**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!

# Contents

# Chapter 1

# Introduction

In this lecture we consider intruder and evader problems from an algorithmic point of view. We discuss problems in the context of motion planning in discrete and continuous environments. Discrete environments will be mainly defined by graphs whereras in the continuous case we will make use of the Euclidean plane or subsets of the Euclidean plane.

Generally, there is a set of searcher (or guards) and another set of intruder (or evaders) that compete with each other. The intruder tries to escape from the searcher in the given environment. The other way round the guards would like to protect a certain area from the intruder. Intruder and searcher can have different facilities or sensors. There is always some kind of dynamics, the searcher and/or the intruder can move or perform tasks over time.

The intruder search problem might also be considered as a scenario for the cleaning or the enclosure of a spreading contamination. For any moment in time the current contamination represents all possible remaining positions of the intruder. So the task of the searcher is to reduce the number or to bound the area of these positions and to give a guarantee where the intruder is located.

More precisely, from a theoretical point of view we would like to answer the following questions:

- Computational complexity (i.e. existence of a strategy, question for the number of guards required, running time for the computation of a strategy)
- Correctness or Failure (i.e. success of a given strategy, limits of the strategy)
- Efficiency (i.e. number of steps or path length required for a given strategy, area saved by the guards)
- Optimality (i.e. cost measures for the efficiency and its analysis)

We will also see that there are many interesting open questions. Sometimes simulations will be helpful for getting more insight into the problem and a conjecture for a solution. Additionally, the usage of simulation tools will be part of this lecture.

## 1.1 Introductory examples

In the following we would like to discuss some simple introductory examples and relate it to the above questions. This gives some insight into the nature of the problems we would like to discuss and also shows the variety of methods that will be applied.

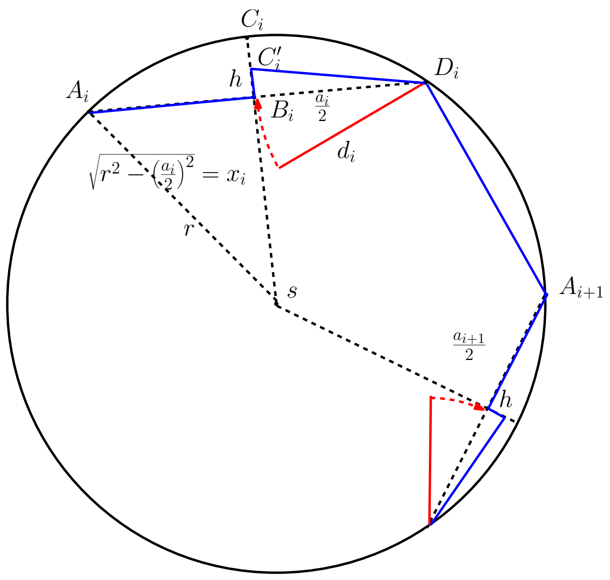


Figure 1.1: Constructing a polygon with doors for a set of $n$ integer $a_1, a_2, \ldots, a_n$. The radius $r$ and the speed $v$ are chosen appropriately so that after $t + 0.5$ time steps the intruders can reach any point $C_i'$ but before $t$ time steps no point $B_i$ can be reached.

### 1.1.1 Protecting parts of a polygonal area from a set of intruders

Let us assume that we have a simple room modeled by a polygonal area $P$ (a simple polygon) and there is a set of $n$ doors that can be closed successively. Additionally, there is a set of intruders starting at some point $s$. The intruder moves with speed $v$. For a given set of $n$ doors each door can protect the area behind the door. The time for closing a single door $d_i$ is given by a value $a_i$. Any door $d_i$ protects a portion $A_i$ of the polygon from the intruders. (Somehow it makes sense that the time $a_i$ for closing the door is proportional to the area $A_i$ that can be saved by $d_i$.) We would like to close as many doors as possible, thus maximizing the portion that cannot be entered by the intruders. The doors can only be closed one after the other.

The question is, which doors should be closed successively, so that the intruders cannot enter the part of the polygon behind the doors. We would like to maximize the area that will be safe. As a matter of computational complexity we show that this problem is $NP$-hard.

*Optimal-Closing-Sequence*
**Instance:** A simple polygon, a set of $n$ intruders with starting positions and speed 1, a set of $m$ *doors* that can be closed successively in time $c_i$ and saves area $A_i$.
**Output:** Compute the optimal sequence of doors that has to be closed for maximizing the area saved.

**Theorem 1** *Computing the optimal closing-sequence for a set of doors in a polygon is NP-hard in general.*

**Proof.** The subset-sum problem can be reduced to our problem. We have $n$ integer numbers $a_1, a_2, \ldots, a_n$ and a integer threshold $t$ and we are searching for a subset-sum, so that the sum does not exceed $t$ but is as close as possible to $t$. [1] The subset-sum problem will be transformed into our problem as depicted in Figure 1.1. We make use of a circle $C_r$ of radius $r$. Let $s$ be the center of the circle. Along the boundary we use non-intersecting secants of length $a_i$ with endpoints $A_i$ and $D_i$ in clockwise order. A ray starting at $s$ and running through the midpoint

[1]Note that partition can be reduced to the subset-sum problem.

$B_i$ of each secant hits the boundary of $C_r$ at some point $C_i$. Along the segment $B_iC_i$ we use a point $C_i'$ a fixed distance $h$ away from $B_i$. For the construction of the polygon we use the chain $A_i$,$B_i$,$C_i'$,$D_i$ and connect $D_i$ with $A_{i+1}$ and also finally $D_n$ with $A_1$. There is a door $d_i$ for each pair of points $B_i$ and $D_i$. This door can be closed within $a_i$ time steps and saves a region of area $\frac{a_i \cdot h}{4}$ proportional to $a_i$.

Now we choose the speed $v$ so that $v(t+0.5) = r$ holds. This means that after $t+0.5$ time steps an intruder can reach any $C_i'$. Additionally, we have to take care that after $t$ time steps, none of the entry points $B_i$ closest to $s$ can be reached by an intruder. Therefore, we choose $r$ large enough so that $vt < x_i = \sqrt{r^2 - \left(\frac{a_i}{2}\right)^2}$ holds. Substituting $v$ by $\frac{r}{(t+0.5)}$ gives

$$\left(\frac{a_i}{2}\right)^2 < \left(1 - \frac{t^2}{(t+0.5)^2}\right) r^2$$

which can be fulfilled for any $a_i$.

Altogether, after $t+0.5$ time steps any entrance point $B_i$ could be passed but after $t$ time step none of the entrance points have been visited. If we would like to save the maximum area from the intruders, we have to compute a maximum sum of values $a_i$ not exceeding $t+0.5$ but beeing as close as possible to $t$. This gives a solution for the subset-sum problem.

Conversely, for a solution of the subset-sum problem we obtain the maximum sum of values $a_i$ smaller than $t$, thus we save a maximum area if we close the corresponding doors successively.

□

**Exercise 1** *Make use of the* `GeoGebra`*-tool and compute an interactive construction of the above reduction for a finite set of elements $a_1, a_2, \ldots, a_n$. Give examples $a_1, a_2, \ldots, a_n$ and $t$ where the subset-sum question is answered with Yes and with No, respectively.*

### 1.1.2 Catching an evader in a grid world

The next example considers a discrete problem variant and a different intention.

There is some evader that can move in a rectangular grid world and a guard would like to enclose and catch the evader. For the enclosement and for catching the evader the guard can block and check cells of the grid while the evader is moving. The evader starts at a single cell and can visit the cells of the 4Neighborship in one step or could stay in its current cell. Simultaneously, the guard can block or check $k$ of the cells. Any cell that was blocked cannot be entered by the evader anymore. In order to synchronize the movements we first let the evader move and then the guard can choose its $k$ blocking and checking cells; see Figure 1.2.

We do not have an idea of the strategy of the evader. Therefore, simply all possible locations of the single evader will be considered and marked by red circles. The blocked cells and checked cells are marked by black circles. We would like to find an efficient strategy for the guard. That is, we would like to enclose the evader as fast as possible. After that the guard simply checks all the remaining red cells. The game ends when there is no single red cell any more.

*Evader-Enclosement (Grid-Graph):*
**Instance:** A rectangular grid, a start vertex $s$ of the evader and $k$ protecting guards per time step.
**Output:** Compute a protection strategy that encloses the evader (and finally find the evader).

We first discuss failure and correctness for different values of $k$, we will see that some machinery is necessary.

**Lemma 2** *Catching an evader in a grid world by setting $k = 1$ blocking cells after each movement of the evader cannot succeed in general.*

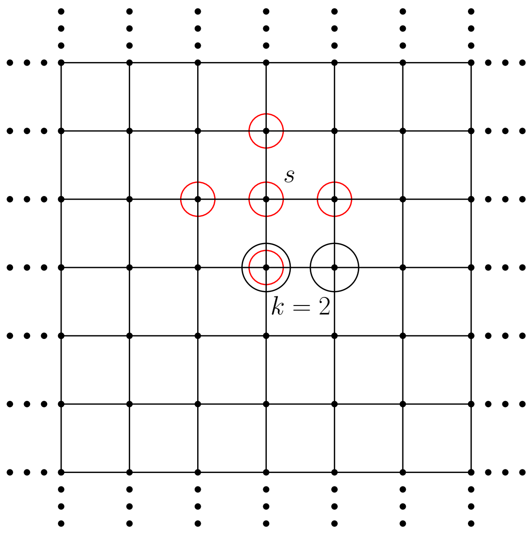


Figure 1.2: The evader starts at $s$. In the first step he can move to the adjacent vertices of the grid. For $k = 2$ the guard blocks and checks two cells afterwards. Now the evader can move on.

**Proof.** We show that the evader always has an escape strategy. Or the other way round, a guard can never cancel out all cells marked red.

For a formal proof we show that it is already impossible to catch the evader in a single quadrant of cells. The evader starts at the source of the quadrant, also called root in the following; see Figure 1.3. For any step $l \geq 1$ let $D_l$ denote the set of vertices that are distance $l$ away from the root vertex. This means, that these cells lie on the $l$-th diagonal of the quadrant.

Let $r_l$ denote the number of blocked cells in $D_{l+1}$, $D_{l+2}$,... at the end of the $l$-th step. The blocked cells in $D_{l+1}$, $D_{l+2}$,... can be considered to build a reserve for the future. Let $B_l \subseteq D_l$ denote the number of red cells at the end of the $l$-th step distance $l$ away from the root. We have $r_0 = 0$.

By induction we show that $B_l \geq 1 + r_l$ holds for all $l \geq 1$. In the first step $l = 1$ the evader can move to both vertices of $D_1$ and the guard can block a single cell, either in $D_t$ for $t = 0, 1$ or in $D_t$ for $t > 1$. The latter one means $r_1 = 1$. In any case we have $B_1 \geq 1 + r_1$.

Let us assume that the statement holds after $l \geq 1$ steps. We can assume $B_l \geq 1 + r_l$. This means that in step $l$ the evader can visit $r_l + 1 + 1$ cells in diagonal $D_{l+1}$. But some of them might already be blocked by the reserve and another one can be blocked by the guard in this step. Let us assume that there are $x \leq r_l$ blocked cells in $D_{l+1}$.

This means that at first (the evader moves) we have $B_{l+1} \geq 1 + 1 + r_l - x$ red cells in $D_{l+1}$. Now the guard can block another cell in step $l$ also. Either in $D_t$ for $t \leq l$, in $D_t$ for $t = l + 1$, or in $D_t$ for $t > l + 1$.

In the latter case we have $r_{l+1} = 1 + r_l - x$ and in the former cases we have $r_{l+1} = r_l - x$. Now the guard can only reduce $B_{l+1}$ by blocking a vertex in $D_{l+1}$. In any case we have $B_{l+1} \geq 1 + r_{l+1}$ which gives the conclusion. □

Note that for the pure enclosement of the evader it seems reasonable that in step $l$ the guard should never block a cell that was marked by a red circle in one of the previous rounds $< l$.

**Exercise 2** *Give a formal argument for the fact that a fast guard strategy should first* enclose *the evader and then check the remaining cells. This means that in step $l$ the guard strategy will not choose a red marked vertex that was marked in a step $< l$ until the evader cannot move anymore.*

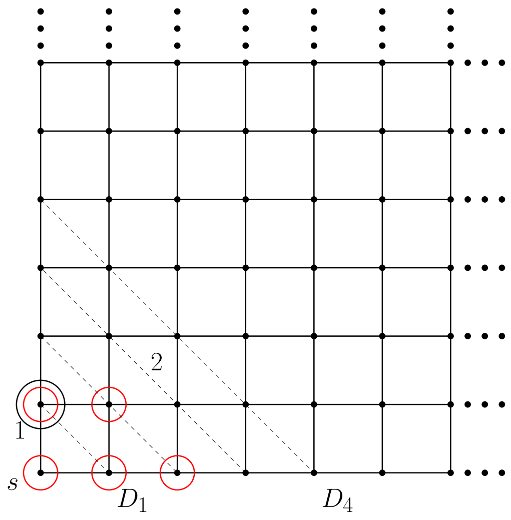


Figure 1.3: The situation for $k = 1$ after step 2. We have $r_2 = 1$ and $B_2 = 1 + r_2 = 2$.

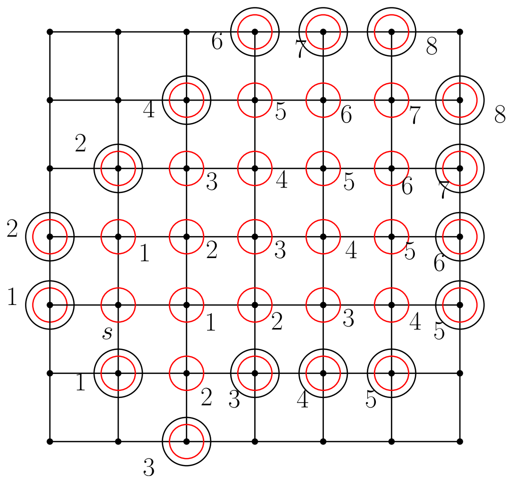


Figure 1.4: For $k = 2$ there is a guard strategy that encloses the evader after 8 steps. Now the evader can be catched by checking the remaining 18 cells.

Fortunately for $k = 2$ there is a successful strategy. We can just give the sequence of steps for the enclosement as presented in Figure 1.4. After that the remaining cells can be cleaned.

**Lemma 3** *There is a successful guard strategy for* $k = 2$. *The evader can be enclosed after* 8 *steps. With 9 additional steps the evader will be catched.*

**Proof.** By construction; see Figure 1.4. □

The above situation can also be considered as the problem of containing a spreading fire. We slightly change the rules of the game. Let us assume that the firemen first blocks $k$ cells and then the fire can spread from the source $s$ to its neighboring cells. A burning cell can never be blocked any more.

In this sense and for $k = 2$ a firemen can enclose a spreading fire in 8 steps as depicted in Figure 1.4. Here only 18 vertices get burned. We give a formal proof that this is optimal.

**Lemma 4** *For the outbreak of a fire on a single source in a grid and the usage of two firefighters* *per time step any successul strategy encloses an area of at least 18 burning vertices. This bound is tight.*

**Proof.** For the proof of the statement we make use of an Integer Linear Program as follows. We already know that it is possible to enclose 18 burning cells by the strategy given in Figure 1.4 in 8 time steps. Therefore we make use of a fixed constant $T > 8$ for the number of time steps. Additionally, we use a grid of size $l \times l$ for another fixed constant $l$.

Let $L = \{(x,y) | |x| \leq l \text{ and } |y| \leq l\}$ and $0 \leq t \leq T$. For $v \in L$ let $N(v)$ denote its 4Neighborship in $L$. For the Integer Program we use the following variables for $v \in L$.

$$b_{v,t} = \begin{cases} 1 & : \text{ vertex } v \in L \text{ burns before or at time } t \\ 0 & : \text{ otherwise} \end{cases}$$

$$d_{v,t} = \begin{cases} 1 & : \text{ vertex } v \in L \text{ is defended before or at time } t \\ 0 & : \text{ otherwise} \end{cases}$$

We would like to minimize the number of vertices that become burned.

$$\begin{array}{llcll}
\text{Minimize} & \sum_{v \in L} b_{v,T} & & & \\
\text{so that} & b_{v,t} + d_{v,t} - b_{w,t-1} & \geq & 0 & : \forall v \in L, v \in N(w), 1 \leq t \leq T \\
& b_{v,t} + d_{v,t} & \leq & 1 & : \forall v \in L, 1 \leq t \leq T \\
& b_{v,t} - b_{v,t-1} & \geq & 0 & : \forall v \in L, 1 \leq t \leq T \\
& d_{v,t} - d_{v,t-1} & \geq & 0 & : \forall v \in L, 1 \leq t \leq T \\
& \sum_{v \in L}(d_{v,t} - d_{v,t-1}) & \geq & 2 & : \forall 1 \leq t \leq T \\
& b_{v,0} & = & 1 & : v \in L \text{ is the origin } (0,0) \\
& b_{v,0} & = & 0 & : v \in L \text{ is not the origin } (0,0) \\
& d_{v,0} & = & 0 & : \forall v \in L \\
& d_{v,t}, b_{v,t} & \in & \{0,1\} & : \forall v \in L, 1 \leq t \leq T
\end{array}$$

The first inequality triggers the spread of the fire. The neighbor of a burning vertex is either blocked or it burns in the next step. Note that with this condition any vertex can also burn accidently. But this will not happen in the optimal solution. The next inequality prevents a defended vertex from burning and a burning vertex from beeing defended. The next two inequalities take care that the status of the vertices remains the same after burning or beeing defended. The sum condition refers to the number $k = 2$ in any step. Initially only one vertex is burning and no vertex is defended. Finally, the program gets binary.

The corresponding program was run for example by Develin and Hartke [] and for $l = 6$ and $T = 9$ and the origin $(0,0)$ the solution of Figure 1.4 was achieved. □

The above proof is an example where we make use of a computer aided process for the conclusion of a theoretical proof. With similar arguments it is also possible to show that 18 vertices cannot be enclosed within less than 8 steps.

### 1.1.3 Enclosing a fire by a single circle

In the last section we have seen that catching an evader and the enclosement of a fire are very similar problems which can be formulated in a similar model. Let us consider the fire containment problem in the following continuous setting.

The fire is already burning for a while and the fire spreads into any direction with the same unit speed 1. There is a single firefighter which can build a firebreak with speed $v > 1$. Geometrically

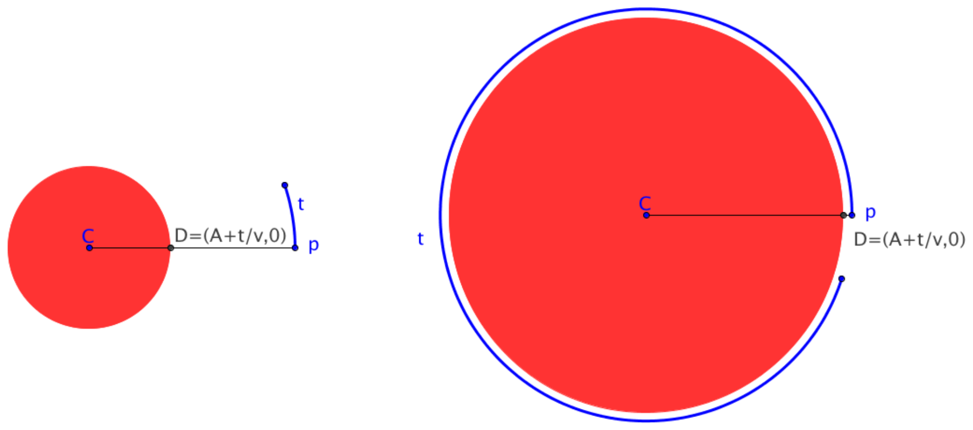


Figure 1.5: The firefighter builds a firebreak of length $t$ with speed $v > 1$ and the fire expands from the source with speed 1. The circular strategy is successful here.

we can assume that in the beginning of the game the fire is a circle of radius $A$ from a source $C$ and the circle grows with speed 1 while the firefighter builds a path $t$ with speed $v > 1$; see Figure 1.5. We call this the *geometric firefighter problem.*

*Geometric Firefigther Problem*
**Instance:** A circle with center $C$ of radius $A$ that grows with unit speed. An agent who builds a firebreak with speed $v > 1$
**Output:** Compute a strategy that finally fully enclose the spreading fire.

A simple question is: How fast must the firefighter be in order to enclose the fire by a single circular loop around the source of the fire? Let us assume that we can choose an arbitrary starting point $p$ for the firefigther.

