Rheinische Friedrich-Wilhelms-Universität Bonn
Mathematisch-Naturwissenschaftliche Fakultät

Theoretical Aspects of Intruder Search

MA-INF 1318 Manuscript Wintersemsester 2015/2016

Elmar Langetepe

Bonn, 19. October 2015

The manuscript will be successively extended during the lecture in the Wintersemester. Hints and comments for improvements can be given to Elmar Langetepe by E-Mail `elmar.langetepe@informatik.uni-bonn.de`. Thanks in advance!

Chapter 2

Discrete Scenarios

In this chapter we would like to discuss several results on the firefigther or intruder problem on discrete graph structures. The main problem is NP-complete already for trees but there are some variants that can be solved in polynomial time. Besides there are efficient approximation schemes.

We consider dynamic problems for connected graphs $G = (V, E)$ with a root vertex r where the fire or contamination starts only at r. At each time step the firefigther can protect p vertices that are not contaminated. Afterwards the contamination spreads to all unprotected vertices neighboring a contaminated vertex. A protection strategy describes the sequence S of vertices that will be protected over time.

Firefighter Decision Problem for Graphs (Saving k vertices):
Instance: A Graph $G = (V, E)$ of degree d with root vertex r and p firefigther per step and an integer k.
Question: If the fire breaks out at vertex r, is there a protecting strategy so that at least k vertices can be protected?

An optimal protection strategy protects the maximum number of vertices. The problem is NP-complete already for trees of small degree. This general proof is a bit technical and would take some time so we will use a somewhat simpler proof for NP-completeness on graphs.

2.1 Graphs

2.1.1 Polynomial time algorithm for special graphs

Let us assume that the degree d of the vertices of G is bounded to 3. Thus it only makes sense to set p to one. Otherwise the game ends in the very beginning. If the start vertex has degree 2, there is a simple polynomial time algorithm. In principle we force the fire to spread along a path until finally some vertex is enclosed. The running time depends on the nature of this vertex.

We first introduce this measure and a corresponding strategy for all vertices u of G. Let $\text{dist}(u, r)$ denote the length of a shortest path from r to u. Let V_1 denote all vertices of degree 1 and let V_2 denote all vertices of degree 2 and let V_c denote all vertices of degree 3 that belong to a cycle. Let $C(u)$ denote the smallest cycle containing u. The following Lemma shows how long it could take to enclose a vertex u which is finally on fire; see also Figure 2.1.

Lemma 6 *Vertices from $V_1 \cup V_2$ can be enclosed in time $dist(u, r) + 1$ and only $dist(u, r) + 1$ vertices are on fire. Vertices from V_c can be enclosed in time $dist(u, r) + C(u) - 1$ and only $dist(u, r) + C(u) - 1$ vertices are on fire.*

