Lions and Contamination: Monotone Clearings

Daniel Bertschinger
Department of Computer Science, ETH Zürich, Switzerland

Meghana M. Reddy[1]
Department of Computer Science, ETH Zürich, Switzerland

Enrico Mann
Department of Computer Science, ETH Zürich, Switzerland

Abstract

We consider a special variant of a pursuit-evasion game called lions and contamination. In a graph whose vertices are originally contaminated, a set of lions walk around the graph and clear the contamination from every vertex they visit. The contamination, however, simultaneously spreads to any adjacent vertex not occupied by a lion. We study the relationship between different types of clearings of graphs, such as clearings which do not allow recontamination, clearings where at most one lion moves at each time step and clearings where lions are forbidden to be stacked on the same vertex. We answer several questions raised by Adams et al. [2].

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Algorithmic Games, Pursuit-Evasion Games, Graph Contamination, Clearings

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.17

Funding *Meghana M. Reddy*: Supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

1 Introduction

Pursuit-evasion problems have a long and rich history going back more than 50 years [11, 14]. Countless similar problems have been studied under very different names in the past. What they all have in common is that there is a group of pursuers that try to catch an evader. The typical question asked in a pursuit-evasion problem is whether the evader can escape the pursuers, and if so, for how long. Naturally, the more pursuers there are, the harder it is for the evader to escape. Some objectives of the pursuers can be to catch the evader fast (minimize the time taken) or with minimal effort (minimize the distance traveled); the different objectives all have their origins in numerous applications such as robot motion planning, collision avoidance, and intruder detection in networks [1, 8, 13].

There are different variations of the problem depending on the exact rules. For detailed definitions of the various problems, see the surveys [4, 5, 7, 10] and the references therein. In this paper, we study the problem of lions and contamination [9]. Traditionally, a group of lions tries to eradicate contamination from a graph while the contamination spreads to all adjacent vertices that are not occupied by lions. A scenario that is relevant to current day is where a set of doctors tries to get rid of a disease; people can get tested and quarantined to stop the spread of the disease, whereas the contamination spreads to people that had contact with infected people.

[1] The second author's full last name consists of two words and is *Mallik Reddy*. However, she consistently refers to herself with the first word of her last name being abbreviated.

Formally, suppose there is a graph $G = (V, E)$. At the very beginning, every vertex occupied by a lion is considered cleared of contamination, whereas the remaining vertices are considered to be contaminated. Time is viewed as discrete; and in each step, the lions and the contamination both move simultaneously. Every lion is allowed to move along an edge that is incident to its current position. Contamination, on the other hand, spreads along every incident edge to every adjacent vertex, unless the edge is used by a lion or a lion occupies the adjacent vertex. Figure 1 illustrates an example. Note that the sequential variant of this problem, where the lions and contamination move one after the other in alternating time steps, results in a setting where the lions are more powerful and hence is a very different problem compared to the one we study.

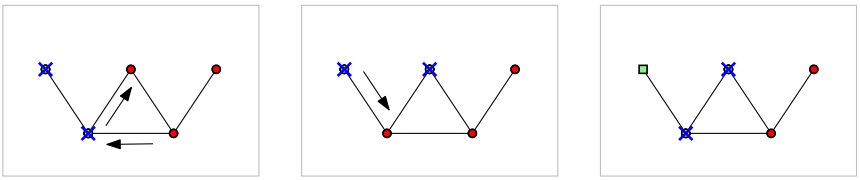

Figure 1 A graph and two lions (indicated by blue crosses), every vertex that is not occupied by a lion is contaminated initially (indicated by red disks). One lion moves in the first time step, however, contamination moves simultaneously and the vertex gets recontaminated. After the second step, a vertex that is not occupied by a lion is cleared of contamination (indicated by green squares).

Note that for some graphs, it is easy to see whether k lions can clear the graph of contamination. However, finding the minimum number of lions required to clear a graph is a hard question. For example, the minimum number of lions required to clear the $n \times n$-grid is not known. Nevertheless, it is known that at least $\lfloor \frac{n}{2} \rfloor + 1$ lions are needed [6]. Since n lions can simply sweep the graph from left to right and clear the grid of contamination, n is an upper bound on the number of lions needed for clearing the $n \times n$-grid, and this is the best upper bound currently known. Further, it is believed that $n - 1$ lions are not sufficient. For higher-dimensional grids, that is for the n^d-grid, it is known that $\Theta(n^{d-1}/\sqrt{d})$ lions are necessary and sufficient [3].

