Computing time-optimal clearing strategies for pursuit-evasion problems with linear programming[⋆]

Hongyang Qu, Andreas Kolling, and Sandor M. Veres

Dept. of Automatic Control and Systems Engineering, University of Sheffield, UK
{h.qu,a.kolling,s.veres}@shef.ac.uk

Abstract. This paper addresses and solves the problem of finding optimal clearing strategies for a team of robots in an environment given as a graph. The graph-clear model is used in which sweeping of locations, and their recontamination by intruders, is modelled over a surveillance graph. Optimization of strategies is carried out for shortest total travel distance and time taken by the robot team and under constraints of clearing costs of locations. The physical constraints of access and timely movements by the robots are also accounted for, as well as the ability of the robots to prevent recontamination of already cleared areas. The main result of the paper is that this complex problem can be reduced to a computable LP problem. To further reduce complexity, an algorithm is presented for the case when graph clear strategies are a priori available by using other methods, for instance by model checking.

1 Introduction

Search and pursuit-evasion problems appear in a variety of formulations in the literature. A survey of methods can be found in [2] where taxonomies of approaches and assumptions based on attributes of searchers, targets and the environments are reviewed. A large class of problems assumes that there is a graph-based description, an “abstraction”, of the environment which models it into locations, represented by vertices, and passages between locations, represented by the edges of a graph. Once such an abstraction is available one can apply the techniques developed in graph-searching for robotic search. Probably the first related analysis of graph searching has been published in [12]. The searchers' objective was to catch evaders along the edges the graph with the minimum number of searchers. A large number of variants of graph searching problems appeared which differed in their assumption on the mobility and cost models of a team of search robots clearing edges and vertices of a graph. A review of these is found in [3]. Many of the methods and results developed in the context of graph-searching were primarily concerned with minimizing the number of searchers rather than the distance or time travelled. In this paper we present

[⋆] This work was supported by the EPSRC project EP/J011894/2.

a solution for minimizing time, in addition to the number of searchers, using an linear programming formulation for a graph-based pursuit-evasion problem.

The recent framework of interest in this paper is the Graph-Clear(GC) model [5, 7, 10] for pursuit-evasion problems with teams of robots. GC is particular useful for search problems where multiple robots cooperate to detect intruders in complex environments with limited sensing capabilities. Following the GC approach, in this paper we consider the problem of searching a graph for an unknown number of omniscient and smart targets moving with unbounded speed. Such targets represent a worst-case scenario and are represented by the formal concept of contamination [12], defined in Section 3. The searchers can execute actions that clear contamination or block it from spreading through parts of the environments. These actions have an associated cost that reflects the number of robots needed to prevent targets from passing through undetected.

Linear programming (LP) techniques have previously been applied to robot pursuit-evasion problems in [14] and [13]. In [14] the authors considered polygonal environments partitioned into convex regions. From a region a searcher is able to detect targets in all other visible regions. The resulting visibility relations between regions are considered in the LP formulation. Searchers can move between adjacent regions at each time step but, to reduce the complexity, multiple pursuers cannot occupy the same region. A receding horizon approach was applied in order to solve the LP formulation, albeit without guarantees for optimality. Our work presented here is a continuation of [13] which considered the computation of strategies for Graph-Clear with the minimal number of searchers using an LP formulation. In this paper we focus on computing time optimal strategies. In contrast to [14] target detection in our scenario may require multiple searchers and our environmental model is a graph with vertices that do not necessarily represent convex regions and can exhibit more complex neighborhood relations. In general, the problem of computing time optimal strategies is shown to be harder than minimizing the number of searchers [1], e.g., it is already NP-hard on star-shaped trees.

In the next section a formal description of surveillance graphs and of the GC problem is provided. Section 3 develops the LP model to obtain minimal total travel time of the robots to clear a graph in a physically feasible way. Section 4 deals with optimisation heuristics to reduce the computational complexity of the optimal strategy. Section 5 applies the LP system to find the shortest execution plan for a given clear strategy. Section 6 presents a computational example and the last section concludes the paper.