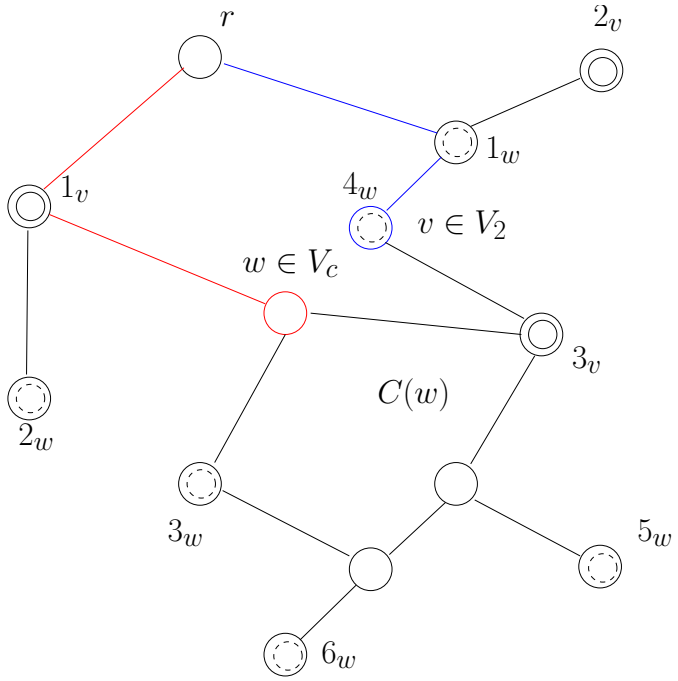

Figure 2.1: Protecting a vertex $v \in V_2$ and $w \in V_c$. The blue and red path denotes the shortest path. The labels i_v and j_w denote the vertices that will be protected at step i and j until v and w are finally enclosed. respectively.

Proof. For the above statements we simply follow a shortest path from r to u. In each time step we defend the vertex that is adjacent to the burning vertex but not on the path to u. This shows the property for all $u \in V_1 \cup V_2$.

For $u \in V_c$ after $\text{dist}(r, u) + 1$ time steps we are reaching u. For the remaining steps we protect the vertex not on $C(u)$ but with a burning neighbor. Thus we encircle u in any case. □

Now we would like to prove a structural property for an optimal strategy. The first protected vertex can always be one of the neighbors of r and after that it also suffices to always defend the vertices neighboring the fire as the following Lemma shows.

Lemma 7 *For a setting $(G, r, 1)$ where G has maximal degree 3 and root r has degree ≤ 2 there is always an optimal protection strategy that protects the neighbor of a contaminated vertex in each time step.*

Proof. If r has degree 1, the statement is trivial. So for $r = 2$ assume that the statement is false and $(G, r, 1)$ is a minimal counterexample. Let v_1 and v_2 be the neighbors of r. If there is an optimal strategy that first protects the neighboring vertex v_1 of r, let G' denote the Graph that is attained by deleting r and v_1. Thus $(G', v_2, 1)$ is also a counterexample that contradicts the minimality.

So we can assume that in a minimal counterexample in the first step neither v_1 nor v_2 will be protected. Let u be the vertex protected first with shortest path distance ≥ 2 to r.

Consider the end of the optimal strategy. If u has no burning neighbor, the strategy could not have been optimal. If u has only one burning neighbor, we can improve the strategy by protecting this neighbor in the first step. If there are two neighbors of u that are burning at the end of the strategy, the vertex u lies on a cycle which is completely burned except for u. Thus the strategy implied in Lemma 6 for one of the neighbors of u is an optimal variant. □

Exercise 4 *Show that the statement of Lemma 6 does not hold for a degree 3 starting vertex. Present a non-trivial example for the optimal strategy of a graph G of degree 3 and root vertex of degree 2 based upon the above calculations.*

Finally, we suggest the following strategy. Let

$$f(u) := \begin{cases} \text{dist}(u,r)+1 & : \text{ if } u \in V_1 \cup V_2 \\ \text{dist}(u,r)+C(u)-1 & : \text{ if } u \in V_c \setminus V_2 \\ \infty & : \text{ otherwise} \end{cases}$$

and find a vertex u with $f(u) = \min_{x \in V} f(x)$. Enclose this vertex by the path strategy.

Theorem 8 *For a problem instance $(G, r, 1)$ of a graph G of maximum degree 3 and a root vertex of degree 2 the above strategy is optimal.*

Proof. In Lemma 7 we have shown that there is always an optimal strategy that protects the neighbor of a contaminated vertex. Let u be one of the vertices burning last. If u has degree 1 or 2, we require at least $d(r,u)+1$ time steps and $d(r,u)+1$ are burning, which is optimal.

If u has degree one there are three neighbors n_1, n_2 and n_3. If only one neighbor n_1 is on fire, u, n_2 and n_3 lie on a cycle that is totally burning, a contradiction to the path strategy.