In this paper, we study different types of clearings that were defined by Adams et al. [2] and answer several questions raised in their paper. A clearing of a graph using k lions is denoted as a *k-clearing* and the graph itself is referred to as *k-clearable*. We say a clearing is *monotone* if no vertex ever gets recontaminated. The lions and the clearing are said to be *polite* if at most one lion moves in each time step and *non-stacked* if no two lions occupy the same vertex at any point in time.

The remainder of this paper is organized in the following way. In Section 2, we show that there exist k-clearable graphs which require more than k lions for any monotone clearing. This implies that monotone clearings are harder to achieve than non-monotone clearings and stands in contrast to a result in the *graph searching* setting [12], where monotonicity does not matter. In Section 3, we show that any monotone clearing can be paused at any time and no recontamination occurs. This allows us to show in an algorithmic way that any monotone clearing can be converted into a monotone and polite clearing. We then show that polite clearings can be transformed into non-stacked clearings (see Theorem 3.6). Finally, in Section 4, we tackle the subgraph question raised in [2]: given a k-clearable graph G and some subgraph $H \subseteq G$, is H k-clearable as well? We answer this question in some settings.

2 Monotone Clearings

2.1 The $n \times n$ Grid

Let us consider the $n \times n$ grid. As already mentioned, at least $\lfloor \frac{n}{2} \rfloor + 1$ lions are needed, while n lions are sufficient to clear the grid. When restricted to monotone clearings, we can improve the lower bound and close the gap between the bounds.

Let $V(t)$ be the set of cleared vertices at time t. We define a *boundary vertex* as a vertex of $V(t)$ that has a neighbor in $V \setminus V(t)$. Before we state and prove our result, we first make two simple observations and recall a lemma proved by Berger et al. [3].

▶ **Observation 2.1.** *The number of cleared vertices in a k-clearing cannot increase by more than k in one time step.*

▶ **Observation 2.2.** *If there are more than k boundary vertices at time t, then at least one vertex gets recontaminated in the next time step.*

▶ **Lemma 2.3** (Lemma 5 of [3])**.** *Any vertex set S that is a subset of the $n \times n$ grid and satisfies $\frac{n^2}{2} - \frac{n}{2} \leq |S| \leq \frac{n^2}{2} + \frac{n}{2}$ has at least n boundary vertices.*

▶ **Theorem 2.4.** *A monotone clearing of the $n \times n$ grid needs at least n lions.*

Proof. Assume that the $n \times n$ grid has a monotone clearing with $n-1$ lions. Initially, the lions start with at most $n - 1 < \frac{n^2}{2} + \frac{n}{2}$ cleared vertices and eventually have to clear all n^2 vertices. Further, Observation 2.1 implies that at most $n-1$ vertices are cleared in each time step. Thus, at some time t, the set of cleared vertices $V(t)$ must satisfy the condition $\frac{n^2}{2} - \frac{n}{2} \leq |V(t)| \leq \frac{n^2}{2} + \frac{n}{2}$. The number of boundary vertices at such a time t will be at least n due to Lemma 2.3, and Observation 2.2 then implies that at least one vertex will get recontaminated at time $t+1$. Hence, no monotone $(n-1)$-clearing of the $n \times n$ grid exists. ◀

In hindsight, this might not be a very surprising result. However, this does not necessarily improve the lower bound for non-monotone clearings as we will see in the next subsection.

2.2 Graphs with no Monotone Clearing

Unfortunately, not every graph with a k-clearing admits a monotone k-clearing. Indeed, we can show that the set of monotone k-clearable graphs is a proper subset of the set of all k-clearable graphs, which implies that monotonicity is a strong assumption on clearings. Theorem 2.4 does not improve the general lower bound for grids due to this reason.

▶ **Theorem 2.5.** *For any $k \geq 2$ there exist k-clearable graphs with no monotone k-clearing.*

We first describe the construction of one set of such graphs for any $k \geq 2$. We start with three distinct $k \times k$-grids connected by $(k-1)$-grid-like paths from row 2 through row n. The center grid additionally has two vertices of degree one each attached to its leftmost and rightmost vertices of row 1 respectively, let these vertices be denoted by u_ℓ and u_r. For simplicity let us denote graphs constructed in this form as G_k, the graph G_3 is illustrated in Figure 2.