2 Pursuit-Evasion problem

For our purposes we adopt the model called Graph-Clear introduced and formalized in [10]. Therein the environment is given by a surveillance graph which is a weighted graph $G = (V, E, w)$ with an undirected graph with vertex set V, edge set E, and $w : V \cup E \to \mathbb{N}^+$ as a weight function. To model contamination and how it is spreading the vertices and edges have an associated state with vertices either being *clear* or *contaminated* and edges either being *clear*, *contaminated*, or *blocked*. These are abbreviated as $\mathcal{R}, \mathcal{C}$, and $\mathcal{B}$ for clear, contaminated, and blocked, respectively. The state of the surveillance graph with n vertices and m edges is then given by $\nu \in \mathcal{V}(G) = \{\mathcal{R}, \mathcal{C}\}^n \times \{\mathcal{R}, \mathcal{C}, \mathcal{B}\}^m$. As a shorthand, we write $\nu(v_i)$ and $\nu(e_j)$ for the state of a particular vertex or edge. Contamination spreads on recontamination paths. These are paths of vertices and edges on which no edge is blocked. Finally, searchers can execute actions which are either a sweep on a vertex or a block on an edge. The executed actions on G can be represented by $a = \{a_1, \ldots, a_{n+m}\} \in \{0,1\}^{n+m} = \mathcal{A}(G)$ where a 1 for an associated vertex indicates a sweep and a 1 for an associated edge indicates a block. The cost of an action a is given by $c(a) = \sum_{i=1}^{n} a_i w(v_i) + \sum_{j=1}^{m} a_{n+j} w(e_j)$, representing the number of robots needed to execute all sweeps and blocks for the action. The spread of contamination and the clearing of actions can now be formalized via a transition function ζ, defined in [10] as follows:

Definition 1 (Transition function). *Let G, $\mathcal{V}(G)$ and $\mathcal{A}(G)$ be defined as above. The* transition function *ζ maps a state and an action into a new state:*

$$\zeta : \mathcal{V}(G) \times \mathcal{A}(G) \to \mathcal{V}(G).$$

Given $a \in \mathcal{A}(G)$ and $\nu \in \mathcal{V}(G)$, the new state ν' is defined as follows:

1. *if $a_{n+j} = 1$, $1 \leq j \leq m$, then $\nu'(e_j) = \mathcal{B}$*
2. *if $a_i = 1$, $1 \leq i \leq n$, then $\nu'(v_i) = \mathcal{R}$*
3. *if $\nu_{n+j} = \mathcal{B}$, $a_{n+j} = 0$, $1 \leq j \leq m$, and no recontamination path between e_j and $x \in V \cup E$ with $\nu(x) = \mathcal{C}$ exists, then $\nu'_{n+j} = \mathcal{R}$*
4. *if there exists a recontamination path between $x \in V \cup E$ and $y \in V \cup E$ with $\nu(y) = \mathcal{C}$, then $\nu'(x) = \mathcal{C}$*
5. *$\nu'_i = \nu_i$ otherwise.*

In colloquial terms the above describes the following rules stated in [10]:

1. edges where a block action is applied become blocked;
2. vertices where a sweep action is applied become clear;
3. blocked edges where a block action is not applied anymore become clear if there is no recontamination path involving them;
4. vertices or edges for which a recontamination path towards a contaminated vertex or edge exists become contaminated;
5. vertices or edges keep their previous state if none of the former cases apply.

A strategy to clear an initially fully contaminated surveillance graph G is a sequence of actions $\mathcal{S} = \{a_1, a_2, \ldots, a_k\}$. A strategy that is a solution to the Graph-Clear problem has in addition a minimal cost, i.e. $ag(\mathcal{S}) = \max_{i=1\ldots k} c(a_i)$ is minimal. In [10] it has been shown that solving the Graph-Clear problem on graphs is NP-hard and a polynomial time algorithms for trees was presented. The algorithms has been applied to robotic search in [6, 8–11]. In [6] a method to extract instances of the Graph-Clear problem from robot maps was presented, validating the graph-based model for practical use.

The above definitions are best illustrated with the simple example shown in Fig. 1. In this example vertices are associated with rooms, and edges with connections between rooms. Edges between vertices are blocked by placing a robot in the connection between rooms. All contaminated parts can hide an intruders while cleared parts are guaranteed to be free of undetected intruders. A room is cleared by using the specified number of robots to sweep through it.

