Saving Critical Nodes with Firefighters is FPT*

Jayesh Choudhari[1], Anirban Dasgupta[2], Neeldhara Misra[3], and M. S. Ramanujan[4]

1 **IIT Gandhinagar, Gandhinagar, India**
choudhari.jayesh@iitgn.ac.in
2 **IIT Gandhinagar, Gandhinagar, India**
anirbandg@iitgn.ac.in
3 **IIT Gandhinagar, Gandhinagar, India**
neeldhara.m@iitgn.ac.in
4 **TU Wien, Vienna, Austria**
ramanujan@ac.tuwien.ac.at

Abstract

We consider the problem of firefighting to save a critical subset of nodes. The firefighting game is a turn-based game played on a graph, where the fire spreads to vertices in a breadth-first manner from a source, and firefighters can be placed on yet unburnt vertices on alternate rounds to block the fire. In this work, we consider the problem of saving a critical subset of nodes from catching fire, given a total budget on the number of firefighters.

We show that the problem is para-NP-hard when parameterized by the size of the critical set. We also show that it is fixed-parameter tractable on general graphs when parameterized by the number of firefighters. We also demonstrate improved running times on trees and establish that the problem is unlikely to admit a polynomial kernelization (even when restricted to trees). Our work is the first to exploit the connection between the firefighting problem and the notions of important separators and tight separator sequences.

Finally, we consider the spreading model of the firefighting game, a closely related problem, and show that the problem of saving a critical set parameterized by the number of firefighters is W[2]-hard, which contrasts our FPT result for the non-spreading model.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases firefighting, cuts, FPT, kernelization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2017.135

1 Introduction

The problem of Firefighting [17] formalizes the question of designing inoculation strategies against a contagion that is spreading through a given network. The goal is to come up with a strategy for placing firefighters on nodes in order to intercept the spread of fire. More precisely, firefighting can be thought of as a turn-based game between two players, traditionally the fire and the firefighter, played on a graph G with a source vertex s. The game proceeds as follows.

- At time step 0, fire breaks out at the vertex s. A vertex on fire is said to be *burned*.
- At every odd time step $i \in \{1, 3, 5, \ldots\}$, when it is the turn of the firefighter, a firefighter is placed at a vertex v that is not already on fire. Such a vertex is permanently *protected*.
- At every even time step $j \in \{2, 4, 6, \ldots\}$, the fire spreads in the natural way: every vertex adjacent to a vertex on fire is burned (unless it was protected).

* A full version of the paper is available at https://arxiv.org/abs/1705.10923.

44th International Colloquium on Automata, Languages, and Programming (ICALP 2017).
Editors: Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl;
Article No. 135; pp. 135:1–135:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The game stops when the fire cannot spread any more. A vertex is said to be *saved* if there is a protected vertex on every path from s to v. The natural algorithmic question associated with this game is to find a strategy that optimizes some desirable criteria, for instance, maximizing the number of saved vertices [4], minimizing the number of rounds, the number of firefighters per round [6], or the number of burned vertices [13, 4], and so on. These questions are well-studied in the literature, and while most variants are NP-hard, approximation and parameterized algorithms have been proposed for various scenarios. See the excellent survey [14] as well as references within for more details.

In this work, we consider the question of finding a strategy that saves a designated subset of vertices, which we shall refer to as the *critical set*. We refer to this problem as Saving A Critical Set (SACS) (we refer the reader to Section 2 for the formal definitions). This is a natural objective in situations where the goal is to save specific locations as opposed to saving some number of them. This version of the problem has been studied by [6, 18, 7] and is known to be NP-hard even when restricted to trees.

Our aim of designing firefighting solutions in order to save a critical set is well-motivated. In the context of studying networked systems for instance, it is often desirable to protect a specific set of critical infrastructure against any vulnerabilities that are cascading through the network (see [15] and [12] for an overview of *survivable network analysis* which aim to design networked systems that survive in the face of failures by providing critical services). Similarly, in the context of handling widely different risk factors that a contagion might have for different sections of the population (e.g. risk-factors that the epidemic of avian flu have for different subpopulations [5]), it is natural to ask for inoculation strategies to protect the identified at-risk groups.

Our Contributions and Methodology. We initiate the study of Saving A Critical Set from a parameterized perspective. We first show that the problem is para-NP-hard when parameterized by the size of the critical set, by showing that Saving A Critical Set is NP-complete even on instances where the size of the critical set is one. It is already clear from known results that Saving A Critical Set is para-NP-hard also when parameterized by treewidth. A third natural parameter is the number of firefighters deployed to save the critical set. Our main result is that Saving A Critical Set is FPT when parameterized by the number of firefighters, although it is not likely to have a polynomial kernel.

Our FPT algorithm is a recursive algorithm that uses the structure of tight separator sequences. The notion of tight separator sequences was introduced in [19] and has several applications [16, 20, 21] (some of which invoke modified definitions). A tight separator sequence is, informally speaking, a sequence of minimal separators such that the reachability set of S_i is contained in the reachability set of S_{i+1}. Note that any firefighting solution is a $s - C$ separator, where s is the source of the fire, and C is the critical subset of vertices. We also obtain faster algorithms on trees by using important separators.

As is common with such approaches, we do not directly solve SACS, but an appropriately generalized form, which encodes information about the behavior of some solution on the "border" vertices, which in this case is the union of all the separators in the tight separator sequence.

