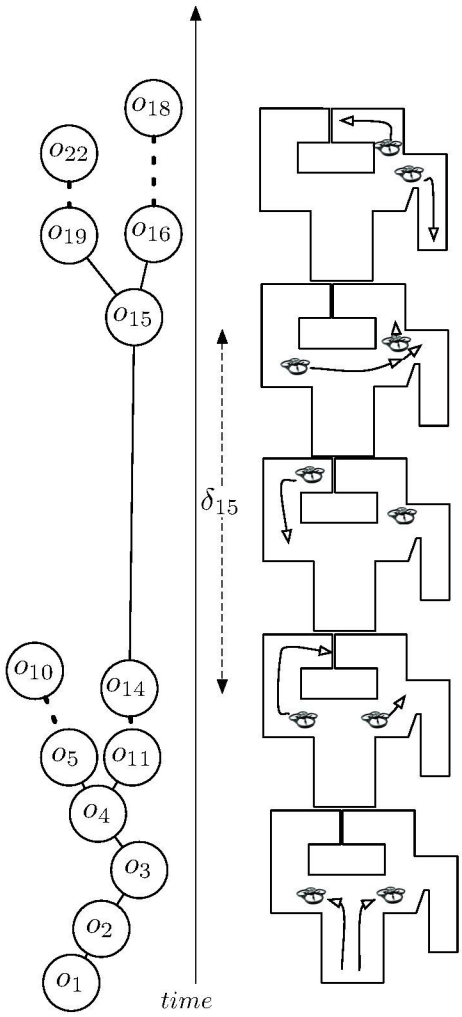

Fig. 5. An improved strategy based on Fig. 4 that considers the parallel execution of sweep lines and thereby removing some delays without requiring additional UAV.

recursively as follows:

$$c^p(o_s) := \max\left\{ \sum_{o \in children(o_s)} c^p(o), c(o_s) \right\},$$

with $children(o_s)$ containing the obstacle indices in T that are the children of the node for o_s.

a) Depth-first Committed Parallelization: At this point we know the minimum cost to execute a strategy entirely sequentially, essentially without regard to minimizing time, and as well as the usually much higher cost of executing it completely in parallel. In between these two extremes lie a range of possible ways to execute the search strategy, each utilizing or leading to different delays δ_s.

Let us now write t_s for the time it takes to execute step s, i.e., split onto o_s. This time can be computed straightforward by using the locations of robots on the sweep line, as done in [16] or using a simulator with simulated robots following planned trajectories as in [9].

In the following, we first adopt a depth-first parallelization perspective. Depth-first search strategies are one of the standard search approaches, but have also been used to compute pursuit-evasion strategies in [10], [7], [12] but to the best of our knowledge not with regard to time. Suppose we are given $r < c^p(o_1)$ searchers, not sufficient for complete parallelization. In this case we have to introduce some non-zero delays δ for some of the steps.

To do this we introduce a separate sequential depth-first cost $c^d(o_s)$ computed recursively as follows:

$$c^d(o_s) = \max\{\min\{c^d(o_s^l) + b(o_s^r), c^d(o_s^r) + b(o_s^l)\}, c(o_s)\},$$

with $children(o_s) = \{o_s^l, o_s^r\}$ as before with the obvious exception that $o_s^l = 0$ or $o_s^r = 0$ and $b(0) = 0$ and $c^d(0) = 0$ when there are only one or no children.

The depth-first parallelization now proceeds according to Alg. 1 setting the δ delays to 0 wherever parallel execution

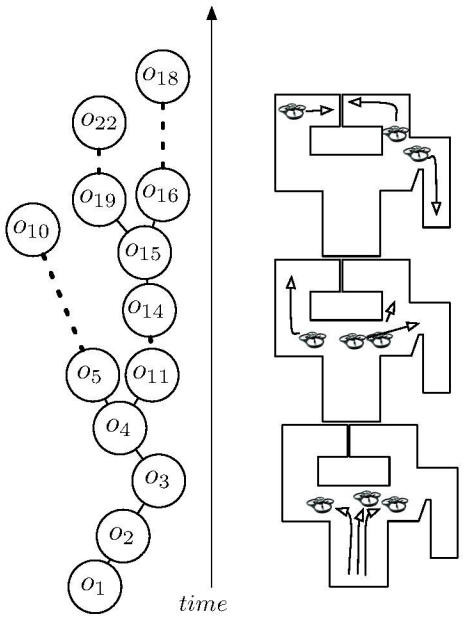

Fig. 6. An improved strategy based on Fig. 4 that uses an additional UAV to further reduce the time required to execute the search.