So let us assume that two neighbors n_2 and n_3 of u are protected and one neighbor n_1 is on fire. There should be another neighbor of either n_2 or n_3 that is on fire. If this is not the case, we could have blocked u one step earlier. This means that one of the neighbors n_2 or n_3 of u has to build a cycle $C(u)$ with u and n_1 so that any vertex of this cycle burns except for u. In this case the above strategy optimizes the best such cycle and is as least as good as the optimum.

Since the degree is bounded, there are no other cases. □

Theorem 9 *For a problem instance $(G, r, 1)$ of a graph $G = (V, E)$ of maximum degree 3 and a root vertex of degree 2 the decision problem can be solved in polynomial time and the maximum number of vertices that can be saved is $|V| - \min_{x \in V} f(x)$.*

Proof. By the above considerations. Just compute $f(x)$ for every vertex $x \in v$ taking the shortest path and the smallest cycle into account. □

Exercise 5 *Analyse the precise running time for the computation of the optimal strategy. That is, present an algorithm and its running time that computes $f(x)$ for every vertex $x \in v$ efficiently.*

Lemma 6 suggests to always choose a vertex close to the fire. For an arbitrary given tree with root r it seems that this is also the best option we will prove this statement in Lemma 12. It seem that for trees a greedy approach could be optimal but this is not the case as Section 2.2.1 shows.

2.1.2 NP-Completeness for graphs

Theorem 10 *The firefighter decision problem for graphs is NP-hard.*

Proof. We reduce the k-Clique problem to the firefigther problem. So let $G = (V, E)$ be a graph and k an integer: Is there a Clique of size k inside G? This question is known to be

NP-complete. We assume that G has at least $k+1$ non-isolated vertices. Otherwise the answer is trivial.

For such a graph G we construct a bipartite graph G' as follows; see Figure 2.2. For every vertex $v \in V$ we construct a vertex s_v of V' and for every edge $e = (u, v) \in E$ we construct a vertex s_e of V'. For edge $e = (u, v)$ we construct two edges (s_u, s_e) and (s_v, s_e) in E'.

Additionally, we construct $i = 1, 2, \ldots, k-1$ columns of k vertices where each vertex of column i has an edges to every vertex on column $i+1$. For every vertex of column $k-1$ there is an edge to every $s_v \in V'$. Finally, there is a unique vertex r that is connected to all vertices of the first column.

The construction means that after $k-1$ steps from r at least one vertex of column $k-1$ is burning and threatens all vertices s_v.

So if we protect a k-Clique of V by choosing the corresponding vertices $s_v \in V'$ in the first k steps, we will protect the k-Clique vertices and also the $\binom{k}{2}$ (edge) vertices s_e of V' of the k-Clique. In an additional step $k+1$ we can save one more edge vertex s_e. This vertex exists because we assumed that $k+1$ vertices in G are non-isolated. Thus, if a k-Clique exists, we can save $k' = k + \binom{k}{2} + 1$ vertices.

So we would like to answer the decision problem for G', root r and k'. The graph G' is constructed in polynomial time.

Conversely, suppose there is a strategy that saves at least k' vertices. After k time steps the fire will always reach the vertices s_v since the $k-1$ columns contain k vertices. For the first $t \geq k-1$ steps, it is not helpful to protect a vertex in one of the columns from 1 to $k-1$ because they only will protect themself and we can also choose one of s_v or s_e instead. This also means that after k steps, all vertices except those chosen by the strategy are burning. Protecting one of the vertices s_e before step k is also needless because it only saves itself. Therefore the best one can do is, choose k vertices s_v in the first k steps. These k vertices can save at most $\binom{k}{2}$ vertices, which is only possible if the chosen vertices build a k-Clique in G. □

2.2 Trees

2.2.1 Greedy approximation for a tree

A greedy algorithm for a tree always protects the subtree with the largest number of vertices. Figure 2.3 shows that this strategy is not better than a 1/2 approximation. The optimal strategy protects $2(k-1)$ vertices whereas greedy protects only $k+1$ vertices. Note that the greedy algorithm always choose a vertex neighboring a contaminated vertex but also the optimal does as we will prove later.