Related Work. The Firefighting problem has received much attention in recent years. It has been studied in the parameterized complexity setting [4, 7, 10, 2] but mostly by using the number of vertices burnt or saved as parameters. King et.al. [18] showed that for a tree of degree at most 3, it is NP-hard to save a critical set with budget of one firefighter per round,

but is polynomial time when the fire starts from a vertex of degree at most 2. Chopin [7] extended the hardness result of [18] to a per-round budget $b \geq 1$ and to trees with maximum degree $b + 2$. Chalermsook et.al.[6] gave an approximation to the number of firefighters per round when trying to protect a critical set.

Anshelevich et.al. [1] initiated the study of the spreading model, where the vaccination also spreads through the network. In Section 4 we study this problem in the parameterized setting.

2 Preliminaries

In this section, we introduce the notation and the terminology that we will need to describe our algorithms. Most of our notation is standard. We use $[k]$ to denote the set $\{1, 2, \ldots, k\}$, and we use $[k]_{\mathcal{O}}$ and $[k]_{\mathcal{E}}$, respectively, to denote the odd and even numbers in the set $[k]$.

Graphs, Important Separators and Tight Separator Sequences. We introduce here the most relevant definitions, and use standard notation pertaining to graph theory based on [9, 11]. All our graphs will be simple and undirected unless mentioned otherwise. For a graph $G = (V, E)$ and a vertex v, we use $N(v)$ and $N[v]$ to refer to the open and closed neighborhoods of v, respectively. The *distance* between vertices u, v of G is the length of a shortest path from u to v in G; if no such path exists, the distance is defined to be ∞. A graph G is said to be *connected* if there is a path in G from every vertex of G to every other vertex of G. If $U \subseteq V$ and $G[U]$ is connected, then U itself is said to be connected in G. For a subset $S \subseteq V$, we use the notation $G \setminus S$ to refer to the graph induced by the vertex set $V \setminus S$.

The following definitions about important separators and tight separator sequences will be relevant to our main FPT algorithm. We first define the notion of the reachability set of a subset X with respect to a subset S.

▶ **Definition 1** (Reachable Sets)**.** Let $G = (V, E)$ be an undirected graph, let $X \subseteq V$ and $S \subseteq V \setminus X$. We denote by $R_G(X, S)$ the set of vertices of G *reachable* from X in $G \setminus S$ and by $NR_G(X, S)$ the set of vertices of G not reachable from X in $G \setminus S$. We drop the subscript G if it is clear from the context.

We now turn to the notion of an X-Y separator and what it means for one separator to cover another.

▶ **Definition 2** (Covering by Separators)**.** Let $G = (V, E)$ be an undirected graph and let $X, Y \subset V$ be two disjoint vertex sets. A subset $S \subseteq V \setminus (X \cup Y)$ is called an $X - Y$ separator in G if $R_G(X, S) \cap Y = \emptyset$, or in other words, there is no path from X to Y in the graph $G \setminus S$. We denote by $\lambda_G(X, Y)$ the size of the smallest $X - Y$ separator in G. An $X - Y$ separator S_1 is said to *cover* an $X - Y$ separator S with respect to X if $R(X, S_1) \supset R(X, S)$. If the set X is clear from the context, we just say that S_1 covers S. An $X - Y$ separator is said to be inclusionwise minimal if none of its proper subsets is an $X - Y$ separator.

If $X = \{x\}$ is a singleton, then we abuse notation and refer to a $x - Y$ separator rather than a $\{x\} - Y$ separator. A separator S_1 dominates S if it covers S and is not larger than S in size:

▶ **Definition 3** (Dominating Separators [8])**.** Let $G = (V, E)$ be an undirected graph and let $X, Y \subset V$ be two disjoint vertex sets. An $X - Y$ separator S_1 is said to *dominate* an $X - Y$ separator S with respect to X if $|S_1| \leq |S|$ and S_1 covers S with respect to X. If the set X is clear from the context, we just say that S_1 dominates S.

We finally arrive at the notion of important separators, which are those that are not dominated by any other separator.

▶ **Definition 4** (Important Separators [8]). Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets and $S \subseteq V \setminus (X \cup Y)$ be an $X - Y$ separator in G. We say that S is an *important* $X - Y$ separator if it is inclusionwise minimal and there does not exist another $X - Y$ separator S_1 such that S_1 dominates S with respect to X.

It is useful to know that the number of important separators is bounded as an FPT function of the size of the important separators.

▶ **Lemma 5** ([8]). *Let $G = (V, E)$ be an undirected graph, $X, Y \subset V$ be disjoint vertex sets of G. For every $k \geq 0$ there are at most 4^k important $X - Y$ separators of size at most k. Furthermore, there is an algorithm that runs in time $O(4^k k(m + n))$ which enumerates all such important X-Y separators, where $n = |V|$ and $m = |E|$.*

We are now ready to recall the notion of tight separator sequences introduced in [19]. However, the definition and structural lemmas regarding tight separator sequences used in this paper are closer to that from [21]. Since there are minor modifications in the definition as compared to the one in [21], we give the requisite proofs for the sake of completeness.