**Lemma 5** *Enclosing a fire of extension $A$ with a single circular loop around the source of the fire is possible, if and only if the speed $v$ of the firefigther is larger than $2\pi$.*

**Proof.** We assume that the fire source is given by the origin and we choose a point $p = (A+x, 0)$ away from the fire. A single circular loop around the origin has length $2\pi(A + x)$ and takes $\frac{2\pi(A+x)}{v}$ time. We can enclose the expanding circle if this is smaller than or equal to $x$. This gives $\frac{2\pi A}{x} + 2\pi \leq v$. For $v > 2\pi$ there will always be an $x$ so that the inequality holds. For $v \leq 2\pi$ we can never fulfill the required inequality. □

**Exercise 3** *Consider the above circular strategy. Assume that it is allowed to move along an arbitrary circle (around an arbitrary center). Is it possible to improve the above bound? I.e., is the above bound tight for arbitrary circles or can we always succeed with a circle and speed less than $2\pi$?*

### 1.1.4 Simulation and conjecture for a discrete spiral strategy

Finally, we would like to motivate that in some settings it is helpful to make some simulation. We would like to enclose a spreading contamination in a grid world and apply a strategy that always moves close to the boundary of the contamination. The agent has to move from one cell to the other and can build a wall as in the previous setting. The agent either moves clockwise or counterclockwise around the starting contamination. We assume that the contamination is a grid square in the beginning.

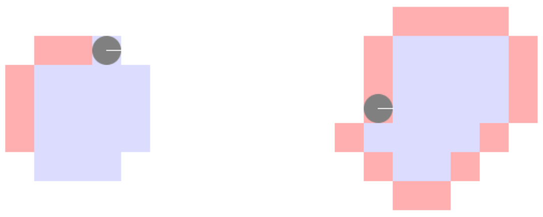


Figure 1.6: The fire spreads after 30 steps and the agent build a wall cell within 5 time steps. The lefthandside figure shows the situation after 30 time steps. The contaminations is enclosed in a single loop after a few more steps.

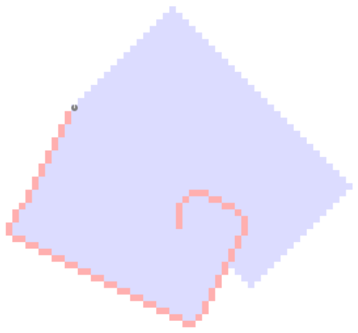


Figure 1.7: The fire spreads after 30 steps and the agent build a wall cell within 15 time steps. After a while it seems that the spiral strategy will not succeed.

**Parameters:** Moving from one cell to the other takes one time step. Building a wall element and cleaning a cell takes $b$ time steps. After $n$ time steps the contamination *spreads*. This means that the 4Neighborship of any contamintated cell becomes contaminated but cells that contain a wall element cannot be contaminated anymore.

*Discrete Firefigther Problem*
**Instance:** A grid contamination some size $B$ that spreads to its 4Neighborship after $n$ time steps. An agent who cleans a cell, builds a wall and leaves the celle within $b$ time steps.
**Output:** Compute a strategy that finally fully enclose the spreading fire.

Figure 1.6 shows an example for a $3 \times 3$ starting contamination with a single agent, $b = 5$ and $n = 30$ after exactly 30 time steps and when the task is finished. Figure 1.7 shows the case where $n = 30$ and $b = 15$, the spiral strategy does not succeed.

Experiments from the thesis of Smiegilski [] give the following conjecture.

**Conjecture 1** *For a grid fire thats spreads after $n$ time steps and an agent that builds a wall within $b$ time steps, the spiral strategy only succeeds if $b < \frac{n-1}{2}$ holds.*

Chapter 2

# Discrete Scenarios for Contaminations

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 $p$ guards 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.

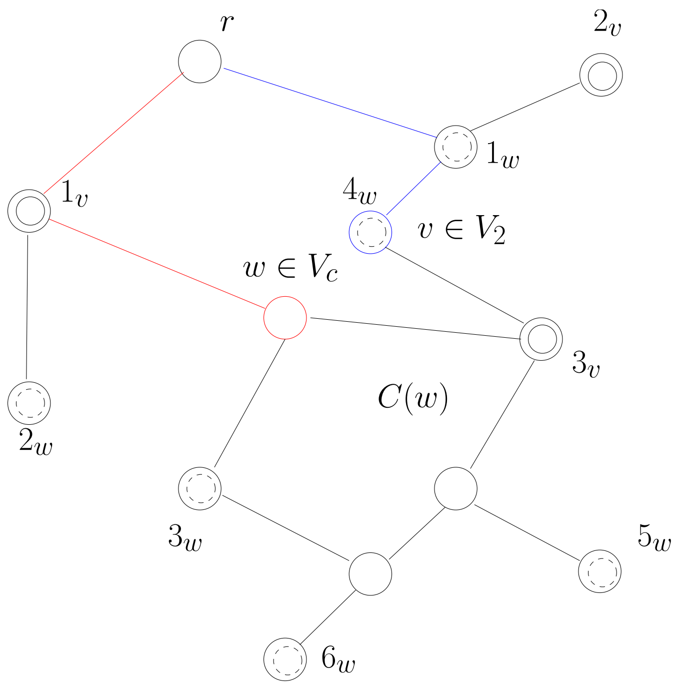


Figure 2.1: Protecting a vertex $v \in V_2$ and $w \in V_c$. The blue and red path denotes the shortest path. The labels $i_v$ and $j_w$ denote the vertices that will be protected at step $i$ and $j$ until $v$ and $w$ are finally enclosed, respectively.

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

**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 a 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 $\leq 2$ 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 let $r$ be of degree 2, assume that the statement is false and let $(G, r, 1)$ be 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 $\geq 2$ 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

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

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 seems 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

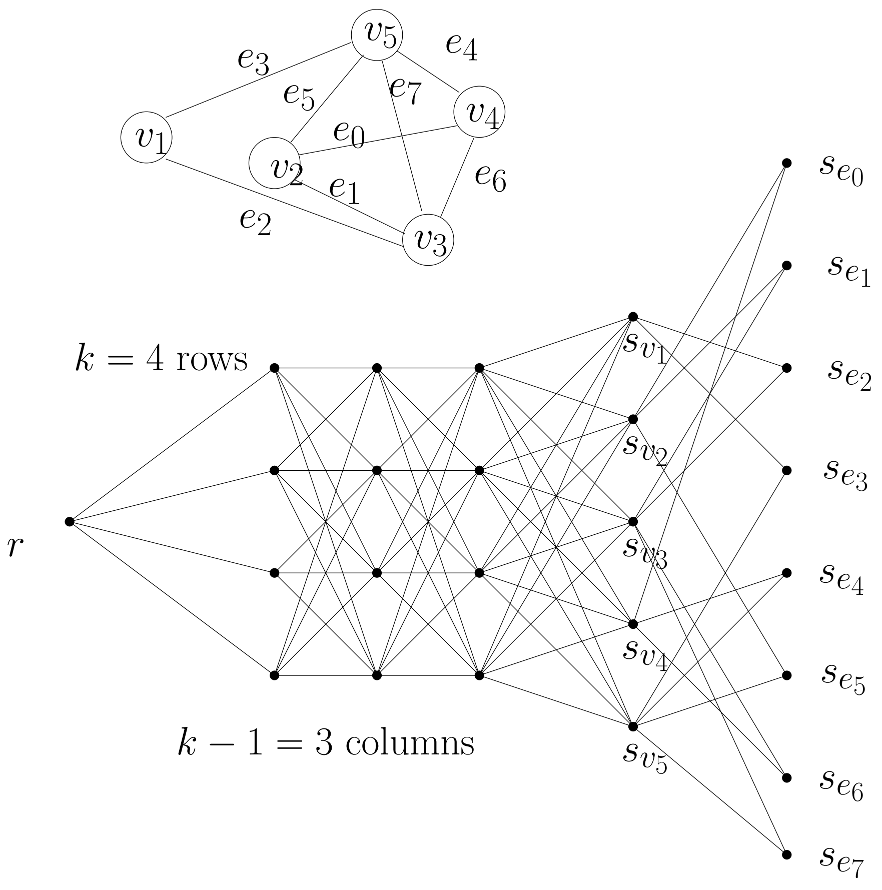


Figure 2.2: Protecting the vertices of the $k$-Clique in $k$ steps and an additional vertex $s_{e_j}$ gives $k' = k + \binom{k}{2} + 1$ protected vertices in total.

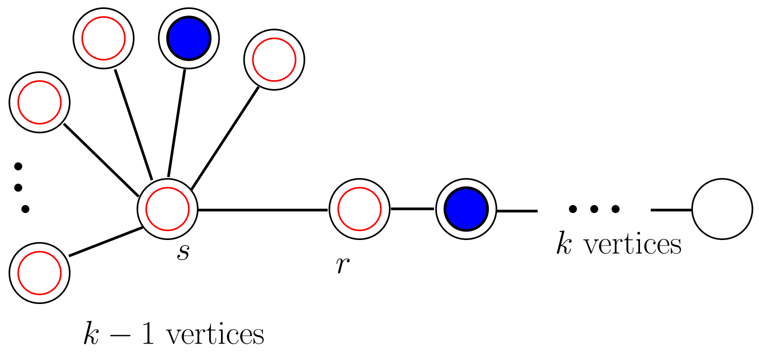


Figure 2.3: The greedy algorithm does not protect the vertex $s$ first and only saves $k+1$ vertices.

means saving at least as many vertices as the optimal move in the same step. For an optimal strategy $\text{opt}$ let $\text{opt}_A$ (greedy is not worse) denote the first category of safe vertices and let $\text{opt}_B$ (greedy is worse) denote the second category of safe vertices. Let $S_G$ denote the number of vertices saved by $S$. 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.

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 giving a somewhat efficient algorithm.

For this purpose 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 problem 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 it 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

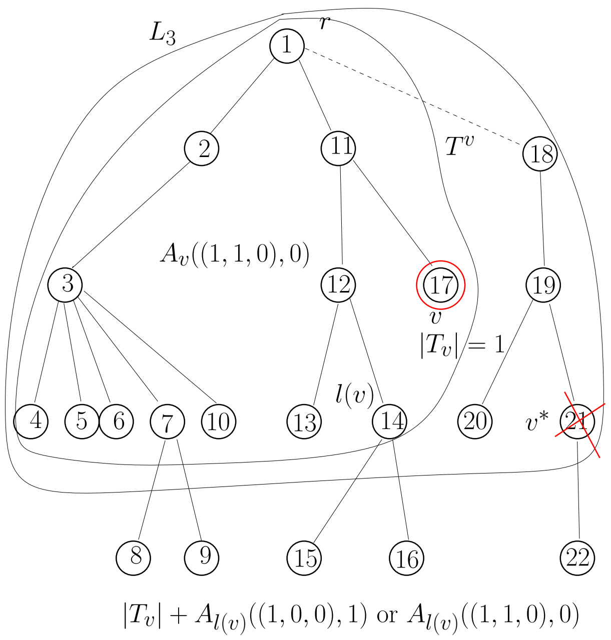


Figure 2.4: The pre-order of a tree $T$, the set $L_k$ of vertices of depth $\leq k$. Computing $A_v((1,1,0),0)$ means that we are searching for a strategy in the tree $T^v$ that sets guards in the first and second depth and if a vertex is protected along the path from $r$ to $v$, then its depth is greater 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.

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 $v \in L_k$, we consider the subtree $T^v$ of $T$ with vertices from $L_k$ and pre-order at most 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 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

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

where $[\phi]$ equals 1 if $\phi$ 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 a tree of size $n$.*

**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 turn an optimal strategy makes use of less than $\sqrt{2n}$ guards. Thus we set $k \leq \sqrt{2n}$ which gives the bound. □

### 2.2.3 Capture of an Intruder by moving agents

Up to now we have considered stationary guards that check and block vertices of the graph or tree. Many other variants are known from the literature. We discuss the following version. A set of $k$ agents starts at a homebase vertex $b$ in a given tree $T = (V, E)$ and there is some intruder somewhere in $T$. The following problem definition and its solution goes back to Barrière et al. [].

The intention is to *clear* all *edges* of the tree. A *contiguous search strategy* can perform one of the following two operations in one search time step:

1. Place a team of $p$ guards on a vertex.
2. Move a team of $m$ guards along an edge.

Additionally, the set of all *cleared* edges $E_i$ after step $i$ has to be connected for any $i$.

In one time step a set of agents located at vertex $u$ can move along an edge $e = (v, u)$ from $v$ to $u$. There is an integer edge weight $w(e) \geq 1$ that says how many agents have to move along $e$ so that the intruder cannot cross the edge from $u$ to $v$ in this step. In this sense the agents *clear* the link in this case. Intuitively, we can think of a large corridor and cleaning the corridor from $v$ to $u$ requires at least $w(e)$ agents. Furthermore, we have integer weights $w(v)$ for the vertices. Locally, a clear link $e = (v, u)$ is preserved from recontamination, if either the vertices $v$ and $u$ are guarded by $w(v)$ and $w(u)$ agents, respectively or all other edges incident to $e$ are also clear.

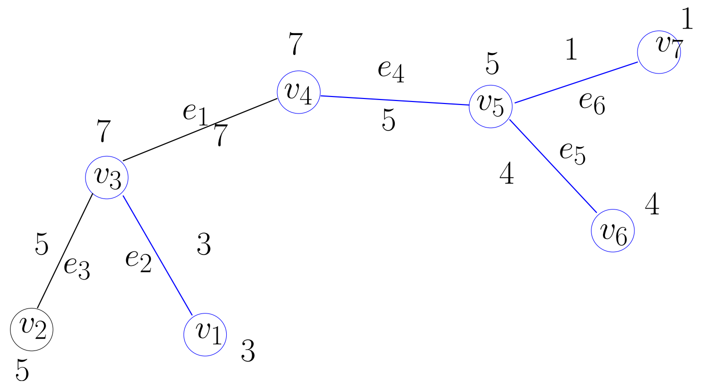


Figure 2.5: A given weighted tree $T = (V, E)$. A successful contiguous strategy can start with 10 agents at $v_1$, also $\mathrm{cs}(T) = 10$ holds. The size of the frontiers of $X_1 = \{e_4, e_5, e_6\}$ and $X_2 = \{e_2\}$ is $w(X_1) = 7$ and $w(X_2) = 10$, respectively.

We assume that guarding a vertex is at least as hard as the maximum number of searchers required for clearing an incident edge. Thus, we require $w(v) \geq w(e)$ for any $e = (v, u) \in E$. So for any vertex $v \in V$ the weight could for example be given by $w(v) := \max_{e=(u,v)\in E} w(e)$. An example for a weighted tree is given in Figure 2.5.

In any time step (after a step of the search strategy) an edge $e$ becomes recontaminated, if there is a path from a contaminated edge $e'$ to edge $e$ that is not blocked by agents on the vertices. This means that a corresponding intruder has infinite speed for crossing an arbitrary number of edges. We consider contiguous search strategies as described above and ask for the minimum number of guards, $\mathrm{cs}(T)$, that suffices to finally clear all links of the tree $T$.

A contiguous strategy for Figure 2.5 can start with 10 agents in vertex $v_7$, clears edge $e_6$. Leaves 6 agents at $v_5$ and clear $e_5$ with 4 agents. Moves back with 4 agents to $v_5$ and clears edge $e_4$ and edge $e_7$ with 10 agents. Now, we leave 7 agents at $v_3$, clear $e_2$ by 3 agents, move back with three agents and finally clear the last edge $e_3$. Note that in the second last step, if we leave less than 7 agents at $v_3$, the edge $e_3$ is contaminated and can recontaminate the full tree.

We would like to consider such monotone strategies. That is, a link that has already been cleaned at some step $t$ should not be contaminated (or visited by the intruder) later again. This means that some of the agents will become stationary for a period of time in order to block the movement of the intruder (or the recontamination of links). Additionally, we can compute optimal monotone strategies efficiently as we will see below. First, we show that monotone strategies always exist.

## 2.2.4 Existance of monotone strategies

One main structural result is, that for a tree $T$ for the computation of $\mathrm{cs}(T)$ it suffices to consider a montone strategy, where all agents start at the same homebase vertex $b$.

**Theorem 17** *For any weighted tree $T$ there is a monotone contiguous search strategy with $cs(T)$ agents where all agents initially start at the same vertex $b$.*

For the proof we require some notations. Let $T = (V, E)$. For a subset $X \subseteq E$ we denote all vertices that have a vertex incident to $X$ and $E \setminus X$ as the *boundary vertices* $\delta(X)$. If $X_i$ denotes the set of clear links after time step $i$ of a strategy then at least

$$w(X_i) := \sum_{v\in\delta(X_i)} w(v) \tag{2.1}$$

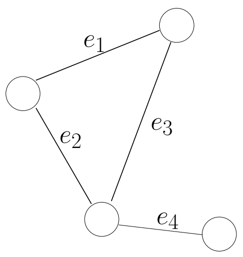


Figure 2.6: Crusades are only defined by subsets of edges of a graph. For the given graph $G$ there is a connected crusade $(\emptyset, \{e_1\}, \{e_1, e_2\}, \{e_2\}, \{e_2, e_3\}, \{e_1, e_2, e_3\}, \{e_3, e_4\}, \{e_1, e_3, e_4\}, \{e_1, e_2, e_3, e_4\})$ and a pogressive connected crusade is $(\emptyset, \{e_1\}, \{e_1, e_2\}, \{e_1, e_2, e_3\}, \{e_1, e_2, e_3, e_4\})$.

guards have been used. (Note that the contamination threatens not only the directly adjacent links but also any fully non-protected path!)

In Figure 2.5 we have $w(\{e_4, e_5, e_6\}) = 7$ and $w(\{e_2\}) = 10$.

For $G = (V, E)$ a sequence $(X_0, X_1, \ldots, X_m)$ of subsets $X_i \subseteq E$ is called a *crusade*, if $X_0 = \emptyset$ and $X_m = E$ and $|X_i \setminus X_{i-1}| \leq 1$ for $1 \leq i \leq m$. The *frontier* of a crusade $(X_0, X_1, \ldots, X_m)$ is given by $\max_{1 \leq i \leq m} w(X_i)$ as defined in Equation 2.1 above, which is the minimum number of guards required for defending any $X_i$. A crusade is *progressive* if $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_m$ and $|X_i \setminus X_{i-1}| = 1$ for $1 \leq i \leq m$. The crusade is *connected*, if $X_i$ is connected for $1 \leq i \leq m$.

Note that the definition of crusades holds for arbitrary graphs $G$. See Figure 2.6 for an example.

There will be a progressive connected crusade that will finally describe a strategy. In the very beginning it is allowed to set agents on the vertices in many non-crusading steps. Then the crusade starts to clean the links. But keep in mind that the above (connected) crusades are only defined over sets of links up to now and do not directly represent a strategy.

**Exercise 11** *Define a crusade $C = (X_0, X_1, \ldots, X_m)$ of a tree $T$ that is not related to a strategy.*

It is easy to see, that for $\mathrm{cs}(T) \leq k$ there should be a connected crusade of frontier $\leq k$ in $T$. For a given contiguous strategy $S$ let $C = (X_0, X_1, \ldots, X_m)$ denote the sequence of clear links after each search step $i$. In any search step we can clear at most one additional edge, which means $|X_i \setminus X_{i-1}| \leq 1$ for $1 \leq i \leq m$. Since $X_i$ is connected (definition of contiguous) and is no further destructed after search step $i$, we have $w(X_i) \leq k$. Of course by construction we have $X_0 = \emptyset$ and $X_m = E$. An example for Figure 2.5 is the above mentioned strategy and the connected crusade $C = (\emptyset, \{e_6\}, \{e_6, e_5\}, \{e_6, e_5, e_4\}, \{e_6, e_5, e_4, e_1\}, \{e_6, e_5, e_4, e_1, e_2\}, \{e_6, e_5, e_4, e_1, e_2, e_3\})$

We conclude:

**Lemma 18** *For $cs(T) \leq k$ there is a connected crusade of frontier at most $k$.*

Now we would like to show that there is also a progressive connected crusade in $T$ of frontier at most $k$. Note that the above connected crusade of Figure 2.5 is indeed progressive.

So starting from the above Lemma from all connected crusades $C = (X_0, X_1, \ldots, X_m)$ of frontier at most $k$ we choose one that satisfies the following properties:

1. $\sum_{i=0}^{m} (w(X_i) + 1)$ is minimum.

2. Amog all crusade satisfying condition 1. choose one with: $\sum_{i=0}^{m} |X_i|$ is minimum.

Obviously, one such crusade has to exist. So we only have to show that the crusade is progressive. This means that we have to show $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_m$ and $|X_i \setminus X_{i-1}| = 1$ for $1 \leq i \leq m$.

Let us first assume that $|X_i \setminus X_{i-1}| = 0$ holds, which means that $X_i \subseteq X_{i-1}$. Therefore, we consider the connected crusade

$$C' = (X_0, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m)$$

of frontier at most $k$ which contradicts condition 1. Note that $|X_{i+1} \setminus X_{i-1}| \leq 1$ can be concluded from $|X_{i+1} \setminus X_i| \leq 1$ and $X_i \subseteq X_{i-1}$.