▶ **Lemma 2.6.** *The graph G_k is k-clearable for any $k \geq 2$, but admits no monotone k-clearing.*

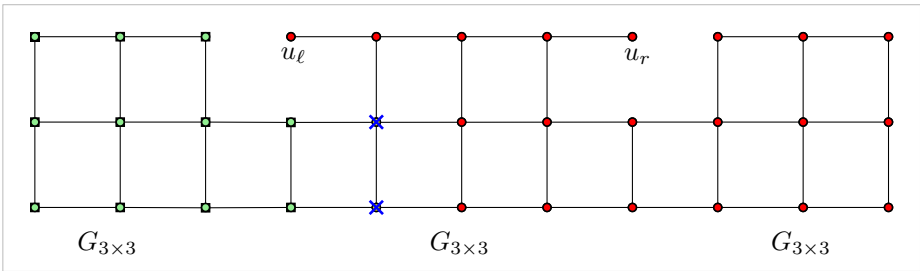

Figure 2 Illustration of G_k when $k = 3$. The center grid has two additional vertices of degree one attached at distinct corners of the first row. The colors indicate one possible clearing state.

Proof. To see that G_k is k-clearable we describe a clearing. Each lion is assigned to clear one row of the graph. The lions start on the leftmost vertices of the graph and start sweeping the graph from left to right. After the lions have cleared the left grid, the lions on rows $2, \ldots, n$ move to the vertices on the $(k-1)$-path between the left and center grids and wait for the lion from row 1 to move to u_ℓ. Observe that this can be achieved without stacking lions, by moving the lion of row 1 to the position of the lion of row 2, and moving the lion of row 2 to u_ℓ. All the lions then together sweep further until the center grid and u_r are cleared; and similarly wait on the $(k-1)$-path between the center and right grids before finally clearing the right grid.

It remains to show that this graph cannot be cleared in a monotone fashion by k lions. To clear the graph monotonically, the lions must all either start on the left or the right grids. All or any subset of the k lions starting in the center grid will always lead to recontamination of some vertex (in the center) since the left and right grids are contaminated. Without loss of generality, assume the lions start in the left grid. Irrespective of the strategy followed by the lions, the lions always end up in a situation where either there is a lion each on u_ℓ and its neighbor, or there are $k-1$ lions that stop the movement of contamination from the center grid to the left grid. In the first case, the remaining $k-2$ lions are not sufficient to stop contamination from recontaminating the left grid, and in the second, the neighbor of u_ℓ gets recontaminated when the last lion moves to u_ℓ to clear the vertex. Hence, we can conclude that no monotone clearing exists for G_k. ◀

We give another construction for graphs that have clearings with k lions but do not admit monotone k-clearings. It gives further insights into how such graphs can look like. Interestingly, the following construction has four cut-vertices; which should ideally make it easy to isolate contamination to one section of the graph.

We denote the second type of graph as H_k. The graph H_k consist of several parts. The main building block is a $(k+1)$-clique K_{k+1}, to which we add four *arms* at different vertices. Each of the arms consists of a long k-grid (length of $3k$ is enough), where the farthest vertices form a clique themselves; see Figure 3 for an example with $k = 5$. The graph H_k is $(k+1)$-clearable, but has no monotone $(k+1)$-clearing.

Theorem 2.5 shows that monotonicity is a very restrictive assumption on clearings. This result however raises the following question for further research.

▶ **Open Question 1.** *Given a k-clearable graph $G = (V, E)$, is it always monotonically clearable with $k+1$ lions? More formally, is there a non-trivial upper bound on the number of lions required for a monotone clearing?*

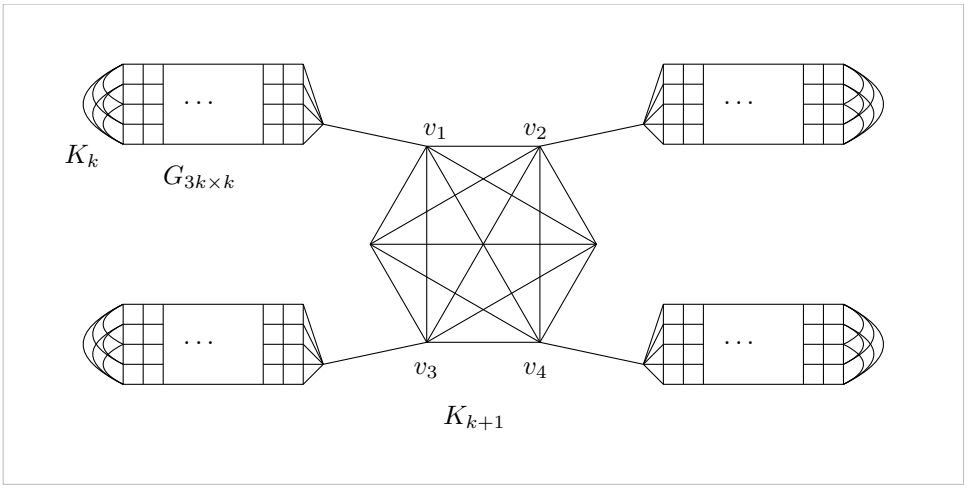

Figure 3 The graph H_k consisting of a $(k+1)$-clique in the middle and four attached arms, each consisting of a long k-grid with a k-clique at the far end.