▶ **Definition 6.** Let X and Y be two subsets of $V(G)$ and let $k \in \mathbb{N}$. A *tight (X,Y)-reachability sequence* of order k is an ordered collection $\mathcal{H} = \{H_0, H_1, \ldots, H_q\}$ of sets in $V(G)$ satisfying the following properties:

- $X \subseteq H_i \subseteq V(G) \setminus N[Y]$ for any $0 \leq i \leq q$;
- $X = H_0 \subset H_1 \subset H_2 \subset \cdots \subset H_q$;
- for every $0 \leq i \leq q$, H_i is reachable from X in $G[H_i]$ and every vertex in $N(H_i)$ can reach Y in $G - H_i$
  (implying that $N(H_i)$ is a minimal (X,Y)-separator in G);
- $|N(H_i)| \leq k$ for every $1 \leq i \leq q$;
- $N(H_i) \cap N(H_j) = \emptyset$ for all $1 \leq i, j \leq q$ and $i \neq j$;
- For any $0 \leq i \leq q - 1$, there is no (X,Y)-separator S of size at most k where $S \subseteq H_{i+1} \setminus N[H_i]$ or $S \cap N[H_q] = \emptyset$ or $S \subseteq H_1$.

We let $S_i = N(H_i)$, for $1 \leq i \leq q$, $S_{q+1} = Y$, and $\mathcal{S} = \{S_0, S_1, \ldots, S_q, S_{q+1}\}$. We call $\mathcal{S}$ a *tight (X,Y)-separator sequence* of order k.

▶ **Lemma 7** (see for example [21]). *There is an algorithm that, given an n-vertex m-edge graph G, subsets $X, Y \in V(G)$ and an integer k, runs in time $O(kmn^2)$ and either correctly concludes that there is no (X,Y)-separator of size at most k in G or returns the sets $H_0, H_1, H_2 \setminus H_1, \ldots, H_q \setminus H_{q-1}$ corresponding to a tight (X,Y)-reachability sequence $\mathcal{H} = \{H_0, H_1, \ldots, H_q\}$ of order k.*

Proof. The algorithm begins by checking whether there is an X-Y separator of size at most k. If there is no such separator, then it simply outputs the same. Otherwise, it uses the algorithm of Lemma 5 to compute an arbitrary important X-Y separator S of size at most k such that there is no X-Y separator of size at most k that covers S.

Although the algorithm of Lemma 5 requires time $O(4^k k(m + n))$ to enumerate *all* important X-Y separators of size at most k, *one* important separator of the kind described in the previous paragraph can in fact be computed in time $O(kmn)$ by the same algorithm.

If there is no X-S separator of size at most k, we stop and return the set $R(X, S)$ as the only set in a tight (X, Y)-reachability sequence. Otherwise, we recursively compute a tight (X, S)-reachability sequence $\mathcal{P} = \{P_0, \ldots, P_r\}$ of order k and define $\mathcal{Q} = \{P_0, \ldots, P_r, R(X, S)\}$

as a tight (X, Y)-reachability sequence of order k. It is straightforward to see that all the properties required of a tight (X, Y)-reachability sequence are satisfied. Finally, since the time required in each step of the recursion is $O(kmn)$ and the number of recursions is bounded by n, the number of vertices, the claimed running time follows. ◀

Saving a Critical Set. We now turn to the definition of the firefighting problem. The game proceeds as described earlier: we are given a graph G with a vertex $s \in V(G)$. To begin with, the fire breaks out at s and vertex s is burning. At each step $t \geq 1$, first the firefighter protects one vertex not yet on fire - this vertex remains permanently protected - and the fire then spreads from burning vertices to all unprotected neighbors of these vertices. The process stops when the fire cannot spread anymore. In the definitions that follow, we formally define the notion of a firefighting strategy.

▶ **Definition 8** (Firefighting Strategy)**.** A k-step firefighting strategy is defined as a function $\mathfrak{h} : [2k]_{\mathcal{O}} \to V(G)$. Such a strategy is said to be *valid in G with respect to s* if, for all $i \in [2k]_{\mathcal{O}}$, when the fire breaks out in s and firefighters are placed according to $\mathfrak{h}$ for all time steps up to $i-2$, the vertex $\mathfrak{h}(i)$ is not burning at time step i, and the fire cannot spread anymore after timestep $2k$. If G and s are clear from the context, we simply say that $\mathfrak{h}$ is a valid strategy.

▶ **Definition 9** (Saving C)**.** For a vertex s and a subset $C \subseteq V(G) \setminus \{s\}$, a firefighting strategy $\mathfrak{h}$ is said to save C if $\mathfrak{h}$ is a valid strategy and $\{\mathfrak{h}(i) \mid i \in [2k]_{\mathcal{O}}\}$ is a $\{s\}$-C separator in G, in other words, there is no path from s to any vertex in C if firefighters are placed according to $\mathfrak{h}$.

We are now ready to define the parameterized problem that is the focus of this work.

Saving A Critical Set (SACS)	Parameter: k
Input:	An undirected n-vertex graph G, a vertex s, a subset $C \subseteq V(G) \setminus \{s\}$, and an integer k.
Question:	Is there a valid k-step strategy that saves C when a fire breaks out at s?

Parameterized Complexity. We follow standard terminology pertaining to parameterized algorithms based on the monograph [9]. Here we define a known technique to prove kernel lower bounds, called cross composition. Towards this, we first define polynomial equivalence relations.

▶ **Definition 10** (polynomial equivalence relation [3])**.** An equivalence relation $\mathcal{R}$ on Σ^*, where Σ is a finite alphabet, is called a *polynomial equivalence relation* if the following holds: (1) equivalence of any $x, y \in \Sigma^*$ can be checked in time polynomial in $|x| + |y|$, and (2) any finite set $S \subseteq \Sigma^*$ has at most $(\max_{x \in S} |x|)^{O(1)}$ equivalence classes.