Thus, we can assume that $|X_i \setminus X_{i-1}| = 1$ holds for $1 \leq i \leq m$. Next, we prove $X_{i-1} \subseteq X_i$ for $1 \leq i \leq m$. We will first conclude that

$$w(X_{i-1} \cup X_i) \geq w(X_i) \tag{2.2}$$

holds for $1 \leq i \leq m$. Otherwise, we make use of the crusade

$$C' = (X_0, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_m) \tag{2.3}$$

of frontier at most $k$ which is a contradiction to condition 1. We know that $X_i$ and $X_{i-1}$ are connected. Thus, the set $X_{i-1} \cup X_i$ is connected since $|X_i \setminus X_{i-1}| = 1$ holds. We can also conclude that $|X_{i+1} \setminus (X_{i-1} \cup X_i)| \leq 1$ holds, since $|X_{i+1} \setminus X_i| = 1$. If $|X_{i+1} \setminus (X_{i-1} \cup X_i)| = 0$ holds we can go back to the first step again.

Now assume that Equation 2.2 holds. For any two arbitrary link sets $A$ and $B$ of $T$ we have $w(A \cup B) + w(A \cap B) \leq w(A) + w(B)$ which is the question of Exercise 12. We conclude $w(X_{i-1} \cap X_i) \leq w(X_i)$ for $1 \leq i \leq m$ from Equation 2.2 and consider

$$C'' = (X_0, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_{i+1}, \ldots, X_m)\,. \tag{2.4}$$

Now, $C''$ has frontier at most $k$.

By the minimality of $C$ w.r.t. condition 2. we conclude that $|X_{i-1} \cap X_i| \geq |X_{i-1}|$ and therefore $X_{i-1} \subseteq X_i$ holds. We conclude $|X_i \setminus (X_i \cap X_{i-1})| = |X_i \setminus X_{i-1}| = 1$ and $|(X_i \cap X_{i-1}) \setminus X_{i-2}| \leq |X_{i-1} \setminus X_{i-2}| \leq 1$.

It remains to show that $X_{i-1} \cap X_i$ is also connected. Assume that this is not the case. Let $\{e\} = X_i \setminus X_{i-1}$ and $W = X_{i-1} \setminus X_i$ and $Z = X_{i-1} \cap X_i$. By assumption $Z = Z' \cup Z''$ where $Z'$ and $Z''$ do not share a vertex. The situation is depicted in Figure 2.7. We have $X_{i-1} = Z' \cup Z'' \cup W$ and $X_i = Z' \cup Z'' \cup \{e\}$. Since $X_i$ and $X_{i-1}$ are both connected, and $\{e\} \notin W$ there is a cycle $Z' \cup Z'' \cup W \cup \{e\}$ in $T$ which contradicts to the assumption that $T$ is a tree, $X_{i-1} \cap X_i$ is also connected. We have proven the following Lemma.

**Lemma 19** *For $cs(T) \leq k$ there is a progressive connected crusade of frontier at most $k$ in $T$.*

**Exercise 12** *For two link (edge) sets $A$ and $B$ in a graph $G$ prove that $w(A \cup B) + w(A \cap B) \leq w(A) + w(B)$ holds.*

Now, for obtaining a monotone contiguous strategy from above progressive connected crusade, we first extend the tree $T$. We replace every link $e$ by two consecutive links $e'$ and $e''$ of the same weight $w(e)$. The newly inserted vertex $v$ has also weight $w(v) := w(e)$. This means that after this transformation any link has at least one vertex of degree 2. Let $T'$ be the outcome of this process. We will use this property for designing a monotone contiguous strategy from a progressive connected crusade. Afterwards we can transform the strategy back for acting on $T$. Obviously, also $\mathrm{cs}(T') \leq \mathrm{cs}(T)$ holds.

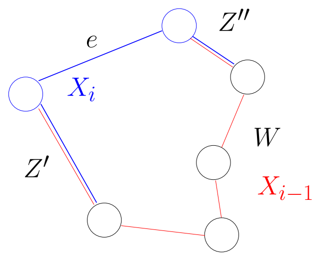


Figure 2.7: Let $W = X_{i-1} \backslash X_i$ and $\{e\} = X_i \backslash X_{i-1}$ and $Z = X_{i-1} \cap X_i$. By assuption $Z = Z' \cup Z''$ where $Z'$ and $Z''$ do not share a vertex. Thus, the cycle $Z' \cup Z'' \cup W \cup \{e\}$ contradicts $T$.

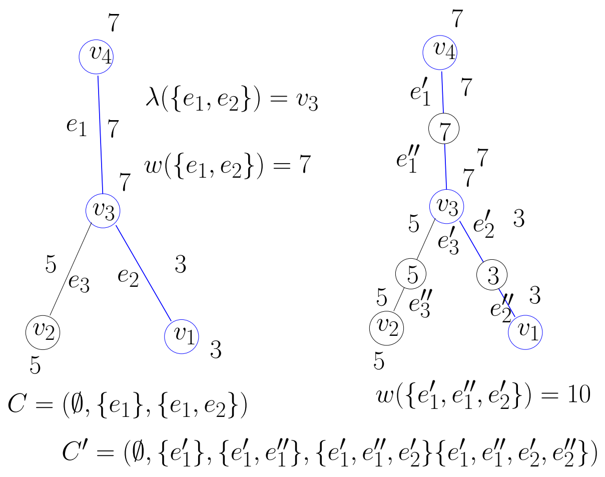

7
$v_4$
$e_1$
7
$\lambda(\{e_1, e_2\}) = v_3$
$w(\{e_1, e_2\}) = 7$
7
$v_3$
5
$e_3$
$e_2$
3
$v_2$
$v_1$
5
3
$C = (\emptyset, \{e_1\}, \{e_1, e_2\})$
7
$v_4$
$e'_1$
7
7
$e''_1$
7
7
$v_3$
5
$e'_3$
$e'_2$
3
5
3
5
$e''_3$
$e''_2$
3
$v_2$
$v_1$
5
$w(\{e'_1, e''_1, e'_2\}) = 10$
$C' = (\emptyset, \{e'_1\}, \{e'_1, e''_1\}, \{e'_1, e''_1, e'_2\}\{e'_1, e''_1, e'_2, e''_2\})$

Figure 2.8: Replacing any link $e$ in the tree $T$ by two consecutive links $e'$ and $e''$ yields the desired correspondance of the frontier and the frontier and the number of agents required for $C = (\emptyset, \{e_1\}, \{e_1, e_2\})$.

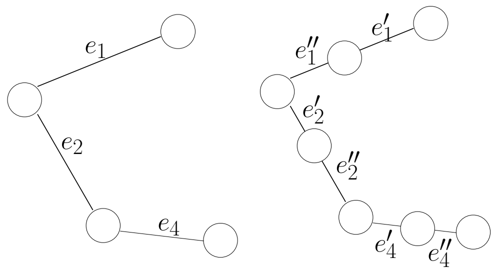


Figure 2.9: Replacing any link $e$ in the tree $T$ by two consecutive links $e'$ and $e''$. Any link in $T'$ has at least one vertex of degree 2.

The main reason for this enlargement is presented in Figure 2.8. Unfortunately, there is no one to one correspondance between the number of agents required for a contiguous strategy and the size of the frontier of connected progressive crusade. For example, in Figure 2.8 we require 10 agents for successively cleaning $C = (\emptyset, \{e_1\}, \{e_1, e_2\})$ but the frontier of $C$ is 7. If we enlarge the tree by artificial edges, we get the required correspondance for $C' = (\emptyset, \{e_1'\}, \{e_1', e_1''\}, \{e_1', e_1'', e_2'\}\{e_1', e_1'', e_2', e_2''\})$. Here $w(\{e_1', e_1'', e_2'\}) = 10$ gives the frontier.

**Lemma 20** *Let $T'$ be a tree so that every link has at least one vertex of degree 2. If there is a progressive connected crusade of frontier $\leq k$ in $T'$, there is a monotone contiguous search strategy using $\leq k$ guards in $T'$ and the guards can be initially placed at a single vertex $v_1$.*

**Proof.** For the progressive connected crusade $C = (X_0, X_1, \ldots, X_m)$ of frontier $\leq k$ and $e_i = (v_i, u_i) := X_i \setminus X_{i-1}$ we construct a strategy that clears the links $e_1, \ldots, e_m$ successively.

In the beginning we set $k$ guards at $v_1$. We have $w(X_1) = w(v_1) + w(u_1) \leq k$ and also $w(e_1) \leq w(u_1)$ and move $w(u_1)$ searchers along $w_1$.

Now by induction let us assume that we have constructed a monotone contiguous strategy for $i-1$ links $e_1, \ldots, e_{i-1}$ without recontaminations. Consider the edge $e_i = (v_i, u_i)$ incident to $X_{i-1}$, say $v_i \in \delta(X_{i-1})$.

If $w(X_{i-1}) + w(u_i) \leq k$ holds, $w(u_i)$ agents can move from $v_i$ along $e_i$ to $u_i$ and clears link $e_i$ and we are done.

Therefore, assume that $w(X_{i-1}) + w(u_i) > k$ holds. In this case not both vertices of $e_i$ can be in $\delta(X_i)$. We have $v_i \in \delta(X_{i-1})$. Assume $v_i \in \delta(X_i)$. We conclude $\deg(v_i) > 2$ and $\deg(u_i) = 2$. This means that $u_i \in \delta(X_i)$ implies that the other link $f_i \neq e_i$ that contains $u_i$ has to be contaminated, and $u_i \notin \delta(X_{i-1})$. Therefore, $w(X_i) = w(X_{i-1}) + w(u_i)$, but this has to be $\leq k$, a contradiction.

For $w(X_{i-1}) + w(u_i) > k$ at most one vertex of $(v_i, u_i)$ is in $\delta(X_i)$ and we consider the corresponding cases; see also Figure 2.10.

1. $v_i \in \delta(X_i), u_i \notin \delta(X_i)$**:** As shown above $\deg(u_i) = 2$ and the other link $f_i \neq e_i$ that contains $u_i$ belongs already to $X_{i-1}$. There are already $w(u_i)$ guards at $u_i$ in step $i-1$ and they can clear the edge from $u_i$ to $v_i$.

2. $v_i \notin \delta(X_i), u_i \notin \delta(X_i)$**:** This means that $e_i$ is the only contaminated edge adjacent to $v_i$ and $u_i$. We can move with $w(v_i)$ searchers from $v_i$ to $u_i$.

3. $v_i \notin \delta(X_i), u_i \in \delta(X_i)$**:** This means that $w(X_i) = w(X_{i-1}) - w(v_i) + w(u_i)$ and we have at least $w(v_i)$ guards at $v_i$. Move all $k - w(X_{i-1})$ free guards to $v_i$. We have $w(v_i) + k - w(X_{i-1}) \geq w(v_i) + w(X_i) - w(X_{i-1}) \geq w(u_i)$ agents at $v_i$ that can clear $e_i$ now.

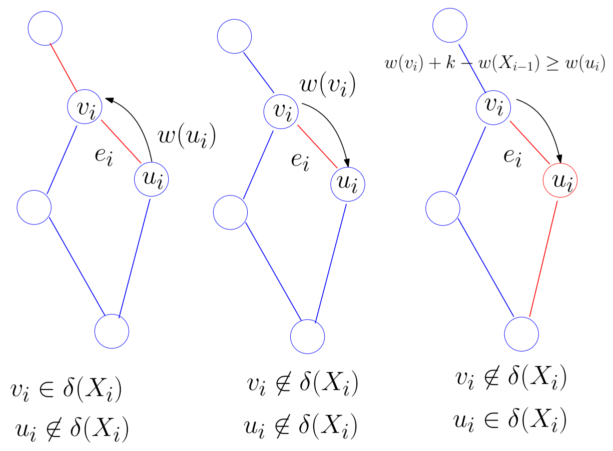


Figure 2.10: Not both vertices $v_i$ and $u_i$ can be in $\delta(X_i)$. Therefore, also edge $e_i$ can be cleaned.

Altogether, there is a successful montone contiguous strategy for $T'$ with $k$ agents adapted from the progressive connected crusade $C = (X_0, X_1, \ldots, X_m)$ of frontier $\leq k$. $\square$

Note, that the above Lemma also holds for graphs with the same properties. The same holds for the following Lemma where we show that we can obtain a strategy for $T$ from the strategy for $T'$.

**Lemma 21** *Any contiguous monotone strategy for $T'$ can be translated to a contiguous monotone strategy for $T$ with the same number $k$ of agents.*

**Proof.** Let $e' = (x, y)$ and $e'' = (y, z)$ be links stemming from the extension of a link $e$. If $q$ guards move from $x$ to $y$ or $z$ to $y$, they stay in there place in $T$. If $q$ guards move from $y$ to $x$ or from $y$ to $z$, they will move from $z$ to $x$ or from $x$ to $z$ in $T$, respectively. $\square$

The other way round, any strategy for $T$ is also a strategy for $T'$.

**Lemma 22** *Any contiguous monotone strategy for $T$ with $k$ agents can be translated to a contiguous monotone strategy for $T'$ with the same number $k$ of agents.*

**Proof.** A move along an edge $e$ in $T$ is splitted into two moves along $e'$ and $e''$ in $T'$. If the move clears $e$, then $q \geq w(e)$ have traversed $e$. From the construction $q$ searchers are also enough for $w(e) = w(e') = w(e'')$ and the weight $w(e)$ of the intermediate vertex. $\square$

We collect our results:

**Proof of Theorem 17:** From Lemma 21 we conclude $\text{cs}(T') \leq \text{cs}(T)$. From Lemma 18 we obtain a connected crusade of frontier $\leq \text{cs}(T)$ in $T'$. From Lemma 19 we conclude that there is a progressive connected crusade of frontier $\leq \text{cs}(T)$ in $T'$. From Lemma 20 we obtain a monotone contiguous search strategy using $\leq \text{cs}(T)$ guards in $T'$ and we can assume that all searchers are initially at a single starting vertex $v_1$. From Lemma 22 we conclude that there is also an optimal monotone contiguous search strategy that starts with all guards in a single vertex.

### 2.2.5 Designing a monotone strategy for unit weights

By Theorem 17 we can start a strategy from a single vertex $v$ and we can consider monotone strategies. Therefore, we design an optimal strategy for any starting vertex $v$ and for the rooted tree $T_v$ we compute the minimum number, $\text{cs}(T_v)$, of agents required for starting in $v$. Finally we have $\text{cs}(T) = \min_{v \in T} \text{cs}(T_v)$.

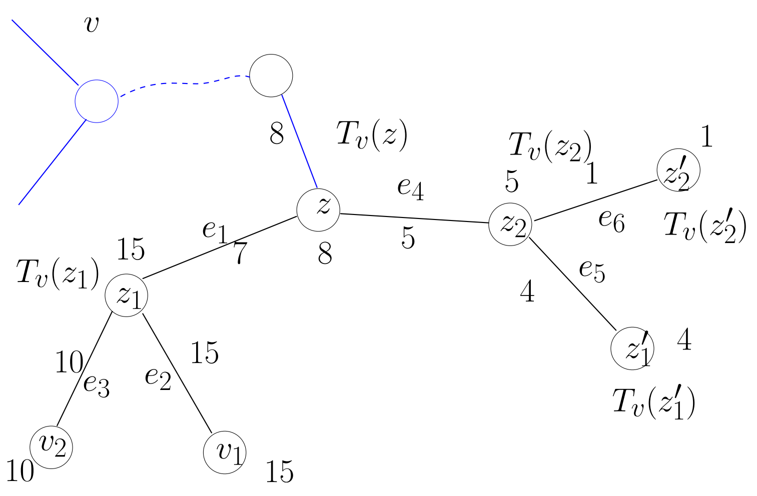


Figure 2.11: The rooted tree $T_v$ has two subtrees $T_v(z_1)$ and $T_v(z_2)$ at vertex $z$. We have $\mathrm{cs}(T_v(z_1)) = 25$ and $\mathrm{cs}(T_v(z_2) = 6$ and $\mathrm{cs}(T_v(z_2)) + w(z) = 14 < 25 = \mathrm{cs}(T_v(z_1))$ which means that $\mathrm{cs}(T_v(z)) = 25$ holds. We leave $w(z)$ agents at $z$ and clean $T_v(z_2)$ first. In $T_v(z_2)$ the same situation occurs, here $\mathrm{cs}(T_v(z_1')) = 4$ and $\mathrm{cs}(T_v(z_2)) = 1$ but $\mathrm{cs}(T_v(z_2')) + w(z_2) = 6 > 4 = \mathrm{cs}(T_v(z_1')$. Therefore we require $\mathrm{cs}(T_v(z_2')) + w(z_2) = 6$ agents, first we clean $T_v(z_2')$ by 1 agent and block $z_2$ by 5 agents. Then we clean $T_v(z_1')$ by 6 agents.

An optimal monotone strategy for computing, $\mathrm{cs}(T_v)$, will also give an ordering of all vertices $z$ of $T_v$, stating which subtree, say $T_v(z)$, of $T_v$ w.r.t. root $v$ is fully cleared first. We can also consider the subtree $T_v(z)$ with root $z$ by its own and ask for $\mathrm{cs}(T_v(z))$ for short and an optimal monotone strategy.

We denote the children of the vertex $z$ of the subtree $T_v(z)$ of $T_v$ by $z_1, \ldots, z_d$ w.r.t. the order $\mathrm{cs}(T_v(z_i)) \geq \mathrm{cs}(T_v(z_{i+1}))$ for $i = 1, \ldots, d-1$. An example is given in Figure 2.11. Now, we can prove the main structural result. Unfortunately, there is a flaw in the proof of Barrière at al. and we can only prove the statement for unit weighted trees. The flaw is precisely marked in the proof below and also shown in Figure 2.12.

**Lemma 23** *Let $z_1, \ldots, z_d$ be the $d \geq 2$ children of a vertex $z$ in $T_v$ and assume that $cs(T_v(z_i)) \geq cs(T_v(z_{i+1}))$ for $i = 1, \ldots, d-1$. We have*

$$cs(T_v(z)) = \max\{cs(T_v(z_1)), cs(T_v(z_2)) + w(z)\} \tag{2.5}$$

*if the tree $T$ is a tree with unit weights.*

**Proof.** We can assume that $\mathrm{cs}(T_v(z)) \geq \mathrm{cs}(T_v(z_1))$ holds because we have to clear $T_v(z_1)$ before clearing $T_v(z)$. If in Equation 2.5 $\mathrm{cs}(T_v(z_1)) \geq \mathrm{cs}(T_v(z_2)) + w(z)$ holds, we can clear $T_v(z)$ by setting $w(z)$ on $z$ and clear all $T_v(z_i)$ by $\mathrm{cs}(T_v(z_1))$ agents but $T_v(z_1)$ last. Altogether, $\mathrm{cs}(T_v(z_1))$ agents are required and they are sufficient.

So let us assume that in Equation 2.5 $\mathrm{cs}(T_v(z_1)) < \mathrm{cs}(T_v(z_2)) + w(z)$ holds. We would like to prove that $\mathrm{cs}(T_v(z_2)) + w(z) - 1$ agents are not sufficient. We consider two cases:

1. **$T_v(z_2)$ is cleared before $T_v(z_1)$:** While $\mathrm{cs}(T_v(z_2))$ agents clear $T_v(z_2)$ there are only $w(z) - 1 = 0$ agents left for blocking a vertex in $T_v(z_1)$. Recontamination!
2. **$T_v(z_1)$ is cleared before $T_v(z_2)$):** While $\mathrm{cs}(T_v(z_1))$ agents clear $T_v(z_1)$ there are no more than $w(z) - 1 = 0$ agents left for blocking a vertex in $T_v(z_2)$ (because $\mathrm{cs}(T_v(z_1)) = \mathrm{cs}(T_v(z_2)))$. Recontamination!

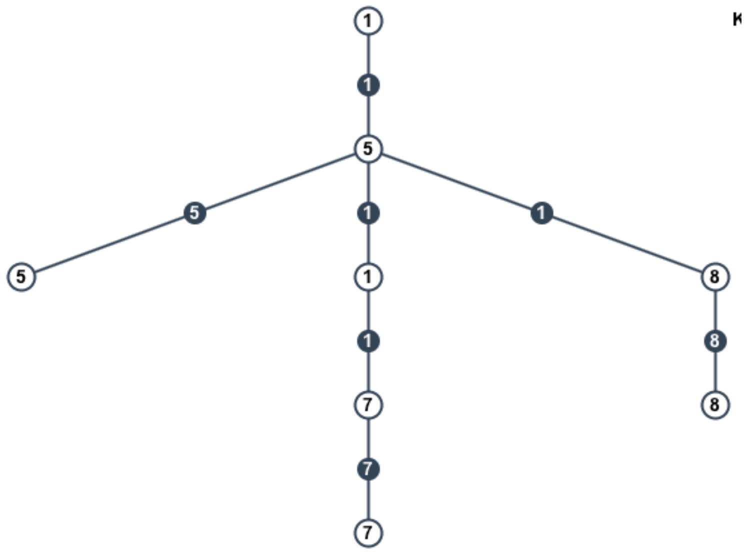


Figure 2.12: The flaw in the prove of Barriére et al. The recursion $\mathrm{cs}(T_v(z)) = \max\{\mathrm{cs}(T_v(z_1)), \mathrm{cs}(T_v(z_2)) + w(z)\}$ does not hold for arbitrary weighted trees.