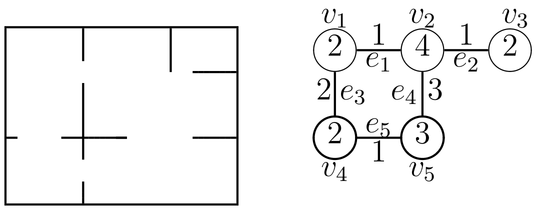

Fig. 1. A simple example environment and a possible surveillance graph that can model the search for a target. Numbers on vertices are clearing costs and numbers on edges are blocking costs.

Much of the prior work in graph-searching by [3] focused on actions that are executed by a single searcher, while the Graph-Clear model explicitly considers actions that require multiple searchers. In this paper we consider connected searching (sometimes also named as contiguous search) to be a search in which the cleared edges and vertices always form a connected subgraph. In Graph-Clear, it is obligatory to block every connected edge when sweeping a node.

Fig. 2 presents a surveillance strategy to solve the Graph-Clear problem associated with the graph shown in Fig. 1. The first column displays the status of the graph in the form of "$\nu(v_1)\cdots\nu(v_5)\ \nu(e_1)\cdots\nu(e_5)$", the second the applied action of the form "$a_{v_1}\cdots a_{v_5}\ a_{e_1}\cdots a_{e_5}$", and the third the cost. In the third row an action sweeps two vertices at the same time. A final action removing all blocks is executed in the end (with 0 cost). The cost of this strategy is 12, i.e., the maximum value read in the third column, and as such not optimal.

$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC}\ \mathcal{CCCCC}$	10000 10100	5
$\mathcal{RCCCC}\ \mathcal{BCBCC}$	00010 10101	6
$\mathcal{RCCRC}\ \mathcal{BCBCB}$	01100 11011	12
$\mathcal{RRRRC}\ \mathcal{BBRBB}$	00001 00011	7
$\mathcal{RRRRR}\ \mathcal{RRRBB}$	00000 00000	0
$\mathcal{RRRRR}\ \mathcal{RRRRR}$		

Fig. 2. A Graph-Clear strategy.

3 Strategy with global minimal execution time

The previous section introduces an optimal clearing strategy that has minimal cost. When such a strategy is implemented in practice, we have to decompose it into execution paths for each individual robot. In general, there are many ways to decompose a strategy. In this paper, we are interested in the executions that have the shortest execution time. For example, Fig. 4 shows an optimal strategy for the model in Fig. 3. Fig. 6 illustrates an execution of the strategy

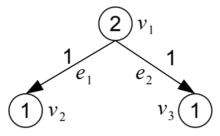

Fig. 3. An example

$\nu(G)$	a	$c(a)$
$\mathcal{CCC}\ \mathcal{CC}$	100 11	4
$\mathcal{RCC}\ \mathcal{BB}$	010 11	3
$\mathcal{RRC}\ \mathcal{BB}$	001 01	2
$\mathcal{RRR}\ \mathcal{RR}$		

Fig. 4. A clearing strategy

$\nu(G)$	a	$c(a)$
$\mathcal{CCC}\ \mathcal{CC}$	100 11	4
$\mathcal{RCC}\ \mathcal{BB}$	011 11	4
$\mathcal{RRR}\ \mathcal{RR}$		

Fig. 5. A shortest strategy.

in Fig. 4 with 4 robots. In this figure, two robots are marked with solid discs and the other two are marked by circles. Clearly, this graph does not represent the shortest solution because step 2 can be skipped by asking the two robots marked with solid discs in v_1 in Fig. 6(b) to move with the other two at the same time. This would make the system evolve from step 1 in Fig. 6(b) to step 3 directly in Fig. 6(d). Indeed, by skipping step 2, we change this strategy into

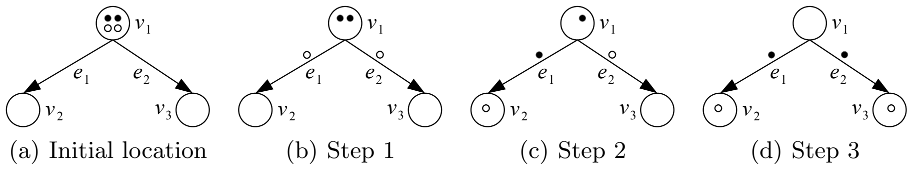

Fig. 6. An execution of a clearing strategy

a new strategy demonstrated in Fig. 5.