▶ **Definition 11** (cross-composition [3])**.** Let $L \subseteq \Sigma^*$ and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L *cross-composes* into Q if there is a polynomial equivalence relation $\mathcal{R}$ and an algorithm which, given t strings $x_1, x_2, \ldots, x_t$ belonging to the same equivalence class of $\mathcal{R}$, computes an instance $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that: (i) $(x^*, k^*) \in Q \Leftrightarrow x_i \in L$ for some $1 \leq i \leq t$ and (ii) k^* is bounded by a polynomial in $(\max_{1 \leq i \leq t} |x_i| + \log t)$.

The following theorem allows us to rule out the existence of a polynomial kernel for a parameterized problem.

▶ **Theorem 12** ([3]). *If an* NP-hard *problem $L \subseteq \Sigma^*$ has a cross-composition into the parameterized problem Q and Q has a polynomial kernel then NP $\subseteq$ coNP/poly.*

3 The Parameterized Complexity of Saving a Critical Set

In this section, we describe the FPT algorithm for Saving A Critical Set and our cross-composition construction for trees. The starting point for our FPT algorithm is the fact that every solution to an instance (G, s, C, k) of SACS is in fact a s-C separator of size at most k. Although the number of such separators may be exponential in the size of the graph, it is a well-known fact that the number of *important* separators is bounded by $4^k n^{O(1)}$ [8]. For several problems, one is able to prove that there exists a solution that is in fact an important separator. In such a situation, an FPT algorithm is immediate by guessing the important separator.

In the SACS problem, unfortunately, there are instances where none of the solutions are important separators. However, this approach turns out to be feasible if we restrict our attention to trees, leading to improved running times. This is described in greater detail in Section 3.2. Further, in Section 3.3, we also show that we do not expect SACS to admit a polynomial kernel under standard complexity-theoretic assumptions. We establish this by a cross-composition from SACS itself, using the standard binary tree approach, similar to [2].

We describe our FPT algorithm for general graphs in Section 3.1. This is an elegant recursive procedure that operates over tight separator sequences, exploiting the fact that a solution can never be contained entirely in the region "between two consecutive separators". Although the natural choice of measure is the solution size, it turns out that the solution size by itself cannot be guaranteed to drop in the recursive instances that we generate. Therefore, we need to define an appropriate generalized instance, and work with a more delicate measure. We now turn to a detailed description of our approach.

We note that the SACS problem is para-NP-complete when parameterized by the size of the critical set, by showing that the problem is already NP-complete when the critical set has only one vertex.

▶ **Theorem 13** ($\star$). SACS *is* NP-complete *even when the critical set has one vertex.*

3.1 The FPT Algorithm

Towards the FPT algorithm for SACS, we first define a generalized firefighting problem as follows. In this problem, in addition to (G, s, C, k), we are also given the following:

- $P \uplus Q \subseteq [2k]_{\mathcal{O}}$, a set of *available time steps*,
- $Y \subset V(G)$, a subset of *predetermined firefighter locations*, and
- a bijection $\gamma : Q \to Y$, a *partial strategy* for Q.

The goal here is to find a valid partial k-step firefighting strategy over $(P \cup Q)$ that is consistent with γ on Q and saves C when the fire breaks out at s. We assume that no firefighters are placed during the time steps $[2k]_{\mathcal{O}} \setminus (P \cup Q)$. For completeness, we formally define the notion of a valid partial firefighting strategy over a set.

▶ **Definition 14** (Partial Firefighting Strategy). A *partial* k-step firefighting strategy on $X \subseteq [2k]_{\mathcal{O}}$ is defined as a function $\mathfrak{h} : X \to V(G)$. Such a strategy is said to be *valid in* G *with respect to* s if, for all $i \in X$, when the fire breaks out in s and firefighters are placed

according to $\mathfrak{h}$ for all time steps upto $[i-1]_{\mathcal{O}} \cap X$, the vertex $\mathfrak{h}(i)$ is not burning at time step i. If G and s are clear from the context, we simply say that $\mathfrak{h}$ is a valid strategy over X.

What it means for partial strategy to save C is also analogous to what it means for a strategy to save C. The only difference here is that we save C despite not placing any firefighters during the time steps j for $j \in [2k]_{\mathcal{O}} \setminus X$.

▶ **Definition 15** (Saving C with a Partial Strategy)**.** For a vertex s and a subset $C \subseteq V(G) \setminus \{s\}$, a partial firefighting strategy $\mathfrak{h}$ over X is said to save C if $\mathfrak{h}$ is a valid strategy and $\cup_{i \in X} \mathfrak{h}(i)$ is a $s-C$ separator in G, in other words, there is no path involving only burning vertices from s to any vertex in C if the fire starts at s and firefighters are placed according to $\mathfrak{h}$.

The intuition for considering this generalized problem is the following: when we recurse, we break the instance G into two parts, say subgraphs G' and H. An optimal strategy for G employs some firefighters in H at some time steps X, and the remaining firefighters in G' at time steps $[2k]_{\mathcal{O}} \setminus X$. When we recurse, we would therefore like to achieve two things:

- Capture the interactions between G' and H when we recursively solve H, so that a partial solution that we obtain from the recursion aligns with the larger graph, and
- Constrain the solution for the instance H to only use time steps in X, "allowing" firefighters to work in G' for the remaining time steps.