**General weighted trees:** The above statement does not hold for general weighted trees, because the fact that an agent could only partially decontaminates $T_v(z_2)$ or $T_v(z_1)$ is not taken into account. For example, consider the vertex, say $v$ with weight 5 in the center of Figure 2.12. and let $z_1$, $z_2$, and $z_3$ be the children of $v$ below $v$ from right to left. We have $\max\{\mathrm{cs}(T_v(z_1)), \mathrm{cs}(T_v(z_2)) + w(z)\} = \max\{8, 7+5\} = 12$ but we can recontaminate the subtree by 10 agents only, if we first clean $z_3$, leaving 5 agent at $v$. Then we clean vertex $z_2$ with one agent and leave this agent there. Aftre that we clean $T_x(z_1)$ with the remaining 9 agents, and finally return to $z_3$ for the last part.

So for unit weights we have shown that $\mathrm{cs}(T_v(z_2)) + w(z)$ are required and are also sufficient. For $\mathrm{cs}(T_v(z_1)) < \mathrm{cs}(T_v(z_2)) + w(z)$ we occupy $z$ with $w(z)$ agents and clear all $T_v(z_i)$ by $\mathrm{cs}(T_v(z_2))$ agents but $T_v(z_1)$ last with $\mathrm{cs}(T_v(z_2)) + w(z)$ agents. □

The consequence of the above Lemma is, that we can compute $\mathrm{cs}(T_v)$ in $O(n)$ time by recursively applying Equation 2.5. Alternatively, we can start from the vertices.

**Exercise 13** *Compute cs($T_{v_4}$) for the tree in Figure 2.5 by the above recursive process.*

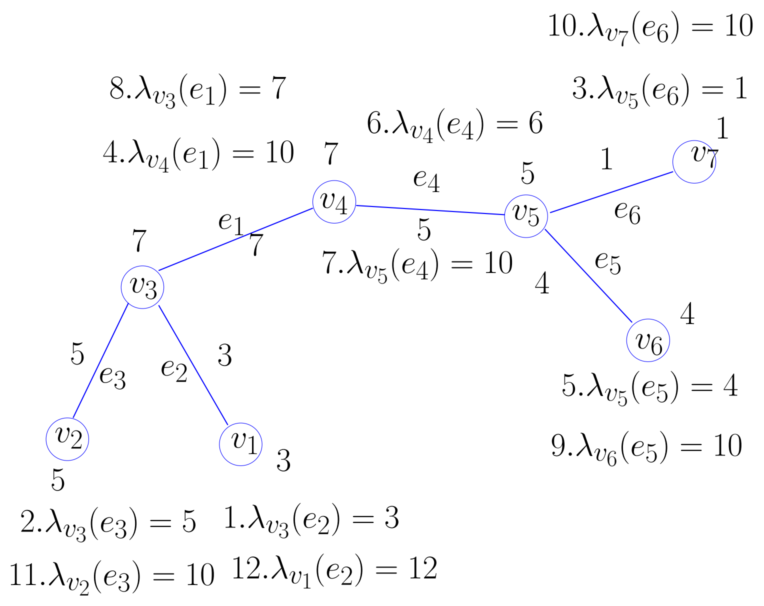
**Corollary 24** *For a unit weighted tree $T$ of size $n$ and for a given starting vertex $v$ we can compute the optimal monotone contiguous strategy starting at $v$ in $O(n)$ time. An overall optimal contiguous strategy can be computed in $O(n^2)$.*

### 2.2.6 Optimal contiguous Intruder Search Strategy for unit weights

We consider a message based algorithm that computes the optimal number of agents required for any starting vertex $v$.

The following local recursive labeling $\lambda_x(e)$ for the links $e = (x, y)$ adjacent to $x$ will be sufficient. Let $e = (x, y)$ be a link incident to $x$.

1. If $y$ is a leaf, set $\lambda_x(e) := w(y)$.



$10.\lambda_{v_7}(e_6) = 10$
$8.\lambda_{v_3}(e_1) = 7$
$3.\lambda_{v_5}(e_6) = 1$
$6.\lambda_{v_4}(e_4) = 6$
$4.\lambda_{v_4}(e_1) = 10$
$7.\lambda_{v_5}(e_4) = 10$
$5.\lambda_{v_5}(e_5) = 4$
$9.\lambda_{v_6}(e_5) = 10$
$2.\lambda_{v_3}(e_3) = 5$
$1.\lambda_{v_3}(e_2) = 3$
$11.\lambda_{v_2}(e_3) = 10$
$12.\lambda_{v_1}(e_2) = 12$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$

Figure 2.13: The message sending algorithm can easily work sequentially.

2. Otherwise, let $d$ be the degree of $y$ and let $x_1, \ldots, x_{d-1}$ be the incident vertices of $y$ different form $x$. Let $\lambda_y(y, x_i) =: l_i$ and $l_i \geq l_{i+1}$. Then,

$$\lambda_x(e) := \max\{l_1, l_2 + w(y)\}.$$

For any link $e = (x, y)$ we will have two labels $\lambda_x(e)$ and $\lambda_y(e)$. By a *messages sending* technique, we can compute the labels $\lambda_x(e)$ and $\lambda_y(e)$ for all edges $e = (x, y)$ in overall linear time. Note that we interpret any link $e = (x, y)$ as undirected, which means that $(x, y) = (y, x) = e$, more formally we could have used a notion $e = \{x, y\}$.

The message sending algorithm works as follows:

1. Start with the leaves and for any leaf $y$ and for $e = (x, y)$ send a message $l = w(y)$ to $x$. After receiving this messages, $x$ sets $\lambda_x(e) := l$.

2. Consider a vertex $y$ of degree $d$ that has received at least $d-1$ messages $l_i$ from the incident vertices $x_1, \ldots, x_{d-1}$ and let $x$ be the remaining incident vertex. Let $l_i \geq l_{i+1}$. Send a message $l = \max\{l_1, l_2 + w(y)\}$ to $x$, after receiving the message $x$, set $\lambda_x((x, y)) := l$.

The above process can be applied sequentially, starting from the leaves. The process will not stop until we have send a message from $x$ to $y$ and $y$ to $x$ along any edge $e = (x, y)$. The process ends and in total $O(n)$ messages have been transmitted. An example is given in Figure 2.13. Keep in mind that setting $\lambda_x(e)$ means that $x$ has received a message from the other vertex $y$ of $e = (x, y)$.

**Lemma 25** *The links of a tree $T$ can be labeled with labels $\lambda_x$ by the above message sending algorithm by $O(n)$ messages in total.*

Finally, we would like to prove that for an edge $e = (x, y)$ the labeling algorithm indeed computes $\text{cs}(T_x(y))$ for the rooted tree $T_x$ and its direct neighbor $y$. Note that we can only proof the result for unit weighted trees.

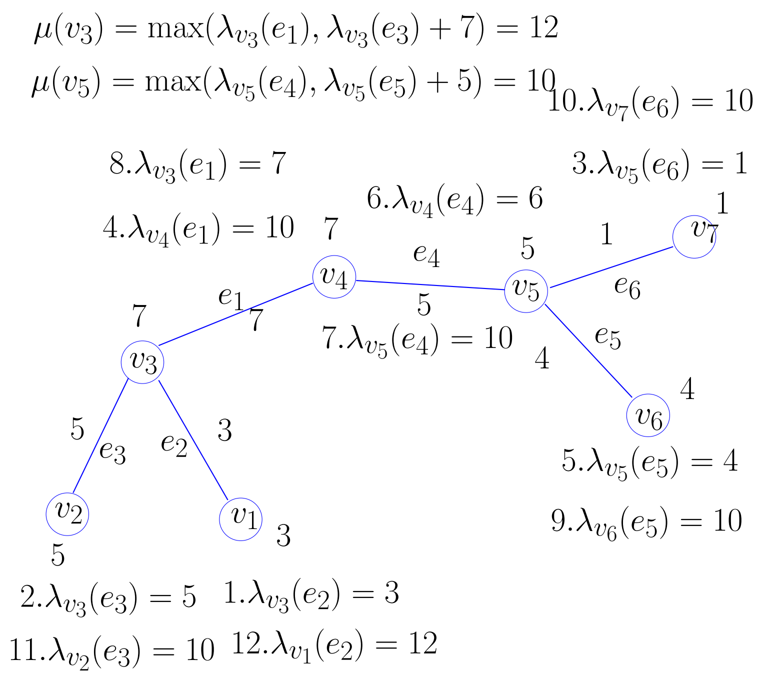

$\mu(v_3) = \max(\lambda_{v_3}(e_1), \lambda_{v_3}(e_3) + 7) = 12$
$\mu(v_5) = \max(\lambda_{v_5}(e_4), \lambda_{v_5}(e_5) + 5) = 10$
$10.\lambda_{v_7}(e_6) = 10$
$8.\lambda_{v_3}(e_1) = 7$
$3.\lambda_{v_5}(e_6) = 1$
$6.\lambda_{v_4}(e_4) = 6$
$4.\lambda_{v_4}(e_1) = 10$
7
$e_4$
5
1
1
$v_7$
$v_4$
$e_1$
$v_5$
$e_6$
7
7
5
$e_5$
$7.\lambda_{v_5}(e_4) = 10$
4
$v_3$
4
$v_6$
5
$e_3$
$e_2$
3
$5.\lambda_{v_5}(e_5) = 4$
$v_2$
$v_1$
3
$9.\lambda_{v_6}(e_5) = 10$
5
$2.\lambda_{v_3}(e_3) = 5$
$1.\lambda_{v_3}(e_2) = 3$
$11.\lambda_{v_2}(e_3) = 10$
$12.\lambda_{v_1}(e_2) = 12$

Figure 2.14: Computing $\mu(v) = \max\{\lambda_v((v, x_1)), \lambda_v((v, x_2)) + w(v)\}$ and the minimal $\min_{v\in V} \mu(v) = \mathrm{cs}(T)$ gives an optimal strategy at least for unit weighted trees.

**Lemma 26** *For a unit weighted tree $T = (V, E)$ and an edge $e = (x, y) \in E$ we have $cs(T_x(y)) = \lambda_x(e)$.*

**Proof.** The proof goes by induction on the height $h(y)$ of $T_x(y)$. If $y$ is a leaf we have $\lambda_x(e) = w(y)$ for $h(y) = 0$. The statement holds.

Assume that the statement holds for $0 \leq h(y) < k$ and consider $h(y) = k$. For edge $e = (x, y)$ let $x_1, \ldots, x_d$ be the $d \geq 1$ be the children of $y$ in $T_x(y)$ and assume that $\lambda_y((y, x_i)) \geq \lambda_y((y, x_{i+1}))$ holds for $i = 1, \ldots, d-1$. We also have $T_y(x_i) = \lambda_y((y, x_i))$ by induction hypothesis and $T_y(x_i) = T_x(x_i)$ by definition. Therefore we also have $\mathrm{cs}(T_x(x_i)) \geq \mathrm{cs}(T_x(x_{i+1}))$ for $i = 1, \ldots, d-1$.

In Lemma 23 the recursion Equation 2.5 for $T_x(y)$ is exactly the same as $\lambda_x((x, y))$ in step 2. of the labeling process 2.2.6. Therefore, we conclude $\mathrm{cs}(T_x(y)) = \lambda_x(y)$. □

Finally, we have to compute the optimal number of agents and also a corresponding strategy. The first part can done as follows. We compute the minimum number of agents, $\mu(v)$, required for starting at a vertex $v$ in the tree $T$.

For this we order all $\lambda_v((v, x_i))$ for all $i = 1, \ldots, d$ incident edges $(v, x_i)$ so that $\lambda_v((v, x_i)) \geq \lambda_v((v, x_{i+1}))$ and compute

$$\mu(v) = \max\{\lambda_v((v, x_1)), \lambda_v((v, x_2)) + w(v)\}\,. \tag{2.6}$$

See for example the computation of $\mu(v_3)$ and $\mu(v_5)$ in Figure 2.14.

Altogether, we have $\mu(v) = \mathrm{cs}(T_v)$ and $\min_{v\in V} \mu(v) = \mathrm{cs}(T)$. For the movements of the agents we choose the vertex $v$ that attains a minimum $\mu(v)$ and apply a strategy as induced by the values $\lambda_y$. We traverse $T_v$ in increasing order of the values $\lambda_y$.

For example, in Figure 2.14 $\mu(v_5) = 10$ gives the minimal number of agents required and we start with 10 agents in $v_4$ w.r.t. decreasing numbers $\lambda_{v_5}$. Thus, first 1 agent moves along $e_6$ and back to $v_5$, then 4 agents move along $e_5$ and back to $v_5$. After that 10 agents move along $e_4$ and so on.

**Theorem 27** *An optimal contiguous strategy for a unit weighted tree $T = (V, E)$ can be computed in $O(n)$ time and space.*

**Proof.** The number of messages required is given by the above considerations. For calculating the messages (and also the values $\mu(x)$) afterwards, we only have to register the largest three entries $\lambda_v(e)$ for any $v$. This can be done successively. For any new message we can adjust these largest three entries in constant time. □

### 2.2.7 Lower and upper bound for the contiguous search

For a given tree, $T_n$, with $n$ nodes we are asking for $\max_n \mathrm{cs}(T_n) =: \mathrm{cs}(n)$. For convenience we consider the unit weighted case, where all weights are equal to 1. We will prove the following Theorem.

**Theorem 28** *For unit weights and for any number of vertices n, we have $\lfloor \log_2 n \rfloor - 1 \leq cs(n) \leq \lfloor \log_2 n \rfloor$.*

We prove each inequality of the Theorem separately by the following lemmata:

**Lemma 29** *For every $n \geq 1$ we find trees $T_n$ with $cs(T_n) \geq \lfloor \log_2 n \rfloor - 1$.*

**Proof.** We consider a rooted tree $T$ with root $r$ and for any vertex $u$ let the *level of* $u$ denote the distance from $r$ to $u$. If $n$ equals $2^k - 1$ we choose a complete binary tree and show that $\mathrm{cs}(T_n) = k - 1 = \log_2(n+1) - 1 \geq \lfloor \log_2 n \rfloor - 1$ agents are required by considering the values $\lambda_v(e)$. See also Figue 2.15.

- We have $\lambda_v((v,u)) = k - i$ and $\lambda_u((v,u)) = k - 1$, for any vertex $u$ of level $i > 0$ and its parent node $v$ w.r.t. $r$. This can be easily seen by induction. The last value stem from the fact that we have to clean a complete tree with $2^{k-1} - 1$ vertices by starting from the root node.
- We have $\mu(u) = k - 1$ for any $u \neq r$ and $\mu(r) = k$, which gives the bound.

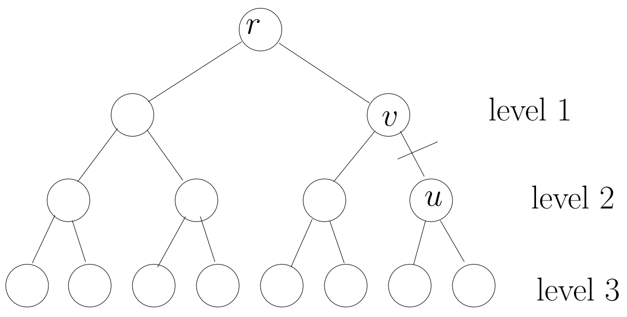
Now, for $n \neq 2^k - 1$ consider the binary representation $n = \sum_{i=1}^{r} 2^{\alpha_i}$ with $\alpha_1 > \alpha_2 > \cdots > \alpha_r$. For example consider $n = 11010$ in binary representation with $\alpha_1 = 4$, $\alpha_2 = 3$, $\alpha_3 = 2$. We build a chain with vertices $x_1, x_2, \ldots, x_r$ and for any $x_i$ we build an edge to a complete binary tree $T_{\alpha_i}$ of size $2^{\alpha_i} - 1$ as depicted in Figue 2.16.

This means that we have $n$ vertices in total. We conclude that $\alpha_1$ agents are required, if $2^{\alpha_1}$ and $2^{\alpha_1 - 1}$ exists in the binary representation. This holds if we start somewhere outside $T_{\alpha_1}$ because we visit the root of $T_{\alpha_1}$ at some point. If we start inside $T_{\alpha_1}$ (for example in a leaf) we require at most $\alpha_1 - 1$ agents for $T_{\alpha_1}$ but at the root node $y_i$ of $T_{\alpha_1}$ we can assume that we have to place an additional agent that blocks the recontamination from $x_1$. For this we assume we assume that we require at least $\alpha_1 - 1 = \lambda_{y_1}((y_1, x_1))$, since the tree for $2^{\alpha_1 - 1}$ exists. Altogether in the above case, we have $2^{\alpha_1} - 1 < n < 2^{\alpha_1 + 1} - 1$ and require $\mathrm{cs}(T_n) = \alpha_1 \geq \log_2(n+1) - 1 \geq$ agents in total which gives the conclusion.

The remaining cases are as follows: Assume that $2^{\alpha_1 - 1}$ does not exist in the binary representation, we have two subcases.

1. If all subtrees from $2^{\alpha_1 - 2}$ to $2^0$ exist, the tree starting at $x_1$ into $x_2$ is a binary tree of depth $\alpha_1 - 1$. Thus, we require again $\alpha_1$ agents for any starting vertex and we can use the simple argumentation above.

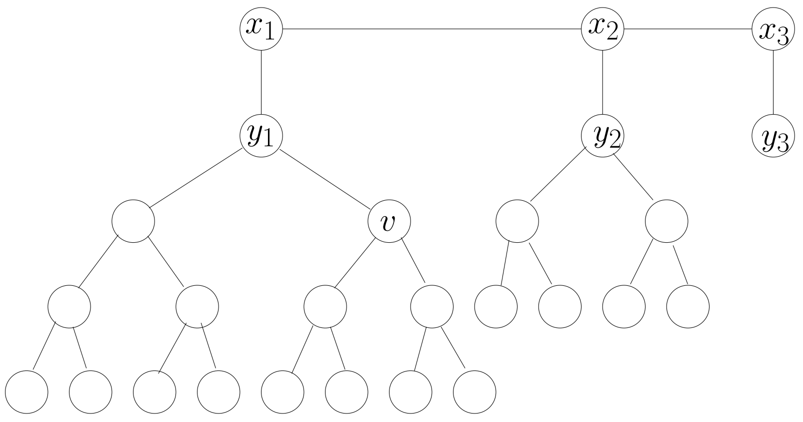
$$k = 4 \text{ and } n = 2^k - 1$$



$$\lambda_v((v,u)) = k - \text{level}(u)$$
$$\lambda_u((v,u)) = k - 1$$
$$\mu(r) = k \text{ and } \mu(u \neq r) = k - 1$$

Figure 2.15: For $k = 4$ and $n = 2^k - 1$ and $T_n$ as the full binary tree, we conclude $\text{cs}(T_n) = k - 1$ which gives the bound.

$$n = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 11010$$



$$\lambda_{y_1}((v, y_1)) = \alpha_1 - 1$$
$$\lambda_{y_1}((x_1, y_1)) = \alpha_2 + 1 = \alpha_1$$

Figure 2.16: A tree $T_n$ with $n = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 26$ vertices, requires $\alpha_1 = 4$ agents.

2. If not all subtrees from $2^{\alpha_1-2}$ to $2^0$ exist, only $\alpha_1-1$ agents are required. But now the value for $n$ is small enough so that we can conclude. $\alpha_1 - 1 \geq \lfloor \log_2(\frac{2}{3}(n+1)) \rfloor \geq \log_2(n+1) - 1$ which gives the bound. This requires the measurement of $\frac{2}{3}(n+1)$ in comparison to $\alpha_1 - 1$ is left as an Exercise.

□

**Exercise 14** *Discuss the remaining case in the above proof. That is $\alpha_2 - 1 < \alpha_1$ and the two cases depicted in the proof.*

On the other hand, we show that $\lfloor \log_2 n \rfloor$ agents are always sufficient.

**Lemma 30** *For every $n \geq 1$ and unit weights, $\lfloor \log_2 n \rfloor$ agents are sufficient for a contiguous search strategy.*

**Proof.** We consider a tree $T_r$ with $n$ vertices and $\mu(r) = \text{cs}(T)$. Now, we simplify this so that it becomes a complete binary tree $T_r'$ w.r.t. $r$ with $\text{cs}(T_r) = \text{cs}(T_r')$ by the following rules. The rules can be applied successively, until none of them is applicable any more. The children/parent relation in the tree is considered w.r.t. $r$.

1. For a node $x$ and its $d > 2$ children $x_1, x_2, \ldots, x_d$ ordered by $\text{cs}(T_r(x_i)) \geq \text{cs}(T_r(x_{i+1}))$ remove all $T_r(x_i)$ for $i > 2$.
2. For a node $x$ with two children $x_1$ and $x_2$ and $\text{cs}(T_r(x_1)) > \text{cs}(T_r(x_2))$, remove $T_r(x_2)$.
3. For a node $x \neq r$ with only one child $x_1$, remove $x$ and connect $x_1$ to the parent of $x$.
4. If there are more than two vertices left, and $r$ has only one child $x_1$, remove $x_1$ and connect the children of $x_1$ to $r$.