3 Transforming between Different Types of Clearings

3.1 Monotone Clearings

We now analyse some properties of monotone clearings. These will then allow us to show that monotone clearings can always be adapted to monotone clearings with polite lions. We start with an important observation.

▶ **Proposition 3.1.** *Let $\mathcal{C}$ be any clearing of a graph G. Assume that all the lions are paused indefinitely at time t, that is, no lion moves after time t. If no vertex gets recontaminated at time $t+1$, then no vertex gets recontaminated at a later time either.*

Proof. Recall that contamination spreads along every incident edge at each time step. If no vertex gets recontaminated at time $t+1$, it implies that every cleared vertex neighboring a contaminated vertex is occupied by a lion that blocks the contamination from spreading. Then, no vertex can get recontaminated after time $t+1$ either, since the lions continue to block the contamination from spreading. ◀

With this result, we are now able to argue about stopping lions in monotone clearings.

▶ **Lemma 3.2.** *A monotone clearing can be paused at any time and no vertex gets recontaminated.*

Proof. Assume that every lion is paused at time t in a monotone clearing. For contradiction, assume v is one of the first vertices to get recontaminated. Then no lion occupies vertex v at time t, vertex v is not contaminated at time t and v gets recontaminated earliest at time $t+1$ due to Proposition 3.1.

Then there must exist $w \in N(v)$ that is contaminated at time t, which contaminates v at time $t+1$. Since the clearing is monotone until time t, vertex w must have never been cleared of contamination. Therefore, the contamination was present at w at $t-1$ and must have spread to v at time t (recall that no lion occupies v at time t). This contradicts the fact that no vertex gets recontaminated in a monotone clearing, and our clearing remains monotone until and including time t. ◀

With a more careful analysis of pausing monotone clearings, we can prove a much stronger result.

▶ **Theorem 3.3.** *Let G be a graph and $\mathcal{C}$ be a monotone clearing of G with k lions. Then, there exists another monotone clearing which uses k polite lions.*

Before being able to prove Theorem 3.3, we need some observations. Let us reconsider a monotone clearing, denoted by $\mathcal{C}$. Let us once again pause all the lions at time t. By Lemma 3.2 we know that no recontamination occurs. Consider a lion ℓ_1 and assume that this lion moved from v to w at time $t+1$ in the clearing $\mathcal{C}$. We define a new strategy for a clearing, denoted by $\mathcal{C}'$, where we move all lions according to $\mathcal{C}$ up to time t and at time $t+1$, we move lion ℓ_1 from v to w. No other lion moves at time $t+1$ or after.

It is now important to note that this strategy might not be a clearing and it can happen that vertices get recontaminated, see Figure 4. However, we certainly know which vertex gets recontaminated first. More specifically, we can prove vertex v is the only vertex that can get recontaminated at $t+1$.

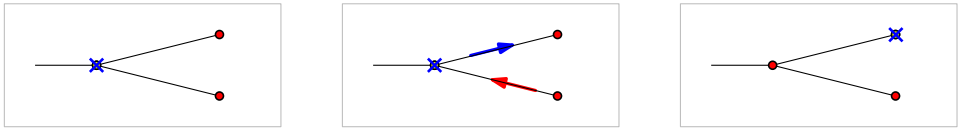

Figure 4 A lion moving from the left vertex to one of the vertices on the right. The left vertex gets recontaminated at the same time.

▶ **Lemma 3.4.** *Let $\mathcal{C}$ be a monotone clearing. Let $\mathcal{C}'$ be a new strategy that mirrors $\mathcal{C}$ until time t, and moves one lion from v to w at time $t+1$, and no other lion is moved from $t+1$ onwards. If recontamination occurs in $\mathcal{C}'$, the first vertex that gets recontaminated must be v and it gets recontaminated at time $t+1$. Further, no other vertex except v can get recontaminated at time $t+1$.*

Proof. We know that if $\mathcal{C}'$ is paused at t, no recontamination occurs due to Lemma 3.2. The only difference between $\mathcal{C}$ and $\mathcal{C}'$ at $t+1$ is that the lion ℓ_1 moved away from v in $\mathcal{C}'$. Thus, at time $t+1$, no vertex other than v can get recontaminated. If v is not recontaminated at time $t+1$, then by Proposition 3.1, we know that no vertex ever gets recontaminated. Hence, v is the only vertex that might get recontaminated at time $t+1$. ◀

Unfortunately, we may not be able to avoid this recontamination when moving lion ℓ_1 in $\mathcal{C}'$. Nonetheless, we use the fact that vertex v does not get recontaminated in $\mathcal{C}$ and analyse the situation closely. Let us assume that v gets recontaminated in $\mathcal{C}'$ at time $t+1$.