The constrained time steps in our generalized problem cater to the second objective, and the predetermined firefighter locations partially cater to the first. We now formally define the generalized problem.

Saving A Critical Set With Restrictions (SACS-R)	Parameter: k
Input:	An undirected n-vertex graph G, vertices s and g, a subset $C \subseteq V(G) \setminus \{s\}$, a subset $P \uplus Q \subseteq [2k]_{\mathcal{O}}$, $Y \subset V(G)$ (such that $\lvert Y\rvert = \lvert Q\rvert$, $2k-1 \in Q$ and $g \in Y$), a bijeciton $\gamma : Q \to Y$ such that $\gamma(2k-1) = g$, and an integer k.
Question:	Is there a valid partial k-step strategy over $P \cup Q$ that is consistent with γ on Q and that saves C when a fire breaks out at s?

We use p and q to denote $|P|$ and $|Q|$, respectively. Note that we can solve an SACS instance (G, s, C, k) by adding an isolated vertex g and solving the SACS-R instance $(G, s, C, 2k+2, g, P, Q, Y, \gamma)$, where $P = [2k]_{\mathcal{O}}$, $Q = \{2k+1\}$, $Y = \{g\}$ and $\gamma(2k+1) = g$. Therefore, it suffices to describe an algorithm that solves SACS-R. The role of the vertex g is mostly technical, and will be clear in due course.

We now describe our algorithm for solving an instance $\mathcal{I} := (G, s, C, k, g, P, Q, Y, \gamma)$ of SACS-R. Throughout this discussion, for the convenience of analysis of Yes instances, let $\mathfrak{h}$ be an arbitrary but fixed valid partial firefighting strategy in G over $P \cup Q$, consistent with γ on Q, that saves C. Our algorithm is recursive and works with pieces of the graph based on a tight $s-C$-separator sequence of separators of size at most $|P|$ in $G \setminus Y$. We describe the algorithm in three parts: the pre-processing phase, the generation of the recursive instances, and the merging of the recursively obtained solutions.

Phase 0 – Preprocessing. Observe that we have the following easy base cases:

- If $G \setminus Y$ has no $s-C$ separators of size at most p, then the algorithm returns No.
- If $p = 0$, then we have a Yes-instance if, and only if, s is separated from C in $G \setminus Y$ and $\mathfrak{h} := \gamma$ is a valid partial firefighting strategy over Q. In this case, the algorithm outputs Yes or No as appropriate.

- If $p > 0$ and s is already separated from C in $G \setminus Y$, then we return YES, since any arbitrary partial strategy over $P \cup Q$ that is consistent with γ on Q is a witness solution.

If we have a non-trivial instance, then our algorithm proceeds as follows. To begin with, we compute a tight $s - C$ separator sequence of order p in $G \setminus Y$. Recalling the notation of Definition 6 , we use $S_0, \ldots, S_{q+1}$ to denote the separators in this sequence, with S_0 being the set $\{s\}$ and $S_{q+1} = C$. We also use $W_0, W_1, \ldots, W_q, W_{q+1}$ to denote the reachability regions between consecutive separators. More precisely, if $\mathcal{H}$ is the tight $s - C$ reachability sequence associated with $\mathcal{S}$, then we have:

$$W_i := H_i \setminus N[H_{i-1}] \text{ for } 1 \leq i \leq q,$$

while W_{q+1} is defined as $G \setminus (N[H_q] \cup C)$. We will also frequently employ the following notation:

$$\mathcal{S} = \bigcup_{i=1}^{q} S_i \text{ and } \mathcal{W} = \bigcup_{i=1}^{q+1} W_i.$$

This is a slight abuse of notation since $\mathcal{S}$ is also used to denote the sequence $S_0, \ldots, S_{q+1}$, but the meaning of $\mathcal{S}$ will typically be clear from the context.

We first observe that if $q > k$, the separator S_q can be used to define a valid partial firefighting strategy. The intuition for this is the following: since every vertex in S_q is at a distance of at least k from s, we may place firefighters on vertices in S_q in any order during the available time steps. Since $|S_q| \leq p$ and S_q is a $s - C$ separator, this is a valid solution. Thus, we have shown the following:

▶ **Lemma 16.** *If G admits a tight $s - C$ separator sequence of order q in $G \setminus Y$ where $q > k$, then $\mathcal{I}$ is a* YES*-instance.*

Therefore, we return YES if $q > k$ and assume that $q \leq k$ whenever the algorithm proceeds to the next phase.

This concludes the pre-processing stage.

Phase 1 – Recursion. Our first step here is to guess a partition of the set of available time steps, P, into $2q + 1$ parts, denoted by $A_0, \ldots, A_q, A_{q+1}$ and $B_1, \ldots, B_{q+1}$. The partition of the time steps represents how a solution might distribute the timings of its firefighting strategy among the sets in $\mathcal{S}$ and $\mathcal{W}$. The set A_i denotes our guess of $\cup_{v \in S_i} \mathfrak{h}^{-1}(v)$ and B_j denotes our guess of $\cup_{v \in W_j} \mathfrak{h}^{-1}(v)$. Note that the number of such partitions is $(2q + 1)^p \leq (2k + 1)^k$. We define $g_0(k) := (2k + 1)^k$. We also use $\mathcal{T}_1(P)$ to denote the partition $A_0, \ldots, A_q$ and $\mathcal{T}_2(P)$ to denote $B_0, \ldots, B_{q+1}$.