Algorithm 1 $Depth\_First\_Par(s, k, \delta)$

1: $\delta_s \leftarrow \delta$
2: $\{o_l, o_r\} \leftarrow children(o_s)$
3: **if** $c^p(o_s) \leq k$ **then**
4: $\quad t \leftarrow Depth\_First\_Par(o_l, c^p(o_l), 0)$
5: $\quad t \leftarrow \max(t, Depth\_First\_Par(o_r, c^p(o_r), 0))$
6: **else**
7: $\quad$ **if** $c^d(o_l) + b(o_r) \leq c^d(o_r) + b(o_l)$ **then**
8: $\quad\quad t \leftarrow Depth\_First\_Par(o_l, k - b(o_r), 0)$
9: $\quad\quad t \leftarrow t + Depth\_First\_Par(o_r, k, t)$
10: $\quad$ **else**
11: $\quad\quad t \leftarrow Depth\_First\_Par(o_r, k - b(o_l), 0)$
12: $\quad\quad t \leftarrow t + Depth\_First\_Par(o_l, k, t)$
13: $\quad$ **end if**
14: **end if**
15: **return** $t + t_s$

of both subtrees is possible. Once the algorithm enters line 4 the further calls to $Depth\_First\_Par$ will all lead to completely parallelized subtrees. This is the depth-first bias towards parallelization in favor of the smaller and deeper subtrees.

V. Experiments and Results

We ran experiments in simulation as well as a demonstration of a real system with three AR.Drone 2.0, a consumer electronics device available commercially. The positions of the UAVs were controlled using a Vicon motion capture system. The code used for these experiments is made available at http://code.google.com/p/guaranteed-search/ under a GNU GPLv3 license.

The simulation environments used are shown in Fig. 7 and Fig. 9. Fig. 8 shows the result of applying Alg. 1 and executing lines in parallel. The larger maze benefits more from further searchers, as one would expected, but with diminishing returns.

The slightly more realistic environment from Fig. 9 shows a similar trend, but requires significantly less time due to the shorter paths between most locations compared to the mazes, even at comparable sizes. Town has a size of $76.95m$ x $76.95m$ with a UAV sensing range of $2.82m$ while Full Maze has a size of $19.2m$ $12.16m$ and a sensing range of $0.7m$.

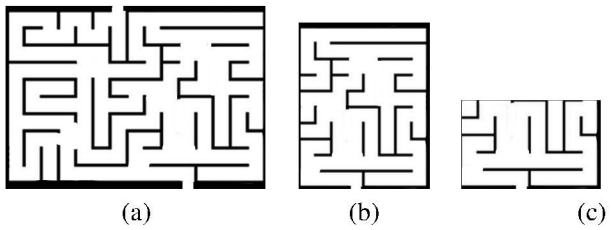

Fig. 7. The three different maze environments used. In (a) we have the Maze, in (b) the Half Maze, and in (c) the Quarter Maze.

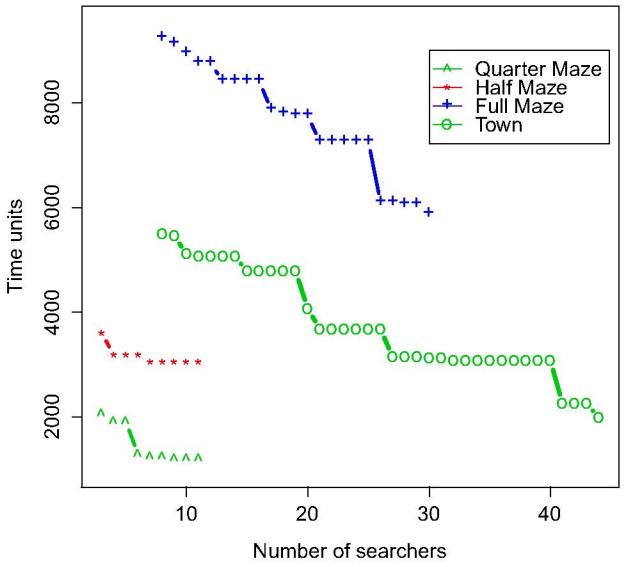

Fig. 8. The time that a given number of searchers needs to clear the maze using Alg. 1 to parallelize the sweep lines.