In this section, we propose a linear programming system to compute a clearing strategy that has the minimal cost and its execution plan for each robot with minimal execution time. We first present the general constraints that can be applied in a broader context, and second the specific constraints for computing the shortest execution path. We assume that all robots move at the same speed, and therefore, execution time in one step can be measured by the maximum travel distance during the step. We also assume that the distance between a vertex and an adjacent edge is constant.

3.1 General constraints

As stated in the previous section, a graph with n vertices and m edges can be cleared in n steps, each of which clears one vertex. Suppose that at least k robots are needed to clear a graph[1]. In each step, a robot can be located in one of the $n+m$ places. Let $l = n+m$ be the number of possible locations. We also assume that initially the robot has been placed into a vertex or edge. Therefore, we need $l \cdot (n+1)$ binary LP variables $X_1, \ldots, X_{l\cdot(n+1)}$ to encode all possible locations of each robot in every step and the initial locations. We assume that the edges

[1] The minimal number of robots can be calculated using the algorithm in [10] or [13].

are numbered after vertices, i.e., from $n+1$ to $n+m$, and that the robots are numbered from 0 to $k-1$. The initial location of robot j $(0 \leq j \leq k-1)$ is constrained by the following equation:

$$X_{j\cdot l+1} + \cdots + X_{(j+1)\cdot l} = 1. \tag{1}$$

As each variable is binary, the above equation captures the fact that the robot locates exactly in one place.

The constraint for the location of robot j at the i-th step $(1 \leq i \leq n)$ is formulated as follows.

$$X_{i\cdot k\cdot l+j\cdot l+1} + \cdots + X_{i\cdot k\cdot l+(j+1)\cdot l} = 1. \tag{2}$$

The expression $i \cdot k \cdot l$ above indicates the total number of LP variables from step 0 to step $i-1$, where the initial location is seen as step 0. We call the variables appearing in Equations (1) and (2) *location variables*. As n steps are required to clear the graph, there are

$$\mathit{\Delta}_1 = (n+1)\cdot k \cdot l \tag{3}$$

location variables in the LP system for k robots.

The following constraint models the travel distance of a robot moving in a step. For a robot j and two locations p and q $(1 \leq p, q \leq l)$, we generate a binary LP variable $X_{f(p,q)}$ to represent the possibility that the robot moves from p to q at the i-th step $(1 \leq i \leq n)$.

$$2 \cdot X_{f(p,q)} - X_{(i-1)\cdot k\cdot l+j\cdot l+p} - X_{i\cdot k\cdot l+j\cdot l+q} \leq 0, \tag{4}$$

where

$$f(p,q) = \mathit{\Delta}_1 + (i-1)\cdot k \cdot l^2 + j \cdot l^2 + (p-1)\cdot l + q. \tag{5}$$

In this constraint, $X_{(i-1)\cdot k\cdot l+j\cdot l+p}$ encodes the location of the robot j at the $(i-1)$-th step and $X_{i\cdot k\cdot l+j\cdot l+q}$ its location at the i-th step. When both variables are set to one, which means that indeed the robot moves from p to q at the i-th step, $X_{f(p,q)}$, a binary *move variable*, can be set to one without violation of Equation (4). In other cases, i.e., at least one location variable is zero, $X_{f(p,q)}$ has to be set to zero, indicating that this is not an actual move. For each robot and each step, there are l^2 possible moves, and hence, we need l^2 move variables. The following constraint requires that exactly one of the l^2 move variables is set to one representing the actual move.

$$\sum_{1\leq p,q\leq l} X_{f(p,q)} = 1. \tag{6}$$

The move variables constitute the majority of total variables used in the LP system: there are

$$\mathit{\Delta}_2 = n \cdot k \cdot l^2 \tag{7}$$

move variables.