We say that the partition $(\mathcal{T}_1(P), \mathcal{T}_2(P))$ is non-trivial if none of the B_i's are such that $B_i = P$. Our algorithm only considers non-trivial partitions – the reason this is sufficient follows from the way tight separator sequences are designed, and this will be made more explicit in due course.

Next, we would like to guess the behavior of a partial strategy over P restricted to $\mathcal{S}$. Informally, we do this by associating a signature with the strategy $\mathfrak{h}$, which is is a labeling of the vertex set with labels corresponding to the status of a vertex in the firefighting game when it is played out according to $\mathfrak{h}$. Every vertex is labeled as either a vertex that had a firefighter placed on it, a burned vertex, or a saved vertex. The labels also carry information about the earliest times at which the vertices attained these statuses. More formally, we have the following definition.

▶ **Definition 17.** Let $\mathfrak{h}$ be a valid k-step firefighting strategy (or a partial strategy over X). The signature of $\mathfrak{h}$ is defined as a labeling $\mathfrak{L}_\mathfrak{h}$ of the vertex set with labels from the set:

$$\mathcal{L} = (\{\mathfrak{f}\} \times X) \cup (\{\mathfrak{b}\} \times [2k]_\mathcal{E}) \cup \{\mathfrak{p}\},$$

where:

$$\mathfrak{L}_\mathfrak{h}(v) = \begin{cases} (\mathfrak{f}, t) & \text{if } \mathfrak{h}(t) = v, \\ (\mathfrak{b}, t) & \text{if } t \text{ is the earliest time step at which } v \text{ burns}, \\ \mathfrak{p} & \text{if } v \text{ is not reachable from } s \text{ in } G \setminus (\{\mathfrak{h}(i) \mid i \in [2k]_\mathcal{O}\}) \end{cases}$$

We use array-style notation to refer to the components of $\mathfrak{L}(v)$, for instance, if $\mathfrak{L}(v) = (\mathfrak{b}, t)$, then $\mathfrak{L}(v)[0] = \mathfrak{b}$ and $\mathfrak{L}(v)[1] = t$. The algorithm begins by guessing the restriction of $\mathfrak{L}_\mathfrak{h}$ on $\mathcal{S}$, that is, it loops over all possible labellings:

$$\mathfrak{T} : \mathcal{S} \to (\{\mathfrak{f}\} \times P) \cup (\{\mathfrak{b}\} \times [2k]_\mathcal{E}) \cup \{\mathfrak{p}\}.$$

The labeling $\mathfrak{T}$ is called legitimate if, for any $u \neq v$, whenever $\mathfrak{T}(u)[0] = \mathfrak{T}(v)[0] = \mathfrak{f}$, we have $\mathfrak{T}(u)[1] \neq \mathfrak{T}(v)[1]$. We say that a labeling $\mathfrak{T}$ over $\mathcal{S}$ is compatible with $\mathcal{T}_1(P) = (A_0, \ldots, A_q)$ if we have:

- for all $0 \leq i \leq r$, if $v \in S_i$ and $\mathfrak{h}(v)[0] = \mathfrak{f}$, then $\mathfrak{h}(v)[1] \in A_i$.
- for all $0 \leq i \leq r$, if $t \in A_i$, there exists a vertex $v \in S_i$ such that $\mathfrak{h}^{-1}(\mathfrak{f}, t) = v$.

The algorithm considers only legitimate labelings compatible with the current choice of $\mathcal{T}_1(P)$. By Lemma 16, we know that any tight $s - C$ separator sequence considered by the algorithm at this stage has at most k separators of size at most p each. Therefore, we have that the number of labelings considered by the algorithm is bounded by $g_1(k) := (p + k + 1)^{(kp)} \leq (3k)^{O(k^2)} \leq k^{O(k^2)}$.

We are now ready to split the graph into $q + 1$ recursive instances. For $1 \leq i \leq q + 1$, let us define $G_i = G[S_{i-1} \cup W_i \cup S_i \cup Y]$. Also, let $\mathfrak{T}_i := \mathfrak{T} \mid_{V(G_i) \cap \mathcal{S}}$. Notice that when using G_i's in recursion, we need to ensure that the independently obtained solutions are compatible with each other on the non-overlapping regions, and consistent on the common parts. We force consistency by carrying forward the information in the signature of $\mathfrak{h}$ using appropriate gadgets, and the compatibility among the W_i's is a result of the partitioning of the time steps.

Fix a partition of the available time steps P into $\mathcal{T}_1(P)$ and $\mathcal{T}_2(P)$, a compatible labeling $\mathfrak{T}$ and $1 \leq i \leq q + 1$. We will now define the SACS-R instance $\mathcal{I}\langle i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}_i\rangle$. Recall that $\mathcal{I} = (G, s, C, k, g, P, Q, Y, \gamma)$. To begin with, we have the following:

- Let $X_i = A_{i-1} \cup A_i$ and let $P_i = B_i$.
- Let $Q_i := X_i \cup Q$ and $Y_i := Y \cup X_i$. We define γ_i as follows:

$$\gamma_i(t) = \begin{cases} \gamma(t) & \text{if } t \in Q, \\ v & \text{if } t \in X_i \text{ and } \mathfrak{T}_i(v) = (\mathfrak{f}, t) \end{cases}$$

Note that γ_i is well-defined because the labeling was legitimate and compatible with $\mathcal{T}_1(P)$. We define H_i to be the graph $\chi(G_i, \mathfrak{T}_i)$, which is described below.