Let us take a closer look at this vertex v. Since v is recontaminated at time $t+1$ in $\mathcal{C}'$, there must exist $x \in N(v)$ that was contaminated at t in $\mathcal{C}'$, which spread the contamination to v at $t+1$. Observe that x must be contaminated at time t in the original clearing $\mathcal{C}$ as well and v is not recontaminated at time $t+1$ in $\mathcal{C}$. Therefore some lion must occupy v at time $t+1$ in $\mathcal{C}$ (otherwise v would also get recontaminated in the monotone clearing $\mathcal{C}$ at time $t+1$). Let this lion be ℓ_j. Note that $\ell_j \neq \ell_1$ as ℓ_1 moved away from v at time $t+1$ in $\mathcal{C}$. If we move ℓ_j before moving ℓ_1, then the recontamination of vertex v can be avoided at time $t+1$. Hence, we say that ℓ_1 *depends on* ℓ_2 *at time* $t+1$.

We construct a graph on the set of lions, where a directed edge from ℓ_j to ℓ_i is added if and only if lion ℓ_j moves to vertex u at time $t+1$ in $\mathcal{C}$, and the lion ℓ_i moves away from u at time $t+1$ in $\mathcal{C}$. In this way, we capture all dependencies of the lions and we refer to this graph as the *dependency graph* of lions at time $t+1$. Note that the dependency graph can contain cycles.

▶ **Observation 3.5.** *By renaming lions, we can avoid any cycle in the dependency graph.*

To see this, we consider one cycle in the dependency graph. Note that this cycle corresponds to a set of vertices in the underlying graph that are occupied by a set of lions; and occupied by the same set of lions at time $t+1$ in $\mathcal{C}$. Each lion moved to a different vertex in the same set. Thus, the state of the corresponding vertices in the underlying graph G is the same at time t and at time $t+1$; and in particular this does not change if the lions are not moved but are only renamed. Once the lions belonging to one cycle are renamed, the dependency graph clearly changes. In particular, the number of edges (and cycles) strictly decreases by renaming the lions and hence we eventually get an acyclic dependency graph.

With this we can now prove Theorem 3.3.

Proof of Theorem 3.3. We prove the theorem by showing how the number of time steps that use polite lions can be iteratively increased by modifying a monotone clearing. Given a monotone k-clearing $\mathcal{C}$ that is polite up to some time t, we define a new clearing $\mathcal{C}'$ that is identical to $\mathcal{C}$ up to time t and is polite up to some time $t' > t$. In particular, let $\mathcal{L}_{t+1}$ be the set of lions that move at time $t+1$ in $\mathcal{C}$. Consider the dependency graph of $\mathcal{L}_{t+1}$ at time $t+1$ obtained after all the cycles have been removed (and thus some lions might have been renamed). Since this graph is acyclic, a topological ordering τ of the lions can be obtained such that any lion ℓ_i that depends on ℓ_j only comes after ℓ_j in τ. After time t, the new clearing $\mathcal{C}'$ moves lions one by one according to the order τ up to time $t' \leq t+k$ (the lions not in τ do not move). Note that the state of $\mathcal{C}'$ at time t' is identical to the state of $\mathcal{C}$ at time $t+1$. Finally, $\mathcal{C}'$ follows the same strategy as $\mathcal{C}$ did from time $t+2$ until the graph is completely cleared.

It is easy to see that $\mathcal{C}'$ is monotone up to time t and since there is no recontamination between time t and $t+k$, we indeed have a clearing. Furthermore, $\mathcal{C}'$ is polite up to time $t' > t$, while $\mathcal{C}$ was polite only until time t. Following this procedure iteratively, we are guaranteed to eventually get a monotone k-clearing that uses only polite lions from any monotone k-clearing. ◀

Note that this proof is algorithmic, in particular, given a monotone clearing, we can compute another monotone clearing that uses polite lions without increasing the number of lions.

3.2 Polite and Non-Stacked Clearings

In this subsection, we study clearings which need not be monotone and consider other restrictions on clearings. In particular, we study the relationship between clearings that use polite lions and clearings that do not stack lions. We can show the following relation.