For the real system we used the environment from Fig. 2 and computed trajectories for the UAVs for three different scenarios. The first scenario was the sequential execution of a strategy computed as in [16] with two UAVs as seen in 4. This strategy took 72 seconds to execute with the real UAVs. A parallelization, applying Alg. 1, lead to the parallelized strategy shown in Fig. 5 with a minor improvement to 64 seconds due to the final few steps being executed in parallel. With one additional UAV Alg. 1 resulted in a strategy that was executed in 43 seconds, a much larger improvement. The accompanying video shows the third scenario with three

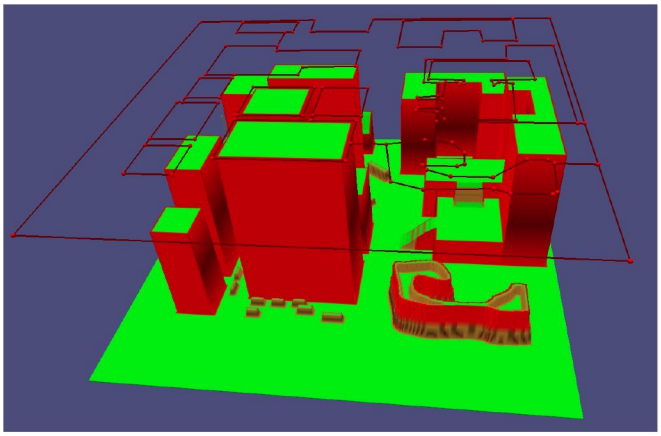

Fig. 9. An artificial town, denoted as Town, and the resulting simply-connected polygon computed as in [16].

UAVs. While these demonstrations do not reveal any particular insights they do demonstrate feasibility and motivate further experiments, especially outdoor search utilizing GPS.

VI. Discussion and Conclusion

The presented approach and algorithm are but a very small step towards addressing the problem of fast search with UAVs. Nonetheless, the motivation from the possibility of a real system should provide a good basis for further work on this problem. Our initial results strongly indicate that significant improvements can be made to speed up search strategies and scale to large environments. The additional searchers added in our simulations and the real experiments reduced the search time significantly. Outdoor experiments with ten UAVs in environments that comprise multiple buildings are easily envisioned at this point. Despite the NP-completeness of the problem of minimizing search strategies on even simple types of graph we have shown that simple heuristics can already provide significantly faster strategies.

While the algorithm we presented here follows the depth-first approach that some graph-based algorithms, e.g. [10], [7], there is clearly room for improvement. As noted in the survey [3] approximation for hard search problems have not yet received much attention and are a promising avenue for further work. Yet, the most interesting direction, rather than trying to push the limits of minimizing time or the number of robots, is to work on more adaptive and decentralized approaches with less strict assumptions. One such step has been made in [14] wherein robots did not need a map nor sophisticated localization to search an environment.

References

[1] S. Bopardikar, F. Bullo, and J. P. Hespanha. On discrete-time pursuit-evasion games with sensing limitations. *IEEE Transactions on Robotics*, 24(6):1429–1439, 2008.

[2] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for pursuit-evasion problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 59–66, 2009.

[3] T. Chung, G. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.

[4] J. Durham, A. Franchi, and F. Bullo. Distributed pursuit-evasion without mapping or global localization via local frontiers. *Autonomous Robots*, pages 81—95, 2012.

[5] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 927–936, 2000.

[6] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

[7] G. Hollinger, A. Kehagias, and S. Singh. GSST: Anytime guaranteed search. *Autonomous Robots*, 29(1):99–118, 2010.

[8] G. Hollinger, S. Singh, and A. Kehagias. Improving the efficiency of clearing with multi-agent teams. *The International Journal of Robotics Research*, 29(8):1088–1105, 2010.

[9] A. Kleiner and A. Kolling. Guaranteed search with large teams of unmanned aerial vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013. accepted for publication.

[10] A. Kleiner, A. Kolling, M. Lewis, and K. Sycara. Hierarchical visibility for guaranteed search in large-scale outdoor terrain. *Autonomous Agents and Multi-Agent Systems*, pages 1–36, 2011.

[11] A. Kolling. *Multi-Robot Pursuit-Evasion*. PhD thesis, University of California, Merced, December 2009.

[12] A. Kolling and S. Carpin. The GRAPH-CLEAR problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1003–1008, 2007.

[13] A. Kolling and S. Carpin. Surveillance strategies for target detection with sweep lines. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5821–5827, 2009.

[14] A. Kolling and S. Carpin. Multi-robot pursuit-evasion without maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3045–3051, 2010.

[15] A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 26(1):32–47, 2010.

[16] A. Kolling and A. Kleiner. Multi-uav motion planning for guaranteed search. In *Proceedings of the Twelth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 79–86, 2013.

[17] X. Tan. Sweeping simple polygons with the minimum number of chain guards. *Information processing letters*, 102(2-3):66–71, 2007.