- To begin with, $V(H_i) = V(G_i) \cup \{s^\star, t^\star\}$
- Let $v \in V(G_i)$ be such that $\mathfrak{T}_i(v)[0] = \mathfrak{b}$. Use ℓ to denote $\mathfrak{T}_i(v)[1]/2$. Now, we do the following:
  - Add $k + 1$ internally vertex disjoint paths from $s^\star$ to v of length $\ell + 1$, in other words, these paths have $\ell - 1$ internal vertices.
  - Add $k + 1$ internally vertex disjoint paths from v to g of length $k - \ell - 1$.

Algorithm 1: Solve-SACS-R($\mathcal{I}$)

Input: An instance $(G, s, C, k, g, P, Q, Y, \gamma)$, $p := |P|$
Result: YES if $\mathcal{I}$ is a YES-instance of SACS-R, and NO otherwise.

```
1  if p = 0 and s and C are in different components of G \ Y then return YES;
2  else return NO;
3  if p > 0 and s and C are in different components of G \ Y then return YES;
4  if there is no s − C separator of size at most p then return NO;
5  Compute a tight s − C separator sequence S of order p.
6  if the number of separators in S is greater than k then return YES;
7  else
8      for a non-trivial partition T_1(P), T_2(P) of P into 2q + 1 parts do
9          for a labeling 𝔗 compatible with T_1(P) do
10             if ⋀_{i=1}^{q+1} (Solve-SACS-R(I⟨i, T_1(P), T_2(P), 𝔗_i⟩)) then return YES;
11     return NO
```

- Let $v \in V(G_i)$ be such that $\mathfrak{T}_i(v) = \mathfrak{p}$. Add an edge from v to $t^\star$.
- We also make $k+1$ copies of the vertices $t^\star$ and all vertices that are labeled either burned or saved. This ensures that no firefighters are placed on these vertices.

For $1 \leq i \leq q+1$, the instance $\mathcal{I}\langle i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}_i\rangle$ is now defined as $(\chi(G_i, \mathfrak{T}_i), s^\star, C = \{t^\star\}, k, g, P_i, Q_i, Y_i, \gamma_i)$.

Phase 2 – Merging. Our final output is quite straightforward to describe once we have the $\mathfrak{h}[\mathfrak{T}_i, i]$'s. Consider a fixed partition of the available time steps P into $\mathcal{T}_1(P)$ and $\mathcal{T}_2(P)$, and a labeling $\mathfrak{T}$ of $\mathcal{S}$ compatible with $\mathcal{T}_1(P)$. If all of the $(q+1)$ instances $\mathcal{I}\langle i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}_i\rangle$, $1 \leq i \leq q+1$ return YES, then we also return YES, and we return NO otherwise. Indeed, in the former case, let $\mathfrak{h}[i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}]$ denote a valid partial firefighting strategy for the instance $\mathcal{I}\langle i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}_i\rangle$. We will show that $\mathfrak{h}^\star$, described as follows, is a valid partial firefighting strategy that saves C.

- For the time steps in Q, we employ firefighters according to γ.
- For the time steps in $\mathcal{T}_1(P)$, we employ firefighters according to $\mathfrak{T}$. This is a well-defined strategy since $\mathfrak{T}$ is a compatible labeling.
- For all remaining time steps, i.e, those in $\mathcal{T}_2(P) = \{B_1, \ldots, B_{q+1}\}$, we follow the strategy given by $\mathfrak{h}[i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}]$.

It is easily checked that the strategy described above agrees with $\mathfrak{h}[i, \mathcal{T}_1(P), \mathcal{T}_2(P), \mathfrak{T}]$ for all i. Also, the strategy is well-defined, since $\mathcal{T}_1(P)$ and $\mathcal{T}_2(P)$ form a partition of the available time steps. Next, we will demonstrate that $\mathfrak{h}^\star$ is indeed a valid strategy that saves C, and also analyze the running time of the algorithm.

Due to lack of space, we refer the reader to the full version of this work for the analysis of the algorithm.

3.2 A Faster Algorithm For Trees

In this section we consider the setting when the input graph G is a tree. WLOG, we consider the vertex s to be the root of the tree. We first state an easy claim that shows that WLOG, we can consider the critical set to be the leaves. The proof of the following lemma follows from the fact that the firefighting solution has to be a $s - C$ separator.

▶ **Lemma 18.** *When the input graph G is a tree, if there exists a solution to* SACS*, there exists a solution such that all firefighter locations are on nodes that are on some path from s to C.*

Given the above claim, our algorithm to construct a firefighting solution is the following– exhaustively search all the important $s - C$ separators that are of size k. For each vertex v in a separator Y, we place firefighters on Y in the increasing order of distance from s and check whether this is a valid solution. The following lemma claims that if there exists a firefighting solution, the above algorithm will return one.

▶ **Lemma 19** (⋆). *Solving the* SACS *problem for input graphs that are trees takes time $O^*(4^k)$.*

3.3 No Polynomial Kernel, Even on Trees

Given that there is a FPT algorithm for SACS when restricted to trees, in this section we show that SACS on trees has no polynomial kernel. As mentioned before, the proof technique used here is on the similar lines of the proof showing no polynomial kernel for SAVING ALL BUT k-VERTICES by Bazgan et. al.[2].