Now we introduce *distance variables* of integer type, one for each step. Let D_i be the distance variable for the i-th step $(1 \leq i \leq n)$. As each robot has one move variable set to one, the following constraint states that D_i is no less than the maximum distance a robot can move in one step:

$$\bigwedge_{0\leq j<k} \{(\sum_{1\leq p,q\leq l} d_{p,q} \cdot X_{f(p,q)}) - D_i \leq 0\}, \tag{8}$$

where $d_{p,q}$ is the minimum travel distance between p and q. Clearly, there are

$$\Delta_3 = n \tag{9}$$

distance variables. The objective function is to minimise the sum of the distance variables, i.e.,

$$\sum_{1 \leq i \leq n} D_i. \tag{10}$$

3.2 Constraints for Graph-Clear strategies

In this subsection, we model the constraints required by Graph-Clear strategies. The first constraint is to block an edge. Let c_{e_r} be the cost of edge e_r ($1 \leq r \leq m$). At the i-th step, edge e_r has a binary *edge blocking* variable $Y_{i \cdot m+r}$ to represent whether it is being blocked at this step. Equation (11) enforces that the edge blocking variable cannot be set to one if the number of robots staying in the edge is fewer than the cost of blocking the edge.

$$c_{e_r} \cdot Y_{i \cdot m+r} - \sum_{0 \leq j < k} X_{i \cdot k \cdot l + j \cdot l + n + r} \leq 0. \tag{11}$$

There are in total

$$\Delta_4 = n \cdot m \tag{12}$$

of edge blocking variables. Equation (13) guarantees that the edge blocking variable is set to one as far as the number of robots in the edge reaches the required number for blocking the edge.

$$\sum_{j=0}^{k-1} X_{i \cdot k \cdot l + j \cdot l + n + r} + (c_{e_r} - 1 - k) \cdot Y_{i \cdot m+r} \leq c_{e_r} - 1. \tag{13}$$

The correctness of Equation (13) can be proved by contradiction. Suppose the number of robots in the edge is large enough to block the edge and the edge blocking variable is set to zero. Then, Equation (13) is transformed into

$$\sum_{j=0}^{k-1} X_{i \cdot k \cdot l + j \cdot l + n + r} \leq c_{e_r} - 1,$$

which is clearly unsatisfiable. Therefore, $Y_{i \cdot m+r}$ has to be set to 1 and Equation (13) becomes

$$\sum_{j=0}^{k-1} X_{i \cdot k \cdot l + j \cdot l + n + r} \leq k,$$

which is tautology.

Clearing a vertex at the i-th step can be modelled in a similar way. Let c_{v_r} be the cost of clearing vertex v_r ($1 \leq r \leq n$), and $Z_{i \cdot n+r}$ the binary *vertex clearing* variable. Equations (14) and (15) formalise the constraint. In addition, all adjacent edges to v_r have to be blocked at the same time. For each adjacent edge e_s, Equation (16) requests e_s to be blocked.

$$c_{v_r} \cdot Z_{i \cdot n+r} - \sum_{0 \leq j < k} X_{i \cdot k \cdot l + j \cdot l + r} \leq 0. \tag{14}$$

$$\sum_{j=0}^{k-1} X_{i\cdot k\cdot l+j\cdot l+r} + (c_{v_r} - 1 - k)\cdot Z_{i\cdot n+r} \leq c_{v_r} - 1. \tag{15}$$

$$Z_{i\cdot n+r} - Y_{i\cdot m+s} \leq 0. \tag{16}$$

There are in total

$$\Delta_5 = n^2 \tag{17}$$

of vertex clearing variables.