First, the number of agents required for $T_r'$ and $T_r$ are the same, because the computation of $\mu(r)$ in $T_r$ makes use of exactly the same values. Note that the weights of the vertices are restricted to one, therefore rule 2. is also correct by $\text{cs}(T_r(x_1)) \geq \text{cs}(T_r(x_2)) + 1$. Cancelling a vertex with one child has no influence.

Second, we show that $T_r'$ is a complete binary tree rooted in $r$. The first rule and the second rule returns a tree that has internal nodes with at most 2 children. Rule three deletes internal nodes with one child except for the root. Rule 4 makes the root have 2 or 0 children.

Thus, we have a binary tree whose internal nodes have degree excactly 2. Finally, we show that the tree is complete. Let $x$ be a node such that the subtree $T_x'$ at $x$ is not complete and there is no other subtree in $T_x'$ with this property. This means that the children $x_1$ and $x_2$ of $x$ in $T_r'$ define complete subtree $T_{x_1}'$ and $T_{x_2}'$ of different size. Thus, rule 2 can be applied which gives a contradiction. □

### 2.2.8 The prize of connectivity

In the previous section we analyzed the contiguous search number for trees and presented a polynomial time algorithm for trees. The key argument was that recontamination does not help for decreasing the search number. The contiguous search idea is mainly based on the fact that searchers should not jump.

In general in the non-continuous setting this is in some sense allowed. More precisely, we extend the rules defined in the beginning of Section 2.2.3. We allow that some of the agents can be retracted from somewhere and placed somewhere else.

1. Place a team of $p$ guards on a vertex.
2. Move a team of $m$ guards along an edge.
3. Remove a team of $r$ guards from a vertex.

We consider the unit-weighted case in this section. Note that the monotonicity proof in the previous section also holds for non-contiguous strategies for trees and also for graphs. And it also holds, if Rule 3. can be applied. This means that the progressive crusades of frontier $k$ and the search number $k$ for graphs correspond in general in the same way. Recontamintion does not help and optimal monotone strategies always exists.

More precisely, the connectivity relationship in the proof of Lemma 19 depicted in Figure 2.7 was only shown for trees. We obtained a progressive *connected* crusade. In general the use of a progressive crusade is sufficient. Conversely, in the proof of Lemma 20 the three cases depicted in Figure 2.10 can also be handled, if the progressive crusade is not connected.

**Exercise 15** *Consider the proof of Theorem 17. Argue, that with the same arguments, we can show:* For any unit-weighted graph $G$, with search number $s(G)$, there is always a monotone strategy with $s(G)$ searchers.

So we can ask what is the prize for the connectivity. General strategies for the above rules indeed have better search numbers as we will show here. We between the search number, $s(G)$, for general strategies and the contiguous search number, $\mathrm{cs}(G)$, for contiguous strategies. As mentioned above for both measures we find optimal monotone strategies.

Let $D_k$ denote a tree with root $r$ of degree three and three full binary trees, $B_{k-1}$, of depth $k-1$ connected to the $r$. We first show that $cs(D_k) = k+1$ holds.

**Lemma 31** *For the graph $D_k$, we conclude $cs(D_k) = k+1$.*

**Proof.** Let $T_1$, $T_2$ and $T_3$ denote the copies of $B_{k-1}$ connected to the root and let $e_i$ denote the edge that connects $T_i$ with the root $r$. For the contiguous search w.l.o.g. we can assume that the edge $e_1$ is cleared first at timestep $i_1$ among the edges $e_1$, $e_2$ and $e_3$. W.l.o.g. let $i_2 > i_1$ denote the time step where for the first time a leaf $l$ of $T_2$ or $T_3$ is reached. Assume w.l.o.g. $f \in T_2$. At time step $i_2 - 1$ the path $P(r, x_k) = r, x_1, \ldots, x_{k-1}$ of length $k-1$ from $r$ to the neighbor $x_{k-1}$ of $f$ with $k$ vertices has to be clean and for any $x_i \in P(r, f)$ there is a unique subtree in $T_2$ different from $f$ that is not fully decontaminated. For the root $r$ there is a subtree in $T_3$ that is not fully decontaminated. So at least one searcher for any $x_i$ and for $r$ is required which gives $k$ in total. One additional searcher now is required for cleaning $f$. This gives $\mathrm{cs}(D_k) \geq k+1$.

On the other hand $k+1$ searchers are sufficient, if we start at the root with $k+1$ agents and first clean a leaf and its neighbor. Recursively, and full binary subtree of depth $l$ is cleaned from the root with $l+1$ searchers. □

Now, we would like to relate this to the number $s(D_k)$. We consider $D_{2k-1}$ with $\mathrm{cs}(D_{2k-1}) = 2k$.

**Lemma 32** *For $D_{2k-1}$ we conclude $s(D_{2k-1}) \leq k+1$.*

**Proof.** For $k = 1$ the statement is trivial. So assume $k > 1$. We first place one agent at the root $r$ and successively clean the copies of $B_{2k-2}$ by $k$ agents. The last statement is shown by induction. For $k = 2$ we place one agent at the root and a single agent starting at a leaf cleans first the left subtree of $r$, is then moved to the leaf of the right subtree and cleans the second subtree. Finally, all agents are placed at the root.

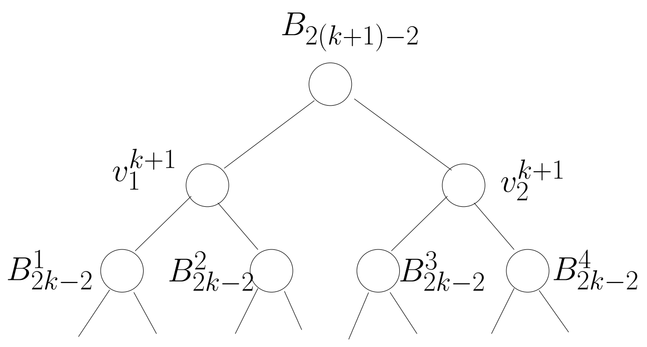


Figure 2.17: The inductive step. Each subtree $B^i_{2k-2}$ can be cleaned by $k$ agents. Placing an additional searcher at $v_1^{k+1}$ in the beginning, we first clean $B^1_{2k-2}$ and $B^2_{2k-2}$ successively by $k$ agents, move all $k+1$ searchers to $v_2^{k+1}$ and do the same for $B^3_{2k-2}$ and $B^4_{2k-2}$ successively. Altogether, $k+1$ searchers are sufficient for $B_{2(k+1)-2}$.

Let us assume that the statement holds for $k \geq 2$. We can fully clean $B_{2k-2}$ with $k$ searchers that are finally located at the root. The tree $B_{2(k+1)-2}$ has four subtrees $B^i_{2k-2}$ for $i = 1, 2, 3, 4$ of depth $2k - 2$as shown in Figure 2.17 that can be cleaned by induction hypothesis with $k$ agents. Let $v_1^{k+1}$ be the ancestor of $B^1_{2k-2}$ and $B^2_{2k-2}$ and let $v_2^{k+1}$ be the ancestor of $B^3_{2k-2}$ and $B^4_{2k-2}$.

We place one additional agent at $v_1^{k+1}$ and clean $B^1_{2k-2}$ and $B^2_{2k-2}$ successively by $k$ agents. Then all $k+1$ agents move over the root toward $v_2^{k+1}$. Here we again leave the additional agent at $v_2^{k+1}$ and clean clean $B^3_{2k-2}$ and $B^4_{2k-2}$ successively by $k$ agents. Thus $k+1$ agents are required.

By induction, $B_{2k-2}$ can be cleaned by $k$ searchers and for $D_{2k-1}$ at most $k+1$ searchers are required. □

Now, we have a fixed relationship between $cs(G)$ and $c(G)$ for $G = D_{2k-1}$. We have $s(D_{2k-1}) \leq k+1$ and $\text{cs}(D_{2k-1}) = 2k$.

**Corollary 33** *There exists a tree $T$ so that $cs(T) \leq 2s(T) - 2$ holds.*

It was also shown by Barrière et al. 2012, that there is no tree $T$ with ratio $\frac{cs(T)}{s(T)}$ larger than 2. More precisely,

$$\frac{cs(T)}{s(T)} < 2 \text{ for all trees } T\,.$$

The proof of this fact relies on the fact that in principle (up to retractions) the trees $D_k$ can be considered to be the category of graphs that gives the worst-case ratio. If there is some time left at the end of the semester we will prove this fact.

# Chapter 3

# Discrete Cop and Robber game

In this chapter we would like to discuss another discrete variant of the intruder search problem. In comparison to the previous chapter, we assume that at any time of the game the position of the single intruder is given.

More precisely, there is a single robber $R$ and a set of cops $C$ and a graph $G = (V, E)$. The game starts with the cops, by choosing the starting vertices for the set $C$. After that, the robber $R$ can choose its starting vertex. The game runs in subsequent steps. First, any cop can move from a vertex to an incident vertex, then the robber can move. The game ends, when one cop enters the position of the robber or the robber enters the position of a cop, respectively.

*Cop and Robber game for graphs:*
**Instance:** A Graph $G = (V, E)$ and the cardinality of the cops $C$.
**Question:** Is there a winning strategy $S$ for the cops $C$?

We are searching for classifications of graphs that allow a winning strategy for $C$ or vice versa a winning strategy for $R$. Aigner and Fromme introduced the problem in the midst of the 90ies.

## 3.1 Classifications of graphs

### 3.1.1 Simple examples and pitfalls

It is interesting to see that it makes a difference, if we do not allow the robber to keep in place during its strategy. This is called the *active* version of the game, in correspondance to the *passive* version, where the robber is not forced to move in any step.

Figure 3.1 shows an example where this makes a difference for a single cop. In the active version the cop starts at vertex $v_1$ and the robber can only choose the opposite vertex $r_2$. The cop moves toward $v$. Now the robber has to move to $r_1$. The cop moves toward $v_2$ and after the next mandatory move of the robber, the robber will be catched. In the passive version the robber can move around or rest in the 4-cycle and holds distance 2 from the cop all the time. In the following we will always discuss the more intuitive passive version of the game. Let $G_C$ denote the set of all graphs that allow a winning strategy for $C$ and let $G_R$ denote all graphs that have a winning strategy for $R$.

Obviously, any tree $T$ belongs to $G_C$ already for a single cop, that successively moves into the subtree of $R$. Additionally, for a single cop, all graphs that contain a cycle of length at least 4 belong to $G_R$.

We concentrate on a single cop. In the winning case for the cop, the final situation is as follows: The robber is located in a vertex $v_r$ and the cop is located in $v_c$ for an edge $e = (v_r, v_c)$. Moreover, all neighbors, $N(v_r)$, of $v_r$ are also neighbors of $v_c$, which means $N(v_r) \subseteq N(v_c)$.

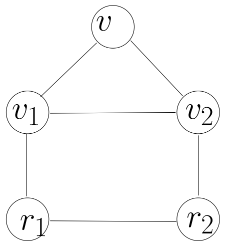


Figure 3.1: In this simple graph for one cop and a robber it makes a difference, if the robber has to perform moves mandatorily.

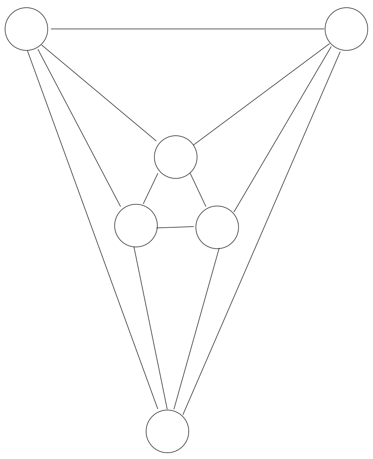


Figure 3.2: A graph without pitfalls.

For a pair $(v_r, v_c)$ of vertices we call $v_r$ a *pitfall* and $v_c$ its *dominating vertex* if $N(v_r) \cup \{v_r\} \subseteq N(v_c)$ holds. Obviously, a graph $G$ whithout a pitfall is in $G_R$. Figure 3.2 shows an example.

**Exercise 16** *Present a construction scheme for graphs of arbitrary size without pitfalls.*

## 3.1.2 Algorithmic approaches

We would like to show that for a single cop the classification of a graph depends on the successive removement of pitfalls of $G$.

**Lemma 34** *Let $v_r$ be a pitfall of some graph $G$. Then*

$$G \in G_C \Longleftrightarrow G \setminus \{v_r\} \in G_C\,,$$

*where $G \setminus \{v_r\}$ results from $G$ by removing all edges adjacent to $v_r$ and vertex $v_r$ from $G$.*

**Proof.** If $G \setminus \{v_r\} \in G_R$ holds, the robber simply identifies any visit of the cop of the pitfall $v_r$ by the dominating vertex $v_c$ and makes use of a strategy in $G \in G_R \setminus \{v_r\}$. Note that the neighborship of the pitfall $v_r$ is a subset of the neighorship of $v_c$. This means that we do not *weaken* the cop by the identification.

If $G \setminus \{v_r\} \in G_C$ holds, the cop wants to extend its winning strategy to $G$. The cop simply identifies any visit of the robber of the pitfall $v_r$ as a visit of the dominating vertex $v_c$ and makes use of the same strategy. Again, since the neighborship of the pitfall $v_r$ is a subset of the neighorship of $v_c$, we do not weaken the robber by this identification. If at the end we catch the imaginary robber at $v_c$ and the robber is physically located at $v_r$, we will catch the robber in the next step. □

Now, we have a simple characterization of $G_C$.

**Theorem 35** *The graph $G$ is in $G_C$, if and only if the successive removement of pitfalls finally ends in a single vertex. The classification of a graph can be computed in polynomial time.*

**Proof.** Lemma 34 gives the key argument, as the classification does not change by removing pitfalls. This means that we either end up in a graph with no pitfalls for $G \in G_R$ or in a single vertex for $G \in G_C$.

Checking the existance of a pitfall can be done locally for any vertex and its neighborship. After computing the neigborship sets, we can check the pitfall property for a vertex in a brute-force manner in $O(n^2)$ time and for all vertices in $O(n^3)$ time for a graph with $n$ vertices. At most $n$ reduction steps can be done. □

**Exercise 17** *Design an efficient algorithm for checking the pitfall property of a single vertex and/or for the graph.*

The above shrinking process answers the classification question algorithmically in polynomial time. On the other hand we would like to construct arbitrary examples of representatives of $G_C$. It can be shown that $G_C$ is closed under the operations *product of two graphs* and *reduction of a graph.*

The *product* $G_1 \times G_2$ of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, G_2)$ is defined by vertex set $V_1 \times V_2$ and an edge set by the folllowing rules: $(v_1, v_2)$ and $(w_1, w_2)$ of $V_1 \times V_2$ build an edge if:

1. $v_1 = w_1$ and $(v_2, w_2) \in E_2$ or
2. $(v_1, w_1) \in E_1$ and $v_2 = w_2$ or
3. $(v_1, w_1) \in E_1$ and $(v_2, w_2) \in E_2$.

**Lemma 36** *If $G_1, G_2 \in G_C$, then $G_1 \times G_2 \in G_C$*

**Proof.** If the cop has a winning strategy for $G_1$ that starts in $v_1^s$ and catches the robber in $v_1^e$ and for $G_2$ that starts in $v_2^s$ and catches the robber in $v_2^e$, the cop can start in $(v_1^s, v_2^s)$ apply the strategies simultaneously and finally catches the robber in a vertex $(v_1^e, v_2^e)$. This strategy is obviously correct. □

For a graph $G$ and its subgraph $H$, the *retraction* from $G$ to $H$ is a mapping $\varphi : V(G) \mapsto V(H)$ of the vertices of $V(G)$ of $G$ to the vertices $V(H)$ of $H$ as follows: $\varphi(H) = H$ for $(u, v) \in E$ we have $(\varphi(v), \varphi(u)) \in E(H)$. The graph $H$ is called a *retract* of $G$, if a retraction from $G$ to $H$ exists.

Note that $G \setminus \{v_r\}$ for a pitfall $v_r$ is a retract of $G$.

**Lemma 37** *If $G \in G_C$, and graph $H$ is a retract of $G$, then $H \in G_C$.*

**Proof.** Assume that $H \in G_R$ holds and let $\varphi$ be the mapping for a retraction from $G$ to $H$. We would like to show $G \in G_R$. We extend the winning strategy for $H$ to a winning strategy of $G$ as follows: $R$ remains in $H$ and identifies the moves of $C$ in $G$ as moves in $H$. That is, if $C$ moves from $v$ to $u$ in $G$, the robber indentifies this move as a move from $\varphi(u)$ to $\varphi(w)$ which exists in $H$ by definition of $\varphi$. The robber always moves according to the strategy in $H$ and cannot be catched. □

Note that, the above lemmata do not rely on the fact that there is only one cop.

**Theorem 38** *The class of graphs $G$ in $G_C$ is closed under the operations product and retraction.*

### 3.1.3 How many cops are required?

Obviously, any graph with a 4-cycle will not belong to $G_C$, therefore it makes sense to think about more than one cop. For a graph $G$ the *cop-number*, $c(G)$, denotes the minimum number of cops required to guarantee that $G \in G_C$ holds.

A *vertex cover* of a graph $G$ is a subset $V_c \subseteq V$ so that any vertex $u \in V \setminus V_c$ has a neighbor in $V_c$. Therefore the minimum vertex cover is an upper bound on $c(G)$. First, we show that $c(G)$ can be arbitrarily large for some graphs.

**Theorem 39** *Let $G = (V, E)$ be a graph with minimum degree $n$ that contains neither* 3- *nor* 4-*cycles. We conclude $c(G) \geq n$.*

**Proof.** Let us assume that $n - 1$ cops are sufficient. If $G$ does not have a vertex cover of size smaller than $n$, the $n-1$ cops located in the beginning at $c_1, \ldots, c_{n-1}$ cannot prevent the robber to choose a safe vertex. So the robber chooses such a vertex whose neighbors are not occupied by the cops. Since there are no 3- and 4-cycles, by the next move a single cop cannot threaten (occupy and/or be adjacent to) two neighbors of the robber in the next step. Therefore, there is still one safe neighbor for the robber after the next move of the cops.

It remains to show that a vertex cover of size $< n$ does not exist. Consider any vertex set $V = \{v_1, \ldots, v_{n-1}\}$ of $G$ and a vertex $w \neq v_i$ for $i = 1, \ldots, n-1$. Note that $|V| \geq n$ holds, so $w$ exists. Now consider the neighborhood, $N(w)$, of $w$. Let it consist of $k$ vertices $v_1, \ldots, v_k$ from $V$ and $l - k$ vertices $w_1, \ldots, w_{l-k}$ not in $V$. We have $l \geq n$, $k \leq n-1$ and $l - k \geq 1$. There are no 3- and 4-cycles, so $N(w_i) \cap N(w_j)$ has to be $\{w\}$ for $i \neq j$. If the set $V$ is a vertex cover for $G$, any $N(w_i)$ has to contain a different vertex from $V$. But none of the $N(w_i)$'s can contain a vertex of $v_1, \ldots, v_k$, since this would give a 3-cycle with $w$. This means that we require $l - k$ different vertices from $v_{k+1}, \ldots, v_{n-1}$ and $n$ vertices from $V$ in total, a contradiction. □

We can construct regular graphs of arbitrary size, which fulfill the condition of Theorem 39. The following Theorem is given by construction.

**Theorem 40** *For every $n$ there exists a graph without* 3- *or* 4-*cylces with minimum degree $n$. So, for any $n$ there is a graph with $c(G) \geq n$.*

**Proof.** For $n = 2$ the simple 5-cycle will work. Note that $C_5$ is 3-colorable, which means that we color the vertices by three colora such that no two colors are adjacent. Three colors are required and sufficient for $C_5$. Inductively, we construct a 3-colorable graph with degree exactly $n$ for any vertex $v$ of $G$ and without 3- and 4-cycles. The coloring is required for maintaining the cycle condition by construction.

The induction base for $n = 2$ was shown above. Let us assume that the statement holds for $n$. We consider four copies $G_0, G_1, G_2, G_3$ of a corresponding graph $G$ for $n$ as depicted in Figure 3.3. We build new edges with respect to the coloring of the vertices so that any vertex obtains an additional edge; see Figure 3.3. From $G_i$ to $G_j$ the exact copies of two vertices of a single color are connected. For example from $G_1$ to $G_2$ all identical copies of color 3 are

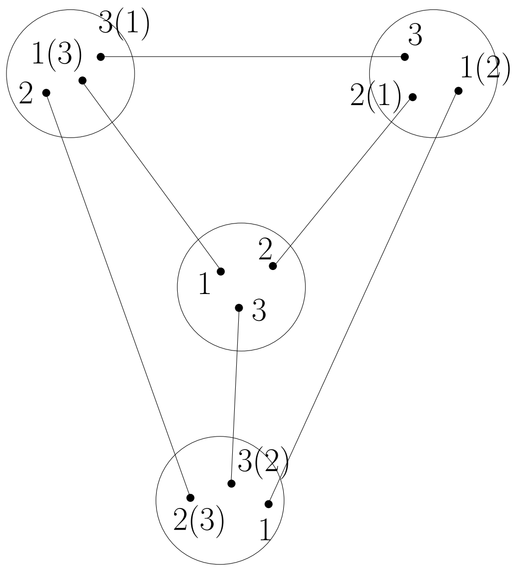


Figure 3.3: In the inductive step we use four copies $G_0, G_1, G_2, G_3$ of a 3-colorable graph $G$ of degree exactly $n$ without 3- and 4-cycles. Then we construct new edges according to the colors and finally interchange some colors, appropriately.

connected, from $G_2$ to $G_0$ all identical copies of color 2 are connected and so on. There is a unique correspondance as shown in Figure 3.3.