Theorem 11 *For a problem instance $(T, r, 1)$ of a rooted tree $T = (V, E)$ the greedy strategy gives a $\frac{1}{2}$ approximation for the optimal number of vertices protected. This bound is tight.*

Proof. As shown by Figure 2.3 greedy is not better than $\frac{k+1}{2(k-1)} \mapsto \frac{1}{2}$.

For the upper bound we can subdivide the greedy strategy S into steps that outperform the current optimal move and steps that do not outperform the current optimal move. Outperforming means saving at least as many vertices as the optimal move in the same step. For an optimal strategy opt let opt_A (greedy is not worse) denote the first category of safe vertices and let opt_B (greedy is worse) denote the second category of safe vertices. Let S_G denote the number of vertices saved by S_G. We can assume that each step of a strategy cuts of a subtree of T and as long as the fire spreads the distance of burning vertices to the root increases by 1 in every time step.

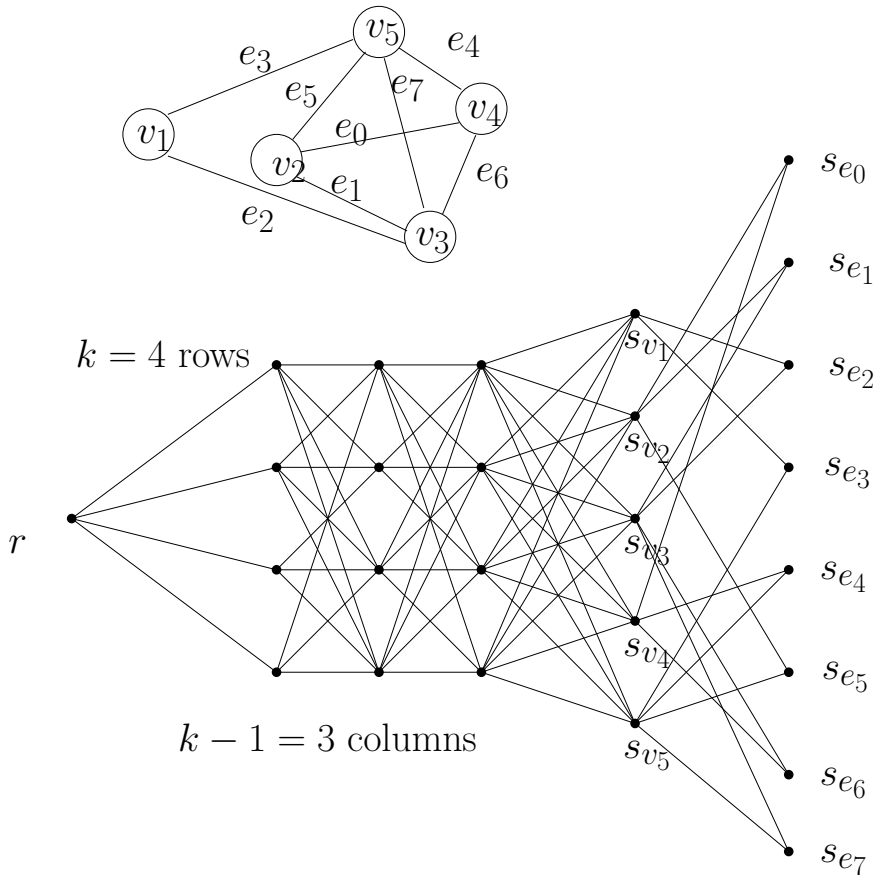

Figure 2.2: Protecting the vertices of the k-Clique in k steps and an additional vertex s_{e_j} gives $k' = k + \left(\frac{k}{2}\right) + 1$ protected vertices in total.