A Graph-Clear strategy clears one vertex at each step, which can be expressed as follows.

$$\sum_{r=1}^{n} Z_{i\cdot n+r} \geq 1. \tag{18}$$

The above equation states that at the i-th step, there exists a node r $(1 \leq r \leq n)$ that is being cleared. When a Graph-Clear strategy is executed completely, we would expect that all vertices have been cleared, which is expressed by the following equation on all vertices: for each node r, there exists a step i $(1 \leq i \leq n)$ at which r is cleared.

$$\sum_{i=1}^{n} Z_{i\cdot n+r} \geq 1. \tag{19}$$

The strategy also requires that the node being cleared at the i-th step $(i > 1)$ should be adjacent to a node that has been cleared at a previous step. Let $V_r = \{r_1, \cdots, r_t\}$ be the set of indices of adjacent nodes for node v_r. We have

$$Z_{i\cdot n+r} - \sum_{j=1}^{i-1} \sum_{p\in V_r} Z_{j\cdot n+p} \leq 0. \tag{20}$$

The correctness of Equation (20) is obvious: $Z_{i\cdot n+r}$ cannot be set to 1 if none of $Z_{j\cdot n+p}$ variables is 1, representing no adjacent nodes in V_r has been cleared before. Similarly, an edge e_r cannot be blocked at the i-th step until one of its adjacent vertices has been swept before or being swept at the same step. Let $E_r = \{r_1, r_2\}$ be the adjacent vertices of e_r. We have

$$Y_{i\cdot m+r} - \sum_{j=1}^{i} \sum_{p\in E_r} Z_{j\cdot n+p} \leq 0. \tag{21}$$

To prevent recontamination, an edge e_r that is blocked for sweeping a vertex has to be continuously blocked until both adjacent vertices have been swept. Let v_p be an adjacent vertex of e_r. This constraint is characterised by the equation:

$$Y_{(i-1)\cdot m+r} - \sum_{j=1}^{i} Z_{j\cdot n+p} - Y_{i\cdot m+r} \leq 0. \tag{22}$$

The correctness of Equation (22) is straightforward: at the i-th step, $Y_{i\cdot m+r}$ has to be set to 1, meaning that e_r is being blocked at this step, if it is blocked at the previous step ($Y_{(i-1)\cdot m+r} = 1$) and vertex v_p has not been swept yet (all Z variables are zero). If one of the Z variable is 1, then there is no constraint on $Y_{i\cdot m+r}$ enforced by v_p.

Complexity The total number of LP variables is

$$\Delta = \Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 + \Delta_5, \tag{23}$$

where Δ_2 is in the dominant position, which means that the number of LP variables is proportional to $\mathcal{O}(k \times n \times (n+m)^2)$. This suggests that the LP program for a large surveillance graph can easily be beyond the capacity of the state-of-the-art LP solvers.

4 Optimisation heuristics

Equation (1) allows the robots to be scattered all over the graph when they start to clear the graph. A reasonable assumption in practice is to place them in a vertex initially. Therefore Equation (24) can be modified to exclude edges.

$$X_{j \cdot l+1} + \cdots + X_{j \cdot l+n} = 1. \tag{24}$$

Similarly, we can assume all robots initially gather in an edge if requested.

After taking Equation (24), we could make another assumption that all robots start from the same location. This can be modelled by the following constraints: for all vertex $1 \leq i \leq n$ and all robots $0 < j < k$, we have

$$X_i - X_{j \cdot l+i} = 0. \tag{25}$$

The equations in this category set all variables corresponding to the same initial location to the same value by letting all robots from 1 to $k-1$ stay in the same location of robot 0.

To speed up the search for an optimal clear strategy with shortest execution time, it is essential to get a tight upper-bound for each distance variables. If all robots are initially positioned in the same vertex, then it is reasonable to assume they would sweep the initial vertex first. Therefore, the shortest time to execute the first step is the same as the time for some robots moving to the adjacent edges to block. Constraints specified in Equations (6) and (8) can be simplified by removing impossible moves between vertices and edges. For example, suppose that robots initially gather at vertex v_3 in Fig. 1. To sweep v_3, there is no need to allow robots to move to vertex v_2 and beyond. Hence, the move variables for those unrealistic moves can be removed from Equations (6) and (8).

For other steps, it is also very helpful if we can estimate the maximum distance that a robot needs to move. For the example in Fig. 1, we can get an optimal execution even if we do not allow a robot to move across three edges if it starts from an edge, or across three vertices if it starts from a vertex. For example, a robot cannot move from e_2 to e_3 or from v_2 to v_4 in one step.