Since there are no cycles of size 3 or 4 in $G_0, G_1, G_2, G_3$ and any two edges between $G_i$ and $G_j$ make use of identical copies of the same color there are at least two edges between them in $G_i$ and $G_j$ respectively. So there are no 3- or 4-cycles in the new graph. For the inductive step, we require a *new* 3-coloring, which will be attained by interchanging the colors for example color 3 by 1 and color 1 by 3 in $G_1$ and color 2 by 1 and color 1 by 2 in $G_2$ and so on. Thus, we maintain 3-coloring in $G_0, G_1, G_3, G_4$ and also for the connections. □

Finally, in this section we prove some positive results by bounding the cop-number from above for special graphs. The corresponding proofs are constructive, i.e., a winning strategy for the cops can be computed.

**Theorem 41** *Consider a graph $G$ with maximum degree 3 and the property that any two adjacent edges are contained in a cycle of length at most 5. Then $c(G) \leq 3$.*

**Proof.** The proof is constructive in the following sense. If the position of the robber is known, for the cops $c_1, c_2$ and $c_3$ we consider three paths towards $r$ that use all incident edges to $r$. We choose $P_1$, $P_2$ and $P_3$ for $c_1, c_2$ and $c_3$ respectively. The paths cover the incident edges by different cops and with length $l_1$, $l_2$ and $l_3$. And the paths make use of any possible shortcut for reaching the incident edges. Note that the paths need not be disjoint and $r$ might also have only one or two incident vertices. But such paths do always exist. We would like to argue that by the condition of the Theorem, we can decrease the overall distance $l := l_1 + l_2 + l_3$ in any move of the cops.

Formally, after the move of the robber, $R$, we move $c_1$, $c_2$ and $c_3$ to $c_1'$, $c_2'$ and $c_3'$ so that $l' < l$ holds. We further assume that $r$ was adjacent to exactly three vertices $r_1$, $r_2$ and $r_3$. The other cases can be handled anlogously, and are given as an exercise. We have $P_1 = \{c_1, \ldots, r_1, r\}$, $P_2 = \{c_2, \ldots, r_2, r\}$ and $P_3 = \{c_3, \ldots, r_3, r\}$ and consider the following cases.

1. The robber $R$ stands still. The cops move along the paths toward $R$ and $l' \leq l - 3$.

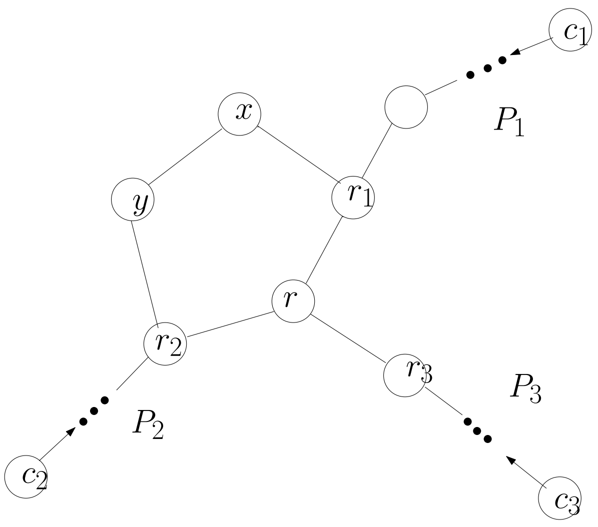


Figure 3.4: If $r$ has degree 3 and $c_1$ is not on $r_1$, there is a 5-cycle so that we can move closer to $r$ at least by one.

2. The robber $R$ moves to $r_1$ (w.l.o.g.):

   $r_1$ **has degree 1:** This cannot happen since $(r, r_1)$ and $(r, r_2)$ are neighboring edges.

   $r_1$ **has degree 2:** Either $c_1$ was on $r_1$ and we are done or move all three cops toward $r$ which gives $l' \leq l_1 - 2 + l_2 + l_3 = l - 2 < l$.

   $r_1$ **has degree 3:** Either $c_1$ was on $r_1$ and we are done or we have $l_1 \geq 2$. At least one adjacent vertex, say $x$, of $r_1$ does not belong to $P_1$, otherwise we use a shortcut for $P_1$. This means that $(x, r_1)$ and $(r_1, r)$ are on a cycle of length at most 5. Since $r$ has only degree 3, one of the vertices $r_2$ or $r_3$, (say $r_2$) also belong to this cycle as depicted in Figure 3.4. So we have a 5-cycle $r_2, y, x, r_1, r$. We move all three cops toward $r$, respectively $r_1$ and use the paths $P_1 = \{c_1', \ldots, r_1\}$ $P_2 = \{c_2', \ldots, r_2, y, x, r_1\}$ and $P_3 = \{c_3', \ldots, r_3, r, r_1\}$ with length $l' \leq l_1 - 2 + l_2 + 1 + l_3 = l - 1 < l'$.

In any case we can reduce the distance of the cops to the robber for the paths to all all three neighboring vertices. □

Finally, we would like to prove that for any planar graph $G$, indeed $c(G) \leq 3$ holds. We first show that in any graph $G$ it is always possible to protect a shortest path between two vertices by two cops. Protection means the robber cannot enter the path without being caught in the next step. The path length is given by the number of edges along a path between to vertices in $G$. By this measure the triangle inequality holds.

**Lemma 42** *Consider a graph $G$ and a shortest path $P = s, v_1, v_2, \ldots, v_n, t$ between two vertices $s$ and $t$ in $G$, assume that we have two cops. After a finite number of moves the path is protected by the cops so that after a visit of the robber $R$ of a vertex of $P$ the robber will be caught.*

**Proof.** First, we move a cop $c$ onto some vertex $c = v_i$ of $P$. Let $d(x, y)$ denote the distance between two vertices in $G$. The robber $R$ located at $r$ can only have a shorter distance to vertices on one side of $P$ w.r.t. $c$ because the triangle inequality holds. Assuming, that $r$ is closer to some $x$ in $s, v_1, \ldots, v_{i-1}$ *and* some $y$ in $v_{i+1}, \ldots, v_n, t$ is a contradiction to the shortest path between $x$ and $y$. That is $d(x, c) + d(y, c) \leq d(x, r) + d(r, y)$. This means that $d(r, x) < d(c, x)$ only holds at most for one side of $P$ w.r.t. $c$ and for the other side we conclude $d(r, y) > d(c, y)$ in this case.

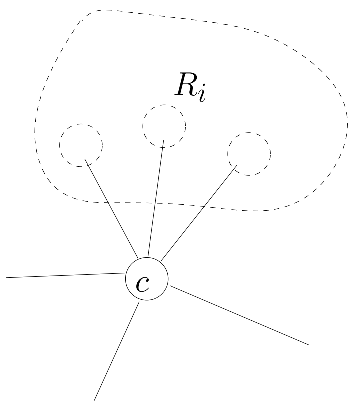


Figure 3.5: Case 1: All three cops in one vertex.

Thus, we move $c$ towards the vertices $x$. Now the robber can move. Again, if there are still vertices on one side of $P$ w.r.t. $c$ which are closer to $r$ than to $c$ we move further on toward these vertices. So finally, we achieved

$$d(r,v) \geq d(c,v) \text{ for all } v \in P \tag{3.1}$$

by this process.

Now the robber again could make its move. We show that we can maintain the inequality at all time, which means that the robber will be caught if it tries to move towards the vertices of $P$.

Assume Equation 3.1 holds. The robber can either stay in its place, so the cop $c$ does and we fulfill Equation 3.1 (and the second cop could move now). Or the robber moves and tries to contradict Equation 3.1 by its single move. Assume $r$ goes to some vertex $r'$, we have

$$d(r',v) \geq d(r,v) - 1 \geq d(c,v) - 1 \text{ for all } v \in P\,.$$

If there is again some $v' \in P$ with $d(c,v') - 1 = d(r',v')$, we have the same situation as above and we can move $c$ toward $v'$ and Equation 3.1 holds again. Again as before the movement toward $r'$ cannot reduce the distance to $x$ and $y$ on opposite sides of $c$ w.r.t. $P$. Thus, by the move toward some $v'$ we fulfill Equation 3.1. □

Finally, we exploit the above property for planar graphs and by the use of 3 cops and two such paths.

**Theorem 43** *For any planar graph $G$ we have $c(G) \leq 3$.*

**Proof.** We show that the region $R_i$ for the robber $R$ will shrink successively, that is $R_{i+1} \subset R_i$ after some moves of the cops. Two situations can appear.

**Case 1:** All three cops occupy a single vertex $c$ and the robber is located in one component $R_i$ of $G \setminus \{c\}$; see Figure 3.5.

**Case 2:** There are two different paths $P_1$ and $P_2$ from $v_1$ to $v_2$ that are protected in the sense of Lemma 42 by cops $c_1$ and $c_2$; see Figure 3.6. In this case $P_1 \cup P_2$ subdivides $G$ into an interior, $I$, and an exterior region $E$. That is $G \setminus (P_1 \cup P_2)$ has at least two components. W.l.o.g. we assume that $R$ is located in the exterior $E = R_i$.

We will show that these two cases can appear successively and the region $R_i$ of the robber will shrink. In the beginning all cops are located in a single vertex $c$ and we are in case 1. We show how we handle the cases.

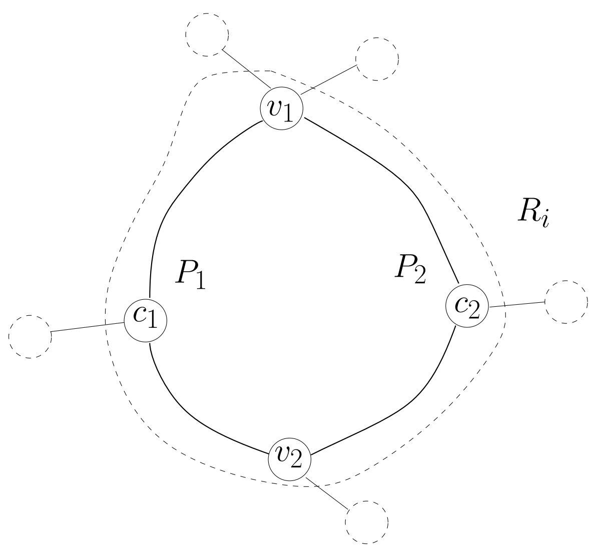


Figure 3.6: Case 2: Two cops protect two paths.

**Movements in Case 1:** We consider different situations for the neighbors of $c$:

$c$ **has one neighbor in** $R_i$**:** Move all cops to this neighbor $c'$ and consider $R_{i+1} = R_i \setminus \{c'\}$. This gives Case 1 again.

$c$ **has more than one neighbor in** $R_i$**:** Let $a$ and $b$ be two of the neighbors and let $R(a, b)$ be a shortest path in $R_i$ between $a$ and $b$. One cop remains in $c$, another cop protects the path $P(a, b)$ by Lemma 42. Thus $P_1 = a, c, b$ and $P_2 = P(a, b)$. We are in Case 2 with $R_{i+1} \subset R_i$.

**Movements for Case 2:** We consider the situation for the composition of $R_i$ and the location of the robber. We first assume that there is another shortest path $P'(v_1, v_2)$ but different from $P_1$ and $P_2$ that partially runs in $R_i$ and also connects $v_1$ and $v_2$.

Let $x_1$ be the first vertex of $P'(v_1, v_2)$ where the path $P'$ leaves $P_1 \cup P_2$ after $v_1$ and $x_2$ be the first vertex of $P'$ after $x_1$ that enters $P_1 \cup P_2$ again.

We let $c_3$ protect the path $P_3$ which results from combining $P_{1,2}(v_1, x_1)$ with $x_1, r_1, \ldots, r_l, x_2$ and with $P_{1,2}(x_2, v_2)$ as depicted in Figure 3.7. While $c_1$ and $c_2$ protect $P_1$ and $P_2$, the cop $c_3$ can protect this path $P_3$ from $v_1$ to $v_2$. At the end $c_3$ protects $P_3$ and $c_1$ or $c_2$ protect the remaining path, we are in Case 2 with $R_{i+1} \subset R_i$.

On the other hand, if there is no path different from $P_1$ and $P_2$ and partially running in $R_i$ that connects $v_1$ and $v_2$, there are no such leave and entry vertices $x_1$ and $x_2$ that are connected in $R_i$. Thus, the robber has to be inside a component that fully is connected to a single vertex $x$ on $P_1$ and $P_2$. Thus, we move $c_3$ to this vertex, and also $c_1$ and $c_2$ and end in Case 1 again. □

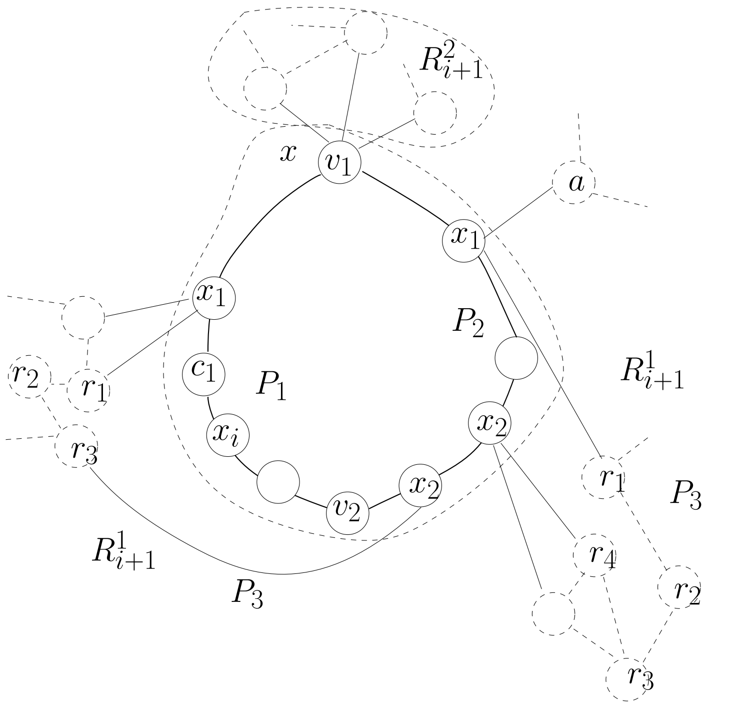


Figure 3.7: The situation for the two chains $P_1$ and $P_2$ protected by $c_1$ and $c_2$. If there is another shortest path from $v_1$ to $v_2$ different from $P_1$ and $P_2$ that runs partially in $R_i$, we construct a path $P_3$ that runs from $v_1$ to $v_2$ that is protected by $c_3$ alone and protects vertices of $R_i$. If there is no such path, a vetex $x$ exists that can be visited by all cops and gives Case 1 again.

# Chapter 4

# Randomized variants

In this chapter and before turning over to some geometric variants of the intruder search problem, we would like to resume the graph decontamination problem for stationary guards in order to show that there are also randomized strategies and problem variants that can be discussed.

We show a slightly better approximation as the greedy algorithm for trees by a randomized strategy. Additionally, we interpret the search number of a graph in the configuration that the fire spreads from any vertex with the same probability. We concentrate on positive results.

## 4.1 Better approximations for trees by randomization

We pick up the firefighter problem for trees again. As already asked for in Exercise 10 we can formulate the problem as an integer LP by the following rules. Let $v \leq u$ denote that $v$ equals $u$ or is a predecessor of $u$ w.r.t. the root $r$ of tree $T$.

$$\begin{array}{lrcll}
\text{Minimize} & \sum\limits_{v \in V} x_v w_v & & & \\
\text{so that} & x_r = 0 & = & 0 & \\
 & \sum\limits_{v \leq u} x_v & \leq & 1 & : \text{ for every leaf } u \\
 & \sum\limits_{v \in L_i} x_v & \leq & 1 & : \text{ for every level } L_i, i \geq 1 \\
 & x_v & \in & \{0,1\} & : \forall\, v \in V
\end{array}$$

In the above integer LP the weights $w_v$ denote the number of vertices in the subtree $T_v$ of vertex $v$ w.r.t. the root $r$.

Let $\text{opt}_{ILP}$ denote the optimal solution of the above integer LP. For the approximation we solve the problem in polynomial time for $x_v \in \mathbb{R}^{\geq 0}$. The optimal solution, $\text{opt}_{RLP}$, is a fractional solution so that a subtree $T_v$ with $x_v = a \leq 1$ is called *$a$-saved*, a portion $a \cdot w_v$ of the subtree is saved. For two vertices $v_1$ and $v_2$ on the same path from the root $r$ to a leaf $u$ and $v_1$ is ancestor of $v_2$ and $x_{v_1} = a_1$ and $x_{v_2} = a_2$ the vertices of $T_{v_2}$ are $(a_1 + a_2)$-saved. The remaining vertices of $T_{v_1}$ are only $a_1$-saved.

The simple idea is that we would like to use a rounding scheme. But we do this in a randomized fashion. For each level we interpret the $a$-values as a probability dsitribution for choosing a vertex to be safe. This is a rounded strategy w.r.t. the distribution. On each level we simply choose a vertex at random according to its distribution. Note that the sum of the $a$-values for level $i$ could be smaller than 1. We interpret the remaining portion as the probability of choosing none of the vertices in this level. The main problem is that we might choose vertices that are on the same path from the root to a leaf. If no such *double-protections* occur the expected value of the rounded strategy would be at least $\text{opt}_{ILP}$ and the expected approximation value would be indeed 1.

If also a successor of a vertex is choosen by our procedure, we simply delete it in the final solution and do not choose another vertex at this level. This makes the choosing procedure independent for every level. Altogether, the only loss we have is for the double-protections. Let us assume that they can occur. What happens if the a tree $T_{v_i}$ at level $i$ is *fully* saved by the fractional strategy? We would like to argue that in the worst-case the fractional strategy has assigned a $1/i$ fraction to all vertices on the path from $r$ to $v_i$ and the subtree is fully saved by the rounding scheme with probability

$$1-(1-1/i)^i \geq 1-\frac{1}{e}\,.$$

We put this intuition into a formal argument.

**Theorem 44** *Consider an algorithm that protects the vertices w.r.t. the probability distribution given by $\text{opt}_{RLP}$. The expected approximation ratio of the above strategy for the number of vertices protected is $\left(1-\frac{1}{e}\right)$.*

**Proof.** Let $S_F$ denote the fractional solution for $\text{opt}_{RLP}$. For an integer solution, we choose a vertex on each level w.r.t. the probability distribution from $\text{opt}_{RLP}$. Let $S_I$ denote the outcome of this assignment. We would like to show, that the expected value of $S_I$ is larger than $\left(1-\frac{1}{e}\right)$ times the value of $S_F$ which in turn outperforms $\text{opt}_{ILP}$.

Let $x_v^F$ denote the value of $x_v$ for the fractional strategy and let $x_v^I$ denote the value $\{0,1\}$ of the integer strategy. For convenience we denote $y_v = \sum_{u\leq v} x_u \in \{0,1\}$, which indicates whether $v$ is finally saved or not. Let $y_v^F = \sum_{u\leq v} x_u^F \leq 1$ denote the fraction of $v$ saved by the fractional strategy. For $y_v = 1$ it suffices that one of the predecessor of $v$ was chosen. Let $r = v_0, v_1, v_2, \ldots, v_k = v$ be the path from $r$ to $v$, so we have

$$\mathbf{Pr}[y_v = 1] = 1-\prod_{i=1}^{k}(1-x_{v_i}^F)\,.$$

For example, the probability that $v_2$ is safe is $x_1 + (1-x_1)x_2 = 1-(1-x_1)(1-x_2)$ and the probability that $v_3$ is safe is $1-(1-x_1)(1-x_2)+(1-x_1)(1-x_2)x_3 = 1-(1-x_1)(1-x_2)(1-x_3)$ and so on.

Thus we compute

$$\begin{aligned}
\mathbf{Pr}[y_v = 1] &= 1-\prod_{i=1}^{k}(1-x_{v_i}^F)\\
&\geq 1-\left(\frac{\sum_{i=1}^{k}(1-x_{v_i}^F)}{k}\right)^k = 1-\left(\frac{k-\sum_{i=1}^{k}x_{v_i}^F}{k}\right)^k\\
&= 1-\left(\frac{k-y_v^F}{k}\right)^k\\
&= 1-\left(1-\frac{y_v^F}{k}\right)^k \geq 1-e^{-y_v^F} \geq \left(1-\frac{1}{e}\right)y_v^F\,. \qquad (4.1)
\end{aligned}$$

The first inequality is a standard inequality for means of positive real values $\frac{x_1+x_2+\cdots+x_n}{n} \geq \sqrt[n]{x_1\cdot x_2\cdots x_n}$. The second and third inequalities stem from classical analysis where we use the fact that $0 \leq y_v^F \leq 1$ holds.