▶ **Theorem 20** (⋆). SACS *when restricted to trees does not admit polynomial kernel, unless NP $\subseteq$ coNP/poly.*

4 The Spreading Model

The spreading model for firefighters was defined by Anshelevich et al. [1] as "Spreading Vaccination Model". In contrast to the firefighting game described in Section 1, in the spreading model, the firefighters (vaccination) also spread at even time steps as similar to that of the fire. That is, at any even time step if there is a firefighter at node v_i, then the firefighter extends (vaccination spreads) to all the neighbors of v_i which are not already on fire or are not already protected by a firefighter. Consider a node v_i which is not already protected or burning at time step $2j$. If u_i and w_i are neighbors of v_i, such that, u_i was already burning at time step $2j - 1$ and w_i was protected at time step $2j - 1$, then at time step $2j$, v_i is protected. That is, in the spreading model the firefighters dominate or win over fire. For the spreading model, the firefighting game can be defined formally as follows:

- At time step 0, fire breaks out at the vertex s. A vertex on fire is said to be *burned*.
- At every odd time step $i \in \{1, 3, 5, \ldots\}$, when it is the turn of the firefighter, a firefighter is placed at a vertex v that is not already on fire. Such a vertex is permanently *protected*.
- At every even time step $j \in \{2, 4, 6, \ldots\}$, first the firefighter extends to every adjacent vertex to a vertex protected by a firefighter (unless it was already protected or burned), then the fire spreads to every vertex adjacent to a vertex on fire (unless it was already protected or burned). Needless to say, the vertices protected at even time steps are also permanently *protected*.

In the following theorem, we show that in spite of the spreading power that the firefighters have, SACS is hard.

▶ **Theorem 21** (⋆). *In the spreading model,* SACS *is as hard as k-*Dominating Set.

5 Summary and Conclusions

In this work, we presented the first FPT algorithm, parameterized by the number of firefighters, for a variant of the Firefighter problem where we are interested in protecting a critical set. We also presented a faster algorithms on trees. In contrast, we also show that in the spreading model protecting a critical set is W[2]-hard. Our algorithms exploit the machinery of important separators and tight separator sequences. We believe that this opens up an interesting approach for studying other variants of the Firefighter problem.

References

1 Elliot Anshelevich, Deeparnab Chakrabarty, Ameya Hate, and Chaitanya Swamy. Approximation algorithms for the firefighter problem: Cuts over time and submodularity. In *International Symposium on Algorithms and Computation*, pages 974–983. Springer, 2009.

2 Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R Fellows, Fedor V Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. *Journal of Computer and System Sciences*, 80(7):1285–1297, 2014.

3 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

4 Leizhen Cai, Elad Verbin, and Lin Yang. Firefighting on trees:$(1 - 1/e)$–approximation, fixed parameter tractability and a subexponential algorithm. In *International Symposium on Algorithms and Computation*, pages 258–269. Springer, 2008.

5 Center for Disease Control. People at High Risk of Developing Flu–Related Complications. `https://www.cdc.gov/flu/about/disease/high_risk.htm`, 2016.

6 Parinya Chalermsook and Julia Chuzhoy. Resource minimization for fire containment. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1334–1349. Society for Industrial and Applied Mathematics, 2010.

7 Morgan Chopin. *Optimization problems with propagation in graphs: Parameterized complexity and approximation*. PhD thesis, Université Paris Dauphine-Paris IX, 2013.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 Marek Cygan, Fedor V Fomin, ukasz Kowalik, Daniel Lokshtanov, D 'aniel Marx, Marcin Pilipczuk, Micha Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, Cham, Cham, 2015.

10 Marek Cygan, Fedor V Fomin, and Erik Jan van Leeuwen. Parameterized Complexity of Firefighting Revisited. In *Parameterized and exact computation*, pages 13–26. Springer, Heidelberg, Berlin, Heidelberg, 2012.

11 Reinhard Diestel. *Graph Theory*. Springer Graduate Text GTM 173. Reinhard Diestel, July 2012.

12 Robert J Ellison, David A Fisher, Richard C Linger, Howard F Lipson, and Thomas Longstaff. Survivable network systems: An emerging discipline. Technical report, DTIC Document, 1997.

13 S Finbow, B Hartnell, Q Li, and K Schmeisser. On minimizing the effects of fire or a virus on a network. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 33:311–322, 2000.

14 Stephen Finbow and Gary MacGillivray. The firefighter problem: a survey of results, directions and questions. *The Australasian Journal of Combinatorics*, 43:57–77, 2009.

15 Howard Frank and I Frisch. Analysis and design of survivable networks. *IEEE Transactions on Communication Technology*, 18(5):501–519, 1970.

16 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1670–1681, 2016.

17 Bert Hartnell. Firefighter! an application of domination. In *25th Manitoba Conference on Combinatorial Mathematics and Computing*, 1995.

18 Andrew King and Gary MacGillivray. The firefighter problem for cubic graphs. *Discrete Mathematics*, 310(3):614–621, 2010.

19 Daniel Lokshtanov and M. S. Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 750–761, 2012.

20 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. A linear time parameterized algorithm for directed feedback vertex set. *CoRR*, abs/1609.04347, 2016.

21 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. A linear time parameterized algorithm for node unique label cover. *CoRR*, abs/1604.08764, 2016.