▶ **Theorem 3.6.** *Let G be a graph on n vertices and $\mathcal{C}$ be a polite clearing of G with $k \leq n$ lions. Then, there exists another k-clearing that does not stack lions.*

Proof. For each time step in $\mathcal{C}$, we describe how to move the corresponding lion (and probably some additional lions) in $\mathcal{C}'$ by avoiding stacking while ensuring that the new strategy developed is a valid clearing of G. Let $V_{\mathcal{C}}(t)$ and $V_{\mathcal{C}'}(t)$ denote the set of cleared vertices at time t in $\mathcal{C}$ and $\mathcal{C}'$ respectively. We show that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t)$ at any time t.

Every lion in $\mathcal{C}$ begins at its *starting vertex*, follows a walk in the graph, and finally ends at its *end vertex*, and remains there for the rest of the clearing. In the process of adapting $\mathcal{C}$ to $\mathcal{C}'$, some lions are labelled as *retired*. These are lions that are stacked on their end vertices in $\mathcal{C}$, and thus their position in $\mathcal{C}'$ is irrelevant since they do not help in clearing any more vertices.

Before we describe the overall strategy (or algorithm), we describe the procedure of finding the next location for a lion ℓ_i in $\mathcal{C}'$. Assume ℓ_i has to be placed at v at time $t \geq 0$ because ℓ_i starts at v or moves to v from a neighboring vertex. Note that since $\mathcal{C}$ is a polite clearing, this is the only lion that needs to be moved in this time step. If there is no lion at v, ℓ_i is placed on v (or moved to v). For the other case, let us assume that there is another lion ℓ_j on v at time t. If ℓ_j is a retired lion, then ℓ_i is placed at v, and ℓ_j is moved to the closest vertex with no lion. We do not really care where ℓ_j is placed since the position of a retired lion is irrelevant in the rest of the clearing. If ℓ_j is not retired, the situation is a bit more delicate. If ℓ_i leaves v before ℓ_j in $\mathcal{C}$, then we instead place ℓ_j at the next vertex in the path of ℓ_i in $\mathcal{C}$ (which must be a neighbor of v). Then we place ℓ_i at v and switch the naming of the lions. This ensures that we do not stack the lions in this time step, while keeping them on their path. In the other case, namely if ℓ_j leaves v before ℓ_i in $\mathcal{C}$, we place ℓ_i at v, and place ℓ_j at the next vertex in its path in $\mathcal{C}$. Finally, in the case when v is the end vertex of both ℓ_i and ℓ_j, we label the lion ℓ_i as retired and place ℓ_i on the closest vertex which has no lion. It could happen that all the neighboring vertices of v are occupied by lions, or the vertex that we intend to move ℓ_i or ℓ_j to is occupied by a lion, in which case we have to follow this procedure recursively and move multiple lions. Note that even though multiple lions might have to be moved through this procedure to simulate one time step of the polite clearing, every step will eventually terminate as we reach retired lions (since $k \leq n$).

To find the starting positions of the lions in $\mathcal{C}'$ we use the procedure just described. The overall strategy follows the procedure as well. We consider one lion movement at a time (since $\mathcal{C}$ is polite) and move lions according to the description above. As already mentioned, this might move more than one lion for an individual movement in $\mathcal{C}$, but most importantly, this ends for sure. As each movement of a single lion individually terminates, the recursive procedure eventually terminates as well.

It remains to show that $\mathcal{C}'$ is indeed a clearing. For this, observe that whenever the last lion ℓ leaves a vertex u in $\mathcal{C}$ at time t, it also holds that ℓ leaves u in $\mathcal{C}'$ at the same time t. This is because the last lion to leave u is the lion that is positioned at u in $\mathcal{C}'$ by construction. Thus, every vertex that has a lion in $\mathcal{C}$ at some time t also has a lion in $\mathcal{C}'$ at t. Thus, it follows that at every time step t, $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t)$ and hence, $\mathcal{C}'$ is a clearing. ◀

Note that this result implies that the set of graphs that are clearable with polite lions is a subset of the set of graphs that admit non-stacked clearings. Though we were unable to prove it, we believe that the converse of Theorem 3.6 is false, because the time taken by polite lions might be much higher when compared to clearings where multiple lions can be moved simultaneously, even when stacking is not allowed.

▶ **Open Question 2.** *Is the converse of Theorem 3.6 also true or are there graphs with a non-stacked clearing but no polite clearing?*

When the clearing $\mathcal{C}$ is monotone, the converse is indeed true (follows from Theorem 3.3).