The value of $S_F$ is simply the sum of all $y_v^F$. Thus, we conclude

$$\mathbf{E}(|S_I| = \sum_{v \in V} \mathbf{Pr}[y_v = 1] \geq \left(1 - \frac{1}{e}\right) \sum_{v \in V} y_v^F = \left(1 - \frac{1}{e}\right) |S_F| .$$

□

Altogether, we have a randomized polynomial time algorithm for trees with an expected approximation ratio better than the $\frac{1}{2}$-approximation of greedy.

**Exercise 18** *Prove* $\frac{x_1+x_2+\cdots+x_n}{n} \geq \sqrt[n]{x_1 \cdot x_2 \cdots x_n}$ *for positive real values* $x_i$*. Also prove the last two inequalities of Equation 4.1 in the proof of Theorem 44.*

## 4.2 Search numbers for random fire sources

The second part on randomization is that we might use it to consider the situation that the starting vertex of the fire has some influence on the number of agents required. Therefore, in this section we again consider the firefigther problem on graphs but the start of the fire is choosen uniformly at random among all vertices. The question is, what is the number of agents required so that for a given class $C$ of graphs it can be expected that at least linear number of vertices can be saved.

This subsumes many questions handled before. We would like to have a classification by the properties of $C$, we would like to find a minimum number $k$ of agents required and we use an expected value for assuming that the fire can start in any vertex with the same probability.

For a graph $G = (V, E)$ and a fixed number $k$ of agents, the $k$-surviving rate, $s_k(G)$, is the expectation of the *proportion* of vertices that can be saved if the fire can start from any vertex with the same probability. We are looking for classes, $C$, of graphs $G$ so that for a fixed constant $\epsilon$, $s_k(G) \geq \epsilon$ holds for any $G \in C$. This means that at least $\epsilon \cdot |V|$ vertices will be saved. For a given graph $G$, a given $k$ and a vertex $v \in V$ let $\mathrm{sn}_k(G, v)$, denote the number of vertices that can be protected by $k$ agents, if the fire starts at $v$.

We are also searching for the minimal number $k$ that guarantees $s_k(G) \geq \epsilon$. This means that

$$\frac{1}{|V|} \sum_{v \in V} \mathrm{sn}_k(G, v) \geq \epsilon |V|$$

has to be shown. For a class $C$ let the minimum number $k$ that guarantees $s_k(G) > \epsilon$ for any $G \in C$ be denoted as the firefighter-number, $\mathrm{ffn}(C)$, of $C$.

*Firefighter-Number for a class $C$ of graphs:*
**Instance:** A class $C$ of graphs $G = (V, E)$.
**Question:** Assume that the fire breaks out at any vertex of a graph $G \in C$ with the same probability. Compute $\mathrm{ffn}(C)$.

**Theorem 45** *For planar graphs we have* $2 \leq \mathit{ffn}(C) \leq 4$.

There is a simple argument for the lower bound $2 \leq \mathrm{ffn}(C)$. Consider a planar bipartite complete graph with 2 and $n-2$ vertices on the corresponding sides. For any starting vertex at most one vertex can be saved and $\frac{1}{n}$ will become arbitrarily small.

For the upper bound we first show a somewhat easier result that shows the main idea. The vertices are subdivided into classes $X$ and $Y$, where a root vertex from set $X$ allows to save many (a linear number of) vertices and a root vertex from the set $Y$ allows to save only few (almost zero) vertices. Finally, $|Y| \leq c|X|$ gives the bound.

**Theorem 46** *For planar graphs $G$ with no 3- and 4-cycle, we have $s_2(G) \geq 1/22$.*

**Proof.** We make use of the Euler formula, $c + 1 = v - e + f$, for planar graphs with $e$ edges, $v$ vertices, $f$ faces and $c$ components. We assume that the graph is connected, that is $c = 1$. A planar graph with no 3- and 4-cycle has average degree less than $\frac{10}{3}$. If not, we assume $\frac{10}{3}v \leq 2e$, summing up the degrees of all vertices gives exactly twice the number of edges and we count less than that by assuption. We can also conclude $5f \leq 2e$, since any face has at least 5 edges that can neighbor two faces, but we might not count all edges twice. This means $f \leq \frac{2}{5}$. Inserting $v \leq \frac{3}{5}e$ into the formula gives $f \geq 2 + \frac{2}{5}$, a contradiction. With similar arguments we can show that a graph with no 3-, 4 and 5-cylces has average degree less than three, which is the question of Exercise 19.

We subdivide the vertices $V$ of $G$ into groups w.r.t. the degree and the neighborship.

- Let $X_2$ denote the vertices of degree $\leq 2$.
- Let $Y_4$ denote the vertices of degree $\geq 4$.
- Let $X_3$ denote the vertices of degree exactly 3 but with at least one neighbor of degree $\leq 3$.
- Let $Y_3$ denote the vertices of degree exacly 3 but with all neighbors having degree $> 3$ (degree 3 vertices not in $X_3$).

Let $x_2$,$x_3$,$y_3$ and $y_4$ denote the cardinality of the sets, respectively.

Let $|V| = n$. For a vertex starting in $X_2$, by two agents we protect the neighbors and safe $n - 2$ vertices. For a vertex in $X_3$, we save two neighbors so that the fire spreads to the neighbor $u$ of degree $\leq 3$ and in the next step we protect the remaining neighbors of $u$, thus protecting $n - 2$ vertices in total. For starting vertices in $Y_3$ and $Y_4$, we assume that we can save no vertex.

We have to show that $\frac{1}{n}\sum_{v\in V} \mathrm{sn}_k(G, v) \geq \epsilon \cdot n$ holds and we consider

$$s_2(G) = \frac{1}{n^2}\sum_{v\in V} \mathrm{sn}_k(G, v) \geq \frac{1}{n^2}(x_2 + x_3)(n - 2) = \frac{n-2}{n} \cdot \frac{x_2 + x_3}{x_2 + x_3 + y_3 + y_4} \tag{4.2}$$

since $x_2 + x_3 + y_3 + y_4 = n$ holds.

We first would like to compute a correspondance between $Y_3$ and $Y_4$ and consider the graph $G_Y = (V_Y, E_Y)$ that consists of the edges of $G$ with precisely one vertex in $Y_3$ and one vertex in $Y_4$. The graph $G_Y$ has precisely $3y_3$ edges and at most $y_3 + y_4$ vertices. Note that some of the vertices of $Y_4$ might be neighbors for more than one vertex of $Y_3$. The graph $G_Y$ is bipartite and a subgraph of $G$. A cycle of size 5 has to go forth and back from $Y_3$ to $Y_4$ vertices and has to end at the same class $Y_4$ or $Y_3$. Therefore in $G_Y$ we only have cycles of size at least 6 and by Exercise 19 the average degree of vertices of $G_Y$ is at most 3. This means by counting $3(y_3 + y_4)$, we have counted at least any edge twice, which gives $3(y_3 + y_4) \geq 6y_3$ and $y_3 \leq y_4$.

Now we would like to compute a fixed relation between $x_2 + x_3$ and $y_3 + y_4$. By the average degree, by counting $\frac{10}{3}(x_2 + x_3 + y_3 + y_4)$ edges we have at least counted $3x_3 + 3y_3 + 4y_4$ edges, which gives $9x_3 + 9y_3 + 12y_4 \leq 10(x_2 + x_3 + y_3 + y_4)$ and in turn $2y_4 - y_3 \leq 10x_2 + x_3$. By $y_3 \leq y_4$ we have $y_4 \leq 10x_2 + x_3$ and also $y_3 + y_4 \leq 20x_2 + 2x_3 \leq 20(x_2 + x_3)$.

Now insertion into Equation 4.2 gives

$$s_2(G) \geq \frac{n-2}{n} \cdot \frac{x_2 + x_3}{x_2 + x_3 + y_3 + y_4} \geq \frac{n-2}{n} \cdot \frac{x_2 + x_3}{21(x_2 + x_3)} = \frac{n-2}{21n}\,. \tag{4.3}$$

If $G$ has only two vertices, in any case the vertex distinct from the root can be saved. If $G$ has $3 \leq n \leq 44$ vertices, at least $\frac{2}{44}$ are saved in a single step. For $n \geq 44$ we have $s_2(G) \geq \frac{42}{21\cdot 44} = \frac{1}{22}$. So the expected value of saved vertices is always $\frac{1}{22}n$. □

**Exercise 19** *Prove by the Euler formula that a graph with no* 3-, 4-*cycle and* 5-*cylces has average degree less than three.*

Finally, we would like to prove the statement $\mathrm{ffn}(C) \leq 4$ of Theorem 45. To this end we prove the following result with a precise value of $s_4(G)$ for planar graphs.

**Theorem 47** *Using four firefighters in the first step and then always three firefighters in each step, for every planar graph $G$ there is a strategy such that $s_4(G) \geq \frac{1}{2712}$ holds.*

**Proof.** We can assume that $G$ is a maximal planar graph without multi-edges. Inserting more edges will only help the fire but multi-edges will not. This means that $G$ is a triangulation and we can assume that any face has exactly three edges.

We provide the proof in several steps and the proof contains four Lemmate, namely Lemma 48, Lemma 49, Lemma 50, and Lemma 51.

Similarily, to the proof above we subdivide the vertices $V$ of $G$ into sets $X$ and $Y$. Where $X$ will be the set of vertices where a strategy saves at least $n-6$ vertices and for $Y$ we do not expect to save any vertex, for $|V| = n$.

The final conclusion is that for some $\alpha = \frac{1}{872}$ we will conclude

$$|Y| \leq \left(93 + \frac{3}{\alpha}\right)|X| = 2709|X|\,. \tag{4.4}$$

Thus from $|X| + |Y| = n$ and Equation 4.4 we conclude

$$s_4(G) \geq \frac{n-6}{n} \cdot \frac{|X|}{|X|+|Y|} > \frac{n-2}{n} \cdot \frac{|X|}{2710|X|} = \frac{n-6}{2710n}\,. \tag{4.5}$$

For $n \geq 10846$ we have

$$s_4(G) \geq \frac{1}{2710} - \frac{6}{4 \cdot 2710^2} \geq \frac{2710 - 3/2}{2710^2} \geq \frac{1}{2712}$$

For $2 \leq n < 10846$ we save at least $\min(4, n-1)$ in the first step, which gives also $s_4(G) \geq \frac{1}{2712}$.

The remaining task is, to establish the above bounds. First, we subdivide the vertices accordingly. Note that for starting vertices of degree 3 or four we can save $n-1$ vertices in the first step.

- For degree $3 \leq d \leq 6$ let $X_d$ denote the vertices that guarantee to save at least $|V| - 6$ vertices.
- All other vertices form the set $Y_d$ for $d \geq 5$.

Also note that a starting vertex $v$ of degree 5 with a neighbor $u$ of degree at most 6 is in $X_5$. Because of the triangulation $u$ and $v$ have two common neighbors $n_1$ and $n_2$ . In the first step, we let the fire only spread to $u$ by protecting 4 neigbors at $u$. Then the neighbors $v$, $n_1$ and $n_2$ of $u$ are already protected. So we fully protect the graph in the next step by 3 agents.

We require some more structural properties for the relationsship between $X$ and $Y$ which stem from the triangulation. The length of a path in the graph is given by the number of edges.

**Lemma 48** *For a vertex $v \in Y_6$ there is a path of length at most 3 from $v$ to a vertex $u$ that has degree distinct from $v$ (i.e., $\neq 6$) and the inner vertices of the path have degree exactly 6.*

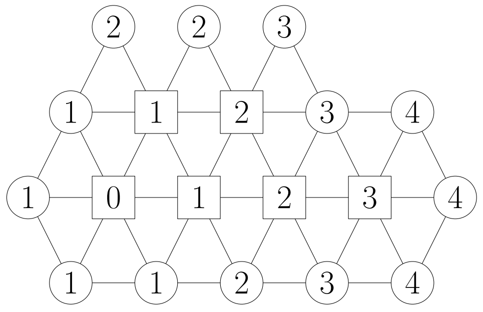


Figure 4.1: If a vertex $v$ of degree 6 is in $Y_6$, we will find a vertex $u$ as given in Lemma 48 in the neighborhood, or we finally end in a situation where $v$ is blocked locally in a hexagon and thus belongs to $X_6$.

**Proof.** Let us assume that this is not the case. In the first step we can always protect 4 subsequent neighbors of $v$ as depicted in Figure 4.1. If one of the remaining two neighbors (step 1) does not have degree 6, we are done. So assume that also these two neigbors have degree 6. Because of the triangulation, they will span a hexagon and we can protect 3 neighbors of these two as depicted in Figure 4.1. The fire spread to only two remaining neighbors (step 2). If one of them does not have degree 6 we are done again. So assume that both also have degree 6 and we extend the hexagonal grid. We can protect the neighbors by three agents as depicted in Figure 4.1 and only one vertex remains on fire after the fire spreads (step 3). If this vertex does not have degree 6 we are done again. But if this vertex also have degree 6 we will finally enclose the fire in the next step and only 6 vertices ($v$, 2 in (step 1), 2 in (step 2), 1 in (step3)) gets burned in total, a contradiction to $v \in Y_6$. Without the above property, $v$ will be in $X_5$! □

The next lemma tells us something about vertices from $Y_d$ with $d \geq 7$ related to $y_5$. Let $d(v)$ denote the degree of vertex $v$.

**Lemma 49** *A vertex with $d(v) \geq 7$ has at most $\lfloor \frac{1}{2} d(v) \rfloor$ neighbors in $Y_5$.*

**Proof.** A neighbor $u$ of $v$ from $Y_5$ has two neighbors $n_1$ and $n_2$ in common with $v$. If one of them has degree strictly less than 7, the vertex $u$ belongs to $X_5$. So the vertices from $Y_5$ around $v$ are *seperated* by vertices of degree $\geq 7$, which gives the bound. □

Finally, we make use of the following structural lemma that stems from the Euler formula and the simple, maximal planar triangulation.

**Lemma 50** *For a simple, maximal planar graph we have*

$$\sum_{v \in V} (d(v) - 6) = -12\,. \tag{4.6}$$

**Proof.** For a maximal, simple planar graph we have $3f = 2e$, by counting the edges of every triangular face, we count any edge exactly twice. Additionally, we have $\sum_{v \in V} d(v) = 2e$ because summing up the degree of the vertices counts any edge twice as well. The Euler formula says $v - e + f = 2$ and we conclude $v - e + \frac{2}{3}e = 2 \iff 2e - 6v = -12$ which gives the conclusion. □

Now we present the main idea for obtaining Equation 4.4. The idea is that we distribute the *intitial potential* $p_1(v) := (d(v) - 6)$ of every vertex among the others so that finally any vertex has potential $p_2(v)$ and also $\sum_{v \in V} p_1(v) = \sum_{v \in V} p_2(v) = -12$ holds.

The rules for the distribution are as follows:

**Rule A:** A vertex $v$ of degree at least 7 gives a value of $\frac{1}{4}$ to each neighbor vertex from $Y_5$.

**Rule B:** For a vertex $v \in Y_6$ we choose exactly one vertex $u$ with $d(u) \neq 6$ and distance $d(v,u) \leq 3$ as in Lemma 48. The vertex $u$ gives a value of $\alpha > 0$ to $v$.

We would like to choose $\alpha$ accordingly, the property $\sum_{v\in V} p_1(v) = \sum_{v\in V} p_2(v) = -12$ will hold since the distribution is cost neutral by construction. Such a distribution with desired properties exists.

**Lemma 51** *There is a constant $\alpha > 0$ such that a distribution by Rule A and B gives $\sum_{v\in V} p_1(v) = \sum_{v\in V} p_2(v) = -12$ and for every $v \in X$ we have $p_2(v) \geq -3 - 93\alpha$ and for every $v \in Y$ we have $p_2(v) \geq \alpha$.*

Before we prove this final lemma, we use its conclusion. An $\alpha = \frac{1}{872}$ will do the job. We then conclude

$$-12 = \sum_{v\in V} p_2(v) \geq (-3 - 93\alpha)|X| + \alpha|Y|$$

and this gives

$$|Y| \leq \left(93 + \frac{3}{\alpha}\right)|X| < 2790|X|$$

which is Equation 4.4. It remains to prove Lemma 51.

**Proof of Lemma 51.**
For the claim we estimate the distribution of the potential as given by the rules $A$ and $B$.

Considering Rule B, how often can a vertex $u$ with $d(u) \neq 6$ give a potential of $\alpha$ to some vertex $v$? We give a rough upper bound with respect to the maximal distance $\leq 3$ from $u$:

- Distance 1: $d(v)$ times to a direct neighbor, if all of them are in $Y_6$. This gives $1 \cdot d(u)$.
- Distance 2: For all $d(v)$ neighbors of the first case, at most 5 times, if the $d(v)$ neighbors of the above case have degree 6 and all 5 remaining neigbors are from $Y_6$. This gives $5 \cdot d(u)$.
- Distance 3: For all vertices of the second case, at most 5 times, if the vertices of the second case all have degree 6 and the remaining neighbors are from $Y_6$. This gives $25 \cdot d(u)$.

Altogether, any vertex $u$ with $d(u) \neq 6$ can give a potential $\alpha$ at most $(1+5+25)d(u) = 31d(u)$ times. This gives upper bounds for the potential $p_2(v)$:

- $v \in X_3$: We have $p_2(v) \geq -3 - 93\alpha$ because $d(v) = 3$ and $p_1(v) = -3$.
- $v \in X_4$: We have $p_2(v) \geq -2 - 124\alpha$ because $d(v) = 4$ and $p_1(v) = -2$.
- $v \in X_5$: We have $p_2(v) \geq -1 - 155\alpha$ because $d(v) = 5$ and $p_1(v) = -1$.

For vertices of degree 6 we have the following:

- $v \in X_6$: $p_2(v) = 0$ because $d(v) = 6$ and $p_1(v) = 0$.
- $v \in Y_6$: $p_2(v) = p_1(v) + \alpha = \alpha$ because Rule B gives a single value $\alpha$ from some $u$ to $v$, and by Lemma 48 such a vertex exists, if $v$ exists.

Note that the distributions of these $\alpha$ potentials are cost-neutral in total for $p_1$ and $p_2$.

Considering Rule A, for vertex $v$ and $d(v) \geq 7$, we can apply Lemma 49 and the above estimate for an upper bound

$$p_2(v) \geq (d(v) - 6) - \left\lfloor \frac{1}{2} d(v) \right\rfloor \cdot \frac{1}{4} - 31 d(v) \alpha \,. \tag{4.7}$$

So the remaining cases can be estimated by

- $v \in X \cup Y$ with $d(v) = 7$: $p_2(v) \geq \frac{1}{4} - 217\alpha$.
- $v \in X \cup Y$ with $d(v) \geq 8$: $p_2(v) \geq d(v)\left(\frac{7}{8} - 31\alpha\right) - 6$ by $\lfloor \frac{1}{2} d(v) \rfloor \cdot \frac{1}{4} \leq \frac{1}{8} d(v)$.

We choose $\alpha > 0$ so that $\frac{1}{4} - 217\alpha \geq \alpha$ and $d\left(\frac{7}{8} - 31\alpha\right) - 6 \geq \alpha$ holds for all $d \geq 8$. The first inequality is fulfilled for $\alpha = \frac{1}{218 \cdot 4} = \frac{1}{872}$ and this also fulfills the second inequality. Thus, we have shown that for $v \in Y$ $p_2(v) \geq \alpha$ holds. The same value for $\alpha$ also guarantess $p_2(v) \geq -3 - 93\alpha$ for all $v \in X$.

It remains to note that the distribution is cost-neutral also for Rule A. But this is clear because the sum of the potential added and retracted is the same. Altogether, Lemma 51 holds and the full conclusion can be drawn, which is $s_4(G) \geq \frac{1}{2712}$. □

**Exercise 20** *Present the precise strategies that stem from Theorem 46 and Theorem 47. Analyse the corresponding running time.*

# Chapter 5

# Geometric firefighting

In this chapter we would like to discuss geometric fireighting settings in the Euclidean plane. We assume that there is a fire source $s$ (a point in the plane) and the fire spreads in all directions with the same unit speed 1. This means that the fire can be considered to be an expanding circle, expanding with speed one over time.

The adversary of the fire expansions can build barriers with some speed $b$. In the geometric setting, a barrier is a curve in the plane with given start and endpoint. We consider different models. For example, we can assume that the barriers have to be build up successively one after the other or in parallel. We can also restrict the type of the barrier, barriers can be restricted to line segment or can be arbitrary curves. Additionally, we consider different types of environments.

## 5.1 Firefighting in a simple polygon

Let us assume that the fire spreads inside a simple polygon $P$ at a single point $s$. Let us further assume that a set of $m$ potential line segment barriers, $B = \{b_1, b_2, \ldots, b_m\}$, is given. Each barrier, $b_i$, has a start point and an end point $t_i$ on the boundary of $P$.