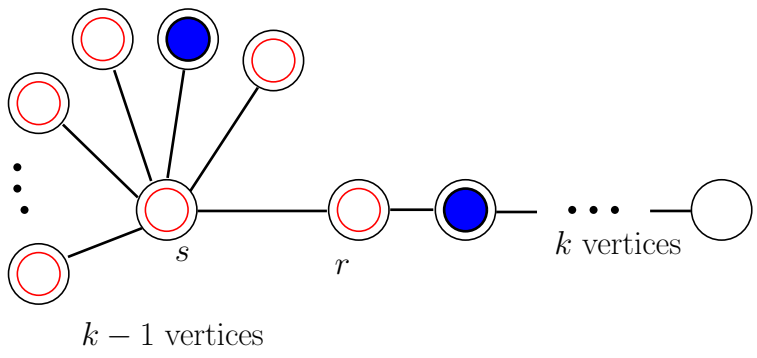

Figure 2.3: The greedy algorithm does not protect the vertex s first and only saves $k+1$ vertices.

We would like to show that $2S_G \geq \text{opt}_A + \text{opt}_B$ holds. This can be seen as follows. Greedy starts with a step as least as good as the optimal one because it is greedy. Let us assume that at some moment in time the optimal move is better than a greedy move or there is no greedy move anymore. Then in one of the previous steps greedy has chosen a predecessor of the optimal move v, otherwise the greedy algorithm could have also picked the optimal move now because of the distance of vertex v to the root. Thus the vertices saved by the optimal move now have already been saved by greedy before. □

Exercise 6 *Give an example for the proof idea of Theorem 11. Show that Theorem 11 also holds for the case (T, r, p) for $p > 1$.*

2.2.2 Exponential time algorithm for general trees

The decision problem is already NP-complete for trees. In this subsection we show that the problem is fixed-parameter-tractable by given a somewhat efficient algorithm.

For this we also introduce another variant of the game. How many vertices can be saved, if only k guards can be placed.

Firefighter Decision Problem (Protection by k guards):
Instance: A Graph $G = (V, E)$ of degree d with root vertex r and p firefigther per step and an integer k.
Question: What is the strategy that saves a maximum number of vertices by protecting k vertices in total?

The algorithm is based on the following structural property for a strategy on trees.

Lemma 12 *For any optimal strategy for an instance of the firefigther decision problems on trees (protection by k guards, saving k vertices) the vertex defended at each time is adjacent to a burning vertex. There is an integer l, so that all protected vertices have depth at most l, exactly one vertex p_i at each depth is protected and all ancestors of p_i are burning.*

Proof. If the vertex defended at some time step t has no burning neighbor, it is possible to improve the strategy by protecting the vertex closer to the root. Thus, in any time step t one vertex at depth t is chosen. □

Exercise 7 *Show an example that Lemma 12 does not hold for a a general graph.*

Now we present an efficient algorithm that computes an optimal strategy by dynamic programming for the above maximum protection-by-k-guards setting. An example is given in Figure 2.4.

Starting from the root vertex we label the vertices in pre-order. By the pre-order we define the relation *v lies to the left of w* by $\text{pre}(v) < pre(w)$. Due to Lemma 12 is suffices to consider the vertices up to depth k only.

For the dynamic program process we more generally consider a substrategy where we place a guard at step t or we do not place a guard at step t. This is due to the fact, that in step t we cut of a subtree, consider a smaller tree and assume that in this remaining tree no guard was placed at step t. Therefore we have a vector $X \in \{0, 1\}^k$ that represents a general strategy. $X(j) = 1$ denotes the case that at step j a guard will be placed at depth j and $X(j) = 0$ denotes the case where no guard is placed at step j.