5 Executing a predefined strategy with minimal execution time

The LP system defined in Section 3 is still very hard to solve even if we apply the heuristics in Section 4. In this section, we apply the LP system to find a shortest execution plan for a given clearing strategy. The strategy specifies the vertex to sweep and edges to block at each step, and thus, greatly reduces the search space. To achieve this, the constraints listed in Section 3.2 are not needed any more. Instead, we add the following constraints.

As in the previous section, we assume that initially the robots stay in the vertex that is swept at the first step. For each edge e_r being blocked at the i-th step, we generate the following inequality.

$$\sum_{j=1}^{n} X_{i\cdot n\cdot l+j\cdot l+n+r} \geq c_{e_r}. \tag{26}$$

Similarly, for vertex v_r being swept at the i-th step, we have

$$\sum_{j=1}^{n} X_{i\cdot n\cdot l+j\cdot l+r} \geq c_{v_r}. \tag{27}$$

The intuition in these equations is that the number of robots in e_r (v_r) should exceed the cost of blocking e_r (sweeping v_r).

Note that the LP system does not impose any constraints on edges that are in the clear, i.e., $\mathcal{R}$, state. A clear edge may still be blocked during the execution when necessary for saving execution time.

Complexity The extra constraints in Equations (26) and (27) do not introduce new LP variables. Thus, the total number of LP variables for computing the execution of a predefined strategy is

$$\Delta = \Delta_1 + \Delta_2 + \Delta_3. \tag{28}$$

This suggests that the complexity for finding an optimal execution plan for a given strategy is in the same level of that for computing a global optimal execution plan. In practice, however, this kind of LP programs can be solved much faster than that in Section 3 due to fewer LP variables and more constraints.

6 Experiment

The LP system described in this paper can be automatically generated by providing the adjacency matrix of the graph, and the distance matrix which records the minimal travel time/distance between any two locations in the graph. The number of robots, i.e., the cost of clearing a graph, also needs to be specified. In [13], we proposed to apply model checking techniques to compute the minimal cost and generating a corresponding strategy. Fig. 7 shows such a strategy.

Our prototype implementation takes this example strategy and generates the LP system described in Section 5, which is then solved by the LP solver Gurobi Optimizer [4]. We assume that the travel distance between a vertex and an adjacent edge is one and takes one unit of time to move. In a computer with dual Intel Xeon E5-2643 v2 processors and 384GB RAM, it took 112 seconds to find an optimal execution with the execution time 9 units of time, which is the minimal execution time for this strategy. The detail of the execution is illustrated in Fig. 7, where we list the location of each robot at every step. Note that the location at step 0 is the initial location. The last column gives the number of time units needed for every step.

$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC}$ $\mathcal{CCCCC}$	00001 00011	7
$\mathcal{CCCCR}$ $\mathcal{CCCBB}$	00010 00111	8
$\mathcal{CCCRR}$ $\mathcal{CCBBB}$	10000 10110	8
$\mathcal{RCCRR}$ $\mathcal{BCBBR}$	01000 11010	9
$\mathcal{RRCRR}$ $\mathcal{BBRBR}$	00100 01000	3
$\mathcal{RRRRR}$ $\mathcal{RBRRR}$		

Fig. 7. A Graph-Clear Strategy.

Step	Robot									Time
	1	2	3	4	5	6	7	8	9	
0	v_5	v_5	v_5	v_5	v_5	v_5	v_5	v_5	v_5	
1	v_5	e_5	v_5	e_5	e_4	e_4	e_4	v_5	e_5	1
2	v_4	e_3	e_4	e_3	e_4	e_1	e_4	e_5	v_4	2
3	e_3	v_1	e_4	v_1	e_4	e_1	e_4	v_5	e_3	1
4	v_2	v_2	e_4	e_1	e_4	e_2	v_2	e_4	v_2	3
5	e_2	v_2	e_4	e_1	e_4	v_3	e_2	e_4	v_3	2

Fig. 8. An execution

However, if we use the LP system defined in Section 3 and the heuristics in Section 4, then the solver managed to find an execution plan with 8 time units. The locations at each step is listed in Fig. 9 and the corresponding strategy is shown in Fig. 10. We can see that vertices v_1 and v_4 can be cleared in one step, which makes the fifth step redundant. However, the time for solving the LP system was increased to 60143 seconds.