4 Clearable Subgraphs

In the final section of this paper, we study clearings of subgraphs of a k-clearable graph. More formally, let $G = (V, E)$ be a k-clearable graph, and let H be a subgraph of G. It is natural to ask if H also admits a k-clearing (see Question 6.1 in [2]). On one hand, the contamination is restricted due to some missing edges; on the other hand, some edges or paths in $G \setminus H$ might be crucial for the lions to clear the graph.

We show that this question can be answered in the affirmative if the clearing on G is monotone, and that surprisingly, this need not be possible in some other restricted settings.

▶ **Theorem 4.1.** *Let G be a graph with a monotone k-clearing. Then any connected subgraph $H \subset G$ also admits a (polite) k-clearing.*

Proof. Consider a monotone clearing $\mathcal{C}$ of G using k polite lions (which exists by Theorem 3.3). The idea is to use the same clearing in H with some modifications. In $\mathcal{C}$, if all lions use edges (and vertices) that are present in H, then the clearing $\mathcal{C}$ restricted to H is a clearing of H and the theorem is proved.

Otherwise, assume for now that $V_G = V_H$. Let t be the first time step such that a lion ℓ_i uses edge $e = (v, w)$ to move from v to w in $\mathcal{C}$, where $e \in G \setminus H$. Since H is connected, there exists a path $p_{v,w}$ from v to w. We construct a new strategy $\mathcal{C}'$ for H, which mirrors $\mathcal{C}$ until time $t-1$, then moves lion ℓ_i along the path $p_{v,w}$ instead of moving it along the edge e at time t, and no other lion moves until ℓ_i reaches w. Let $t' > t$ be the time when ℓ_i reaches w in $\mathcal{C}'$. Let $V_{\mathcal{C}}(t)$ and $V_{\mathcal{C}'}(t')$ denote the set of cleared vertices in $\mathcal{C}$ at time t and $\mathcal{C}'$ at time t' respectively. We claim that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t')$. Note that in $\mathcal{C}'$, v is the only vertex that might get recontaminated at time t due to Lemma 3.4. However, v gets recontaminated in $\mathcal{C}'$ at time t if and only if v gets recontaminated at time t in $\mathcal{C}$, and we know that v is not recontaminated in $\mathcal{C}$ since $\mathcal{C}$ is monotone. We further claim that no vertex $u \in V_{\mathcal{C}}(t)$ is recontaminated in $\mathcal{C}'$ after time t while ℓ_i is moving along the path $p_{v,w}$. Assume some vertex $u \in V_{\mathcal{C}}(t)$ is recontaminated in $\mathcal{C}'$ at time $t'' > t$. Since the only vertex that contains a lion in $\mathcal{C}$ at time t but contains no lion in $\mathcal{C}'$ is w, the contamination must have spread to u from w through a path $w, u_1, u_2, \ldots, u$ which contains no lion. However, the vertex u_1 must have been contaminated at time t in $\mathcal{C}$ as well since no lion is present on u_1 and $\mathcal{C}$ uses polite lions. The contamination would have then spread to u if the lions were paused at time t in $\mathcal{C}$, which is a contradiction to Lemma 3.2. This proves that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t')$.

Now, consider the case where $V_H \subsetneq V_G$. In $\mathcal{C}$, if a lion moves along an edge (v, w) in $\mathcal{C}$ where $v, w \in V_H$, but $(v, w) \notin E_H$, we follow the same strategy as discussed above. Now, let t be the first time when a lion ℓ_i uses edge $e = (v, w)$ to move from v to w in $\mathcal{C}$, where $v \in V_H$, but $w \notin V_H$. Let the path of ℓ_i in $\mathcal{C}$ be $\ldots, v, w, w_1, w_2, \ldots, w_k, \ldots$ and let w_i be the first vertex in this path after v such that $w_i \in V_H$. We proceed similar to the previous case, and construct a new strategy $\mathcal{C}'$ which mirrors $\mathcal{C}$ until time $t-1$, and then moves lion ℓ_i to w_i while keeping the other lions stationary. Let ℓ_i reach w_i at time t'. By the same argument as above, we can prove that $V_{\mathcal{C}}(t) \subseteq V_{\mathcal{C}'}(t')$. Observe that with this strategy, no lion ever ends up in a vertex of $G \setminus H$, except for the start vertex. However, if the start vertex of a lion lies in $G \setminus H$, we instead move it to the first vertex on its path lying in H.