Let L_k be the set of vertices of the tree T with depth $\leq k$. For any $v \in L$, we consider the subtree T^v of T with vertices from L_k and pre-order at most $\text{pre}(v)$ (all vertices of L_k to the left of v including v). We would like to compute the maximum number $A_v(X)$ of vertices that

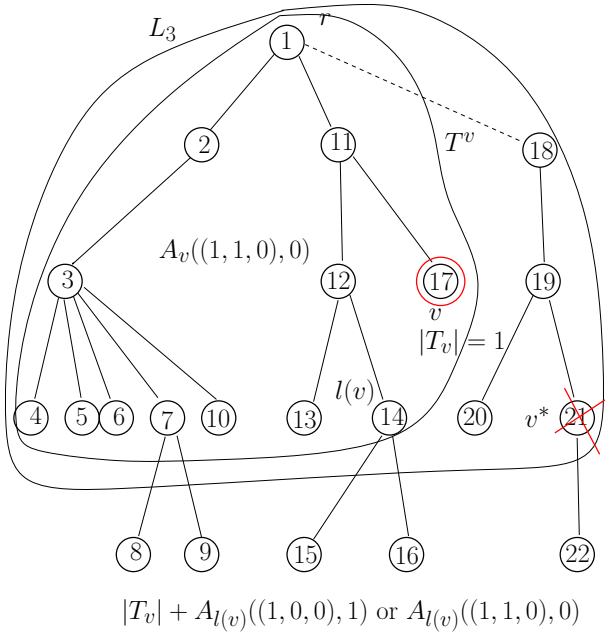

Figure 2.4: The pre-order of a tree T, the set L_k of vertices of depth $\leq k$. Computing $A_v((1,1,0),0)$ means that we are searching for a strategy in the tree T^v that sets guards in the first and second depth and if a vertex is protected along the path from r to v, then its depth is greater than 0. For the recursion we consider two cases. If the vertex v will be protected, we are looking for $|T(v)| + A_{l(v)}((1,0,0),1)$. Here the second parameter 1 says that we are not allowed to block a predecessor of v any more in this case. If the vertex v will be not protected, we are looking for $A_{l(v)}((1,1,0),0)$. The maximum of both is the best choice.

can be saved in T_v if we apply a strategy for up to k steps that behaves as indicated by the vector X.

There will be a rightmost vertex v^* of depth k in T and we are obviously interested in $A_{v^*}(1^k)$. This is the end value we would like to compute.

Let us try to explain the recursive formula for a vertex v in general. By dynamic programming in step j we can either choose vertex v of depth j to be protected in step j or not. Let $|T_v|$ denote the size of the subtree of T starting with root v.

If v is protected, we have saved $|T_v|$ vertices cut of the tree T_v, exchange the entry j of the current vector X by 0 and turn over to the next entry in L_k with pre-order less than v. For vector X and vertex v of depth j let, X^v denote the $0-1$ vector where the entry of X at j is set to 0. Let $l(v)$ denote the vertex of L_k with largest pre-order smaller than v. So in the recursion just consider to compute $A_{l(v)}(X^v)$ and add $|T_v|$ in this case.

There is a problem because in the subtree $T^{l(v)}$ it might be allowed to place a guard on the shortest path from r to v which is no more allowed, if v was protected before. Therefore we introduce a second variable i and always compute $A_v(X,i)$ where i denotes the depth along the path from r to v after that a vertex can be chosen for protection. In the above case we have to compute $A_{l(v)}(X^v, \text{depth}(v)-1)$.

We can assume that at v the recursion was started with $A_v(X,i)$. And the above case only have to be taken into account if $X(\text{depth}(v)) = 1$ and $\text{depth}(v) > i$.

Now let us assume that v will not be protected. In this case for $A_v(X,i)$ we do not change X and turn over to $A_{l(v)}$ for the predecessor $l(v)$ of v in L_k. The parent of v of depth $(\text{depth}(v)-1)$ is always on the path of r to $l(v)$. If it was only allowed to set a guard at depth larger than i on the path from r to v, we only have to take care that i is not larger than $\text{depth}(v)-1$. This means that $A_{l(v)}(X, \min(\text{depth}(v)-1, i))$ has to be computed.