Step	Robot									Time
	1	2	3	4	5	6	7	8	9	
0	v_3	v_3	v_3	v_3	v_3	v_3	v_3	v_3	v_3	
1	e_2	v_3	e_2	v_3	e_2	e_2	e_2	e_2	e_2	1
2	e_4	e_2	e_1	v_2	v_2	e_4	e_4	v_2	v_2	2
3	e_5	e_4	e_4	v_5	e_4	v_5	v_5	e_1	v_5	2
4	v_4	v_4	v_1	e_1	v_1	v_5	e_5	e_3	e_3	3

Fig. 9. An optimal execution.

$\nu(G)$	a	$c(a)$
$\mathcal{CCCCC}$ $\mathcal{CCCCC}$	00100 01000	3
$\mathcal{CCRCC}$ $\mathcal{CBCCC}$	01000 11010	9
$\mathcal{CRRCC}$ $\mathcal{BBCBC}$	00001 10011	8
$\mathcal{CRRCR}$ $\mathcal{BRCBB}$	10010 10101	8
$\mathcal{RRRRR}$ $\mathcal{BRBRB}$		

Fig. 10. A Graph-Clear Strategy.

Discussion. During at any point of the process of solving the LP system, Gurobi Optimizer maintains an upper bound and a lower bound (also called *best bound*) for the value of the objective function. The upper bound is the value obtained by the temporary best solution, called *incumbent solution*, up to that point. The solving process terminates when the upper bound matches the lower bound. However, this process can be interrupted at any point and the incumbent solution is returned. The incumbent solution may not be an optimal solution, but in this example, the incumbent solution found at 110 seconds represents an execution plan with 8 time units, which is the upper bound. It means that this solution is an optimal solution, although the best bound at that point is lower than the upper bound. Therefore, the solving process can be stopped at any point after the upper bound reaches 8. The solving process for a given strategy can be stopped in the same way. When the global minimal value of the objective function is not known, we can use the following two steps to acquire an acceptable solution. First, we compute a strategy with a minimal number of pursuers and obtain the minimal value for the objective function of this strategy. Second, we run the LP solver to solve the LP system for the global optimal solution and terminate the solving process at any point when a better strategy than the known strategy is found.

7 Conclusion

This paper solves the optimal design of movements by robots in an enviroment modelled by a surveillance graph under the constraints of robot movements and resources. A method to find a shortest execution plan for a given clear strategy is also presented. It is assumed that the moving distance between a vertex and an adjacent edge is constant. As this assumption may not hold in practice when a vertex represents a fairly large area, further work can be carried out to improve the modelling approach taken in this paper.

References

1. 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.
2. T.H. Chung, G.A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.
3. F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.
4. Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2014.
5. A. Kolling and S. Carpin. The graph-clear problem: definition, theoretical properties and its connections to multirobot aided surveillance. *Proc. of IEEE/RSJ Intl. Conf. On Intelligent Robots and Systems*, pages 1003–1008, 2007.
6. A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *Proceedings of IROS'08*, pages 2323–2328, 2008.
7. A. Kolling and S. Carpin. Multi-robot surveillance: an improved algorithm for the graph-clear problem. *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2360–2365, 2008.
8. A. Kolling and S. Carpin. Surveillance strategies for target detection with sweep lines. In *Proceedings of IROS'09*, pages 5821–5827, 2009.
9. A. Kolling and S. Carpin. Multi-robot pursuit-evasion without maps. In *Proceedings of ICRA'10*, pages 3045–3051, 2010.
10. A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE T. Robot.*, 26(1):32–47, 2010.
11. A. Kolling and A. Kleiner. Multi-uav motion planning for guaranteed search. In *Proceedings of AAMAS'13*, pages 79–86, 2013.
12. T.D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Applications of Graphs*, volume 642, pages 426–441. Springer, 1976.
13. H. Qu, A. Kolling, and S. M. Veres. Formulating robot pursuit-evasion strategies by model checking. In *Proceedings of IFAC'14*, pages 3048–3055, 2014.
14. J. Thunberg and P. Ögren. A mixed integer linear programming approach to pursuit evasion problems with optional connectivity constraints. *Auton. Robots*, 31(4):333–343, 2011.