Thus, we can follow the described strategy and adapt $\mathcal{C}$ to H to construct a new strategy $\mathcal{C}'$ that is a valid clearing of H. Furthermore, note that we only moved one lion at each time step and thus the resulting clearing of the subgraph is polite. ◀

Note that the strategy given in this proof may not be monotone since the vertices that were cleared on a detour might get recontaminated later on. In general, clearings of a subgraph need not be monotone. Figure 5 illustrates a graph G^* that admits a monotone 2-clearing (sweeping from left to right). However, recall graph G_2 described in Section 2.2 (for reference, graph G_3 is illustrated in Figure 2), which is a subgraph of G^* that has no monotone 2-clearing. Note that this recontamination cannot be avoided with the strategy given, not even for induced subgraphs.

Finally, we illustrate a graph where one of its subgraphs does not admit a 2-clearing with polite and non-stacked lions.

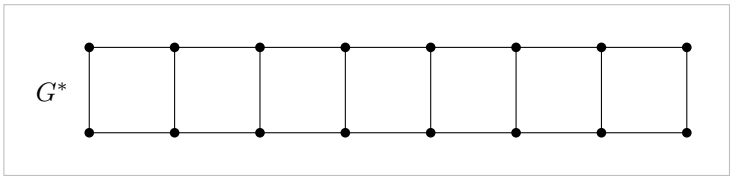

Figure 5 A supergraph of G_2 (graph G_3 is illustrated in Figure 2). G^* is monotone 2-clearable even in the restricted setting of polite and non-stacked lions.

▶ **Observation 4.2.** *The graph G illustrated in Figure 6 is monotone 2-clearable with polite non-stacked lions. However, its subgraph H is not 2-clearable with polite non-stacked lions.*

Note that H is not only a subgraph but also an induced subgraph of G.

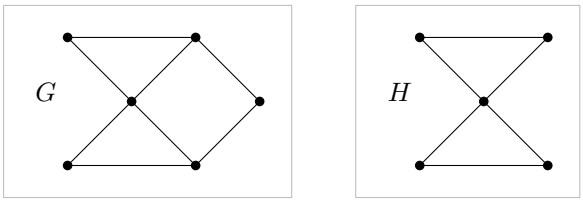

Figure 6 Graph G is 2-clearable with polite and non-stacked lions, whereas subgraph H is not.

5 Conclusion

In the first part of the paper, we studied different types of clearings, namely monotone, polite and non-stacked clearings and we showed some of the relations between them. This gives a good overview over the different restrictions, though a few questions remain open.

In the second part, we focused on the subgraph question raised by Adams et al. [2]. We were able to answer the question in some restricted settings. In the general setting, we believe that there exist graphs with subgraphs that admit no k-clearing. Such a graph might be rather large. A next step in this direction would be to design an algorithm that checks whether a given graph is k-clearable. Such an algorithm, however, may not be easy to find.

References

1 H. Adams and G. Carlsson. Evasion paths in mobile sensor networks. *The International Journal of Robotics Research*, 34(1):90–104, 2015. doi:10.1177/0278364914548051.

2 H. Adams, L. Gibson, and J. Pfaffinger. Lions and contamination, triangular grids, and cheeger constants. *arXiv*, 2020. arXiv:2012.06702.

3 F. Berger, A. Gilbers, A. Grüne, and R. Klein. How many lions are needed to clear a grid? *Algorithms*, 2(3):1069–1086, 2009. doi:10.3390/a2031069.

4 A. Bonato and B. Yang. Graph searching and related problems. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 1511–1558, 2013. doi:10.1007/978-1-4419-7997-1_76.

5 R. Borie, S. Koenig, and C. Tovey. Section 9.5: Pursuit-evasion problems. In J. Yellen J. Gross and P. Zhang, editors, *Handbook of Graph Theory*, pages 1145–1165. Chapman and Hall/CRC, 2013.

6 P. Brass, K. D. Kim, H.-S. Na, and C.-S. Shin. Escaping off-line searchers and a discrete isoperimetric theorem. In *Algorithms and Computation*, pages 65–74, 2007.

7 T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics, a survey, 2011. URL: https://calhoun.nps.edu/handle/10945/45474.

8 V. de Silva and R. Ghrist. Coordinate-free coverage in sensor networks with controlled boundaries via homology. *The International Journal of Robotics Research*, 25(12):1205–1222, 2006. doi:10.1177/0278364906072252.

9 A. Dumitrescu, I. Suzuki, and P. Zylinski. Offline variants of the "lion and man" problem. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, pages 102–111. Association for Computing Machinery, 2007. URL: 10.1145/1247069.1247085.

10 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008. Graph Searching. doi:10.1016/j.tcs.2008.02.040.

11 R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, 1965.

12 A. S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993. doi:10.1145/151261.151263.

13 S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*, 14(6):912–925, 1998. doi:10.1109/70.736775.

14 T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Applications of Graphs*, pages 426–441, 1978.