Theorem 13 *Computing the optimal protection strategy for k guards on a tree T of size n can be done in $O(n2^k k)$ time.*

Proof. By the above definitions we consider a dynamic programming approach with

$$A_v(X,i) = \max \left\{ \begin{array}{l} A_{l(v)}(X, \min(\text{depth}(v)-1, i)) \\ \\ [X(\text{depth}(v) = 1 \text{ and } \text{depth}(v) > i] \cdot |T_v| + A_{l(v)}(X^v, \text{depth}(v)-1) \end{array} \right\}$$

where $[\phi]$ equals 1 if ϕ holds true and 0 otherwise.

For $v \in L_k$ the value $A_v(X,i)$ just denotes the optimal protection number for a strategy in T^v that sets a number of guards on each depth w.r.t. the entries in X and does not set a guard on the path from r to v before or at depth i.

We are simply searching for $A_{v^*}(1^k, 0)$ where vertex v^* is the rightmost vertex of depth k in T.

By the above description the dynamic programming procedure is correct. We compute L_k, $l(v)$ and $|T_v|$ for each vertex v in linear time by a pre-order walkthrough. Then we traverse the vertices of L_k from left to right and we have at most $n \times 2^k \times k$ entries $A_v(X,i)$ where n stands for v, 2^k stands for X and k stands for i. □

Exercise 8 *Compute the best strategy of the example in Figure 2.4 for $k = 2$ and with the dynamic programming approach introduced above.*

Exercise 9 *Consider a binary tree and design an optimal strategy. Analyse the time complexity for computing the strategy.*

Exercise 10 *For a given tree, formalize the computation of an optimal strategy by an Integer Linear Program.*

With the above algorithm we can also answer the question whether k vertices can be saved.

Corollary 14 *Computing a strategy for a tree T of size n that saves at least k vertices can be done in $O(n2^k k)$ time.*

Proof. If (T, r, k) is an instance for the saving-k-vertices variant. It is clear that a strategy with k placements will solve the problem. Therefore it suffices to run the above algorithm for $i = 1, \ldots, k$. We can save k vertices if and only if we can save at least k vertices for some value of i.

Thus we have

$$\sum_{i=1}^{k} i2^i n \leq kn \sum_{i=1}^{k} 2^i = (2^{k+1} - 2)kn$$

so that the worst-case running time is in $O(n2^k k)$.

□

Finally, we would like to build up a good subexponential time bound and make use of the following structural property in order to find a good bound on k w.r.t. n.

Lemma 15 *If a vertex at depth d is burning in an optimal strategy for an instance of the firefigther problem on trees, at least $\frac{1}{2}(d^2 - d)$ vertices are safe.*

Proof. Let us assume that in T and for an optimal startegy, a vertex v at depth d is burning. Then by Lemma 12 there is a protected vertex v_i for any depth $i = 1, \ldots, d$. Any tree T_{v_i} should contain at least $d - i + 1$ vertices. Otherwise it was better to choose a vertex along the path from r to v in this step i. Thus

$$\sum_{i=1}^{d} (d - i + 1) = \frac{1}{2}(d^2 - d)$$

gives the bound. □

Theorem 16 *There is an $O\left(2^{\sqrt{2n}} n^{3/2}\right)$ algorithm for the firefigther problem on trees.*

Proof. We show that we can run the algorithm of Theorem 13 for $k \leq \sqrt{2n}$. Suppose a vertex of depth $\sqrt{2n}$ is burning. Then by Lemma 15 $n + \sqrt{n/2} > n$ vertices are safe witch contradicts the number n of vertices. This means that all vertices of depth $\sqrt{2n}$ are safe in an optimal strategy. In turnan optimal strategy makes use of less than $\sqrt{2n}$ guards. Thus we set $k \leq \sqrt{2n}$ which gives the bound. □