All barriers can be build with the same speed $b$. If $|b_i|$ is the length of the barrier, its *relative completion time* is $\frac{|b_i|}{b}$. We also assume that the barriers have to be constructed one after the other. During the construction of the barrier the fire should never reach the current end point while the barrier is constructed. Any barrier saves a certain amount of the total area of the polygon. The goal of the barrier construction is, that we would like to find a schedule for a valid construction of the barriers, that saves a maximal portion of the total area of the polygon. An example for a polygon, a fire source $s$ and a set of potential barriers is shown in Figure 5.1.

In contrast to the Intruder-Definition given in Section 1.1.1 we define the problem as a firefighter problems in a simple polygon.

*Geometric Firefighter Problem in a simple polygon*
**Instance:** A simple polygon, a fire that spreads from a given starting point $s \in P$ with speed 1, a set of $m$ line segment *barriers*, $b_1, b_2, \ldots, b_m$, which can be successively constructed with the same speed $B$.
**Output:** Compute a valid sequence of barriers that is constructed successively so that the area protected from the fire is maximized.

As already shown in Theorem 1 in Section 1.1.1 this problem is NP-hard in general. It can also be shown that it is approximation hard, which means that there is no polynomial time approximation scheme. Therefore we have to consider constant approximations for the optimal schedule.

## 5.2 A general approximation algorithm

In this section we describe a general approach for the approximation of a scheduling problem. We also relate it to the above firefighting problem.

We consider $m$ jobs $b_1, b_2, \ldots, b_m$ with precise latest starting time and duration. That is, the job $b_i$ has to be started before time $s_i$ and has a duration of $d_i$.

For a *valid schedule* of jobs after $n$ processing steps of an algorithm with $l_n$ entries, we have a consecutive list of jobs $J_n = (b_{n_1}, b_{n_2}, \ldots, b_{n_{l_n}})$ which can be scheduled one after the other. Let $s'_{n_k} \leq s_{n_k}$ be the precise starting time of the scheduled jobs. We require $\sum_{k=1}^{j} s'_{n_k} + d_{n_k} \leq s_{n_{j+1}}$ for $j = 1$ to $l_n - 1$. The next job ha to be started before its starting option ends. This means that the jobs can be started one after the other without conflicts of the starting times.

Each job $b_i$ will contribute with a profit $A_i$ to an overall profit $A$. Profit $A_i$ will be attained, if the job is scheduled. These profits might *overlap* for different jobs. Thus, the profit of a job $b_i$ may change over time, if another job $b_j$ was scheduled before that also covers parts of the profit that can be covered by $b_i$.

For a current schedule of jobs $J_n$ we denote the current profit of a job $b_j \notin J_n$ by $A_j(J_n)$. This depends on the overlapping. More precisely for $J_n = (b_{n_1}, b_{n_2}, \ldots, b_{n_{l_n}})$ and $b_j \notin J_n$ we define

$$A_j(J_n) := A_j \setminus \left( \bigcup_{b_{n_k} \in J_n} A_{n_k} \right).$$

(Note that here $A_{n_k}$ is the initial overall potential profit of $b_{n_k} \in J_n$ but this profit is always covered by the profit of $J_n$, even though some of the profit $A_{n_k}$ was intermediately already covered by another $b_l$ in $J_n$.)

We start with an empty schedule $J_0$ and successively process one of the remaining jobs. This job will never be processed any more after that. So for $m$ jobs we are done after $m$ processing steps.

For the scheduling algorithm and a current schedule $J_n$ after $n$ steps we sort all non-rejected and non-scheduled jobs by decreasing relative profits $\frac{A_j(J_n)}{d_j}$. The profit of the job is considered in relation to its duration. We apply the following *GlobalGreedy* procedure to the current job $b_j$ with maximal $\frac{A_j(J_n)}{d_j}$:

1. If a job $b_j$ with current largest value $\frac{A_j(J_n)}{d_j}$ can be scheduled somewhere in $J_n$ it is inserted into $J_n$ for obtaining $J_{n+1}$.

2. If the job cannot be scheduled because some of the jobs overlap with its possible execution interval, we check whether we can delete such (consecutive) jobs out of $J_n$ with the following property: For a given constant $\mu < 1$, the profit of the deleted jobs is smaller than $\mu$ times the profit $A_j(J_n)$ of job $b_j$. Additionally, after deleting the jobs, job $b_j$ can be scheduled. All jobs that have been deleted will never be considered again. We build $J_{n+1}$ by inserting $b_j$ to the remaining list of $J_n$. There might be some options for such sequences.

3. If such a sequence of jobs does not exist, job $b_j$ is rejected and will never be considered again. $J_{n+1} := J_n$.

For convenience for the profits in the procedure and also for the jobs we define corresponding colors. In the beginning all potential profits are colored red. Profits and jobs that will be scheduled by rule 1. or rule 2. are colored green. A profit (and a job) that was colored green in the schedule and is rejected later by rule 2. will be colored grey. For a schedule $J_n$ let $J'_n$

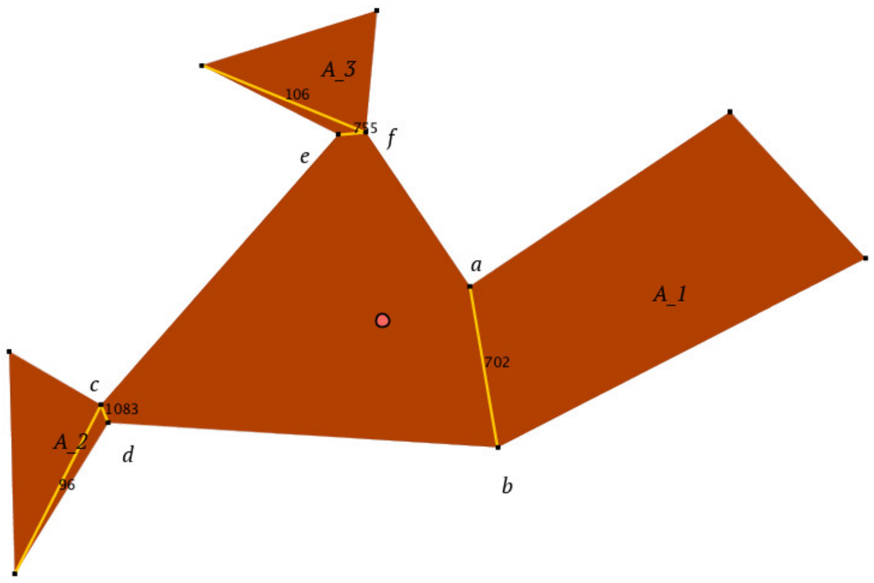


Figure 5.1: Considering the three barriers $b_1 = (a, b)$, $b_2 = (c, d)$ and $b_3 = (e, f)$ with $A_1 = 1053$, $A_2 = 162.45$ and $A_3 = 188.75$ and speed $b = 2$. We first the barriers by priority $p_1 = 702 = 2A_1/|b_1|$, $p_2 = 1083 = 2A_2/|b_2|$ and $p_3 = 755 = 2A_3/|b_3|$. The intermediate schedule $J_2 = (b_2, b_3)$ blocks the construction of $b_1$. Since $\mu \cdot A_1 > A_3$ and $(|b_1| + |b_3|)/2 < d(s, a)$ holds we delete $b_3$ and insert $b_1$.

denote the list of all jobs that have been inserted during the construction of $J_n$. The set $J'_n$ also contains the deleted barriers of color grey and the current barriers of color green.

The above algorithm is a general scheme for a corresponding scheduling problem. We would like to apply it to our geometric firefighter problem and first give a very simple example. In Figure 5.1 there is a fire source and a set of potential barriers which are diagonals.

In this setting for convenience we require that the barrier has to be fully constructed before the fire reaches any point of it. This requirement is not necessary in general but by considering the closest point from $b_i$ to $s$ and the duration $d_i$ of the barrier we can easily compute the a constraint for $b_i$. In general the fire should not have reached the current construction point of the barrier.

We consider only the barriers $b_1 = (a, b)$, $b_2 = (c, d)$ and $b_3 = (e, f)$ as depicted in Figure 5.1, where each barrier will be started with its first vertex. We assume that the building speed is $b = 2$. Each barrier subdivides the polygon into two parts and potentially would save the part $A_i$ that does not contain the fire source. Let $|b_i|$ denote the lenght of each $b_i$, we assume $|b_1| = 3$, $|b_2| = 0.3$ and $|b_3| = 0.5$. The distance $d_P(s, a)$ from $s$ to $a$ in $P$ is assumed to be 1.8. We further have calculated $A_1 = 1053$, $A_2 = 162.45$ and $A_3 = 188.75$ which are the profits of the barriers in the beginning. For the scheduling algorithm we order them relatively by priority $p_i = \frac{2A_i}{|b_i|}$ because $d_i = \frac{|b_i|}{2}$ is the duration for constructing $b_i$. The values in the figure denote the corresponding priority values $p_i$ This means that $p_1 = 702$, $p_2 = 1083$ and $p_3 = 755$ holds. We assume that $\mu = 0.2$.

The scheduling algorithm first chooses $b_2$ and $b_3$ in this order because of the priorities and of the fact that both barriers can be indeed scheduled in this order. The fire will not reach $c$ and $e$ after $\frac{0.3+0.5}{2} = 0.4$ time steps. Because $(|b_1| + |b_2| + |b_3|)/2 = 1.9$ exceeds the distance from $s$ to $a$ in $P$, in the third step of the process we would like to apply rule 2. Since $\mu \cdot A_1 > A_3$ holds, we delete $b_3$ out of the schedule and build $b_1$ instead. The final schedule is $J_3 = (b_2, b_1)$ which saves $A_2$ and $A_1$. The resulting corresponding color scheme is shown in Figure 5.2.

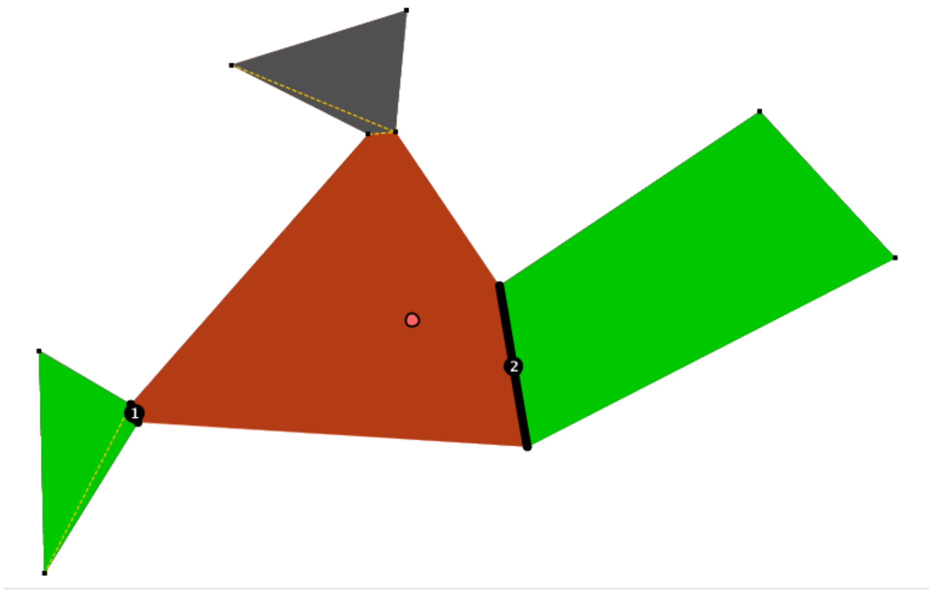


Figure 5.2: The color scheme of the scheduling algorithm for Figure 5.1. After $b_1$ exchanges $b_3$ the profit $A_3$ is colored grey. The part that was not protected remains red-colored. The barriers are build in the order 1. and 2. as depicted.

The following Lemma relates the green profit with the grey profit, so that we do not loose to much by deletions. By rule 2. a profit gets color grey, if it was deleted for insertion of a green profit larger than $\mu$ times the grey profit. Therefore we can prove the following relationship. Let $J_m$ denote the final schedule for a set of jobs $b_1, \ldots, b_m$ and let $J_n(grey)$ and $J_n(green)$ denote the profits that are colored green and grey during the construction of $J_n$.

**Lemma 52** *By the* GlobalGreedy *procedure we finally have*

$$J_m(grey) \leq \frac{\mu}{1-\mu} J_m(green)\,. \tag{5.1}$$

**Proof.** By induction on number of jobs processed during GlobalGreedy. The lemma holds in the beginning for the empty schedule $J_0$. Let us assume that the lemma holds after $n$ steps for $J_n$. Consider step $n+1$.

If no job is deleted by rule 1. or rule 3. only the green profit will increase from $J_n$ to $J_{n+1}$ that is

$$J_n(grey) = J_{n+1}(grey) \leq \frac{\mu}{1-\mu} J_n(green) \leq \frac{\mu}{1-\mu} J_{n+1}(grey)\,.$$

If by rule 2. some of the jobs where cancelled out of $J_n$ for inserting $b_j$ in the $n+1$ step. The overall profit of the cancelled jobs (that becomes grey now) is smaller than $\mu$ times $A_j(J_n)$, the profit of $b_j$ right now. We conclude

$$\begin{aligned}\frac{\mu}{1-\mu} J_{n+1}(green) &\geq \frac{\mu}{1-\mu}(J_n(green) + (1-\mu)A_j(J_n)) \\ &\geq \frac{\mu}{1-\mu} J_n(green) + \mu A_j(J_n) \geq J_n(grey) + \mu A_j(J_n) \geq J_{n+1}(grey),\end{aligned}$$

where we used the induction hypothesis for the secon $\geq$ sign in the second line. $\square$

For the relationsship between the profit of the final schedule $J_m$ and an optimal schedule $J_{\text{opt}}$ we make use of another color, blue.

For the profits, all profit elements in the optimal schedule $J_{\text{opt}}$ that are still have color red after the application of $J_m$ will be colored blue. So none of the grey and green profits of $J_m$ become blue. The blue profit part depends on $J_m$ and $J_{\text{opt}}$, let us denote this profit, by $J_m(\mathit{blue})$. It is clear that

$$|J_{\text{opt}}| \leq J_m(\mathit{blue}) + J_m(\mathit{green}) + J_m(\mathit{grey})\,.$$

holds, we would like to express $J_m(\mathit{blue})$ in terms of $J_m(\mathit{green})$ and $J_m(\mathit{grey})$.

We can assume that the jobs of $J_m$ are colored green and grey by the schedule. Likewise, we assign the blue property to the jobs of $J_{\text{opt}}$. The blue property is assigned to the first job of $J_{\text{opt}}$ that covers a portion of the blue profit. Such a job cannot be green or grey, because otherwise it cannot participate at a the red colored profit at the end of the schedule $J_m$. Thus, all three color classes of jobs are pairwise disjoint.

As already mentioned $J'_m$ denote the list of all jobs that have been inserted during the construction of $J_m$. Any green and grey jobs of $J'_m$ has obtained a unique execution time, that will not be changed by the process afterwards (although grey jobs will no longer be scheduled at the end).

We make use of a scheme that pays money from green and grey jobs to the blue jobs w.r.t. the execution intervals of the jobs $b_i \in J'_m$ and $b_j \in J_{\text{opt}}$.

1. If the execution interval of $b_j \in J_{\text{opt}}$ is fully included in the execution interval of $b_i \in J'_m$, the job $b_i$ pays its green or grey profit times $\frac{d_j}{d_i} < 1$ to $b_j$.
2. If the execution interval of $b_j \in J_{\text{opt}}$ overlaps with the execution interval of $b_i \in J'_m$, the job $b_i$ pays its green or grey profit times $\frac{1}{\mu}$ to $b_j$.

The payment of each green and grey job to the blue jobs has an upper bound w.r.t. their profit.

**Lemma 53** *Any single green or grey job from $J'_m$ pays in total at most $1 + \frac{2}{\mu}$ times its profit to the blue jobs.*

**Proof.** A job $b_i \in J'_m$ has a fixed execution interval $I_i$ and the start- and endtime of the interval can only be located inside at most two intervals of jobs from $J_{\text{opt}}$ which are also executed successively. This gives at most 2 times $\frac{1}{\mu}$ the profit of $b_i$. For all jobs from $b_j \in J_{\text{opt}}$ fully inside the execution interval $I_i$ of $b_i$ a portion $\frac{d_j}{d_i}$ of $b_i$'s profit is payed for $d_j < d_i$. In total this can only sum up to at most 1. □

The other way round, any blue job is payed by at least its own profit from the green and grey jobs.

**Lemma 54** *Any single blue job from $J_{opt}$ achieves at least a payment in the size of its blue profit from the green and grey jobs.*

**Proof.** We consider a blue job $b_j \in J_{\text{opt}}$. This job was rejected at some step $k+1$ and let $J_k = (b_{k_1}, b_{k_2}, \ldots, b_{k_{l_k}})$ be the corresponding schedule of the algorithm. For the final exection time interval of $b_j \in J_{\text{opt}}$ consider an consecutive subset $\overline{J_k}$ of the jobs of $J_k$ of mimimal size whose intervals overlap with the execution time of $b_j$, The deletion of $\overline{J_k}$ would allows to schedule $b_j$.

The total sum profit of the jobs of $\overline{J_k}$ is larger than $\mu$ times the current profit $A_j(J_k)$ of $b_j$ which is currently red. Some of this red part $A_j(J_k)$ will become blue at then end but may be not all of it. In any case $\mu$ times the current red part $A_j(J_k)$ of $b_j$ is larger than $\mu$ times the final blue part of $b_j$. On the other hand, the job $b_j$ had priority less than all previously inserted jobs of $J_k$ which is $\frac{A_i(J_k)}{d_i} \geq \frac{A_j(J_k)}{d_j}$.

We consider the size of $\overline{J_k}$. If $|\overline{J_k}| = 1$ holds for the single job, say $b_i$, the execution time of $b_j \in J_{\text{opt}}$ might be fully inside the execution time of $b_i$, which gives

$$A_i(J_k)\frac{d_j}{d_i} \geq A_j(J_k)\frac{d_i}{d_i} = A_j(J_k)$$

where $A_i(J_k)$ will be the green or grey profit of $b_i$ for the final $J_m$ and $A_j(J_k)$ is larger than the blue profit of $b_j$.

For $|\overline{J_k}| \geq 1$, the execution time interval of $b_j$ is not fully inside an execution interval of a job in $\overline{J_k}$, but the execution interval of $b_j$ overlaps with all execution intervals of the jobs in $\overline{J_k}$, This gives:

$$\frac{1}{\mu}\sum_{b_i \in \overline{J_k}} A_i(J_k) \geq \frac{1}{\mu}(\mu A_j(J_k)) = A_j(J_k)\,.$$

In any case a blue job gets paid with at least its profit form the green and grey jobs. □

By the above Lemmata we can now estimate the profit of $J_m(\mathit{blue})$ in comparison to $J_m(\mathit{green})$ and $J_m(\mathit{grey})$. The blue profit is upper bounded by the payment from green and grey jobs, in turn the payment of green and grey jobs is upper bounded by the profits of green and grey jobs.

We have

$$J_m(\mathit{blue}) \leq \left(1 + \frac{2}{\mu}\right)(J_m(\mathit{green}) + J_m(\mathit{grey}))\,. \tag{5.2}$$

By Lemma 52 we can now express the profit of $J_{\text{opt}}$ in terms of $J_m(\mathit{green})$, only.

$$\begin{align}
|J_{\text{opt}}| &\leq J_m(\mathit{blue}) + J_m(\mathit{green}) + J_m(\mathit{grey}) \tag{5.3}\\
&\leq \left(2 + \frac{2}{\mu}\right)(J_m(\mathit{green}) + J_m(\mathit{grey})) \tag{5.4}\\
&\leq \frac{2(\mu+1)}{\mu}(J_m(\mathit{green}) + \frac{\mu}{1-\mu}J_m(\mathit{green})) \tag{5.5}\\
&\leq \frac{2(\mu+1)}{\mu}\frac{1}{1-\mu}J_m(\mathit{green}) \tag{5.6}\\
&\leq 2\frac{\mu+1}{\mu(1-\mu)}J_m(\mathit{green}) \leq 2\frac{\mu+1}{\mu(1-\mu)}|J_m|\,. \tag{5.7}
\end{align}$$

The factor $f(\mu) := 2\frac{\mu+1}{\mu(1-\mu)}$ is minimized for $\mu = \sqrt{2}-1$ and gives $f(\mu) = 6 + 4\sqrt{2} \approx 11.657$ and our final result.

**Theorem 55** *For the geometric firefighter problem inside a simple polygon with non-intersecting barriers there is an approximation algorithms that saves at least $\frac{1}{6+4\sqrt{2}} = \frac{3}{2} - \sqrt{2} \approx 0.086$ times the area of the optimal barrier solution.*

**Exercise 21** *Verify the computation of the optimization value $\mu = \sqrt{2}-1$ and $f(\mu) = 6+4\sqrt{2} \approx 11.657$.*

The result is restriced to non-intersecting barriers. Additionally, we require that any point in the polygon that can be covered, can already be covered by a single barrier. Otherwise some barriers $b_1, b_2, \ldots, b_l$ could help each other for covering a part of the polygon that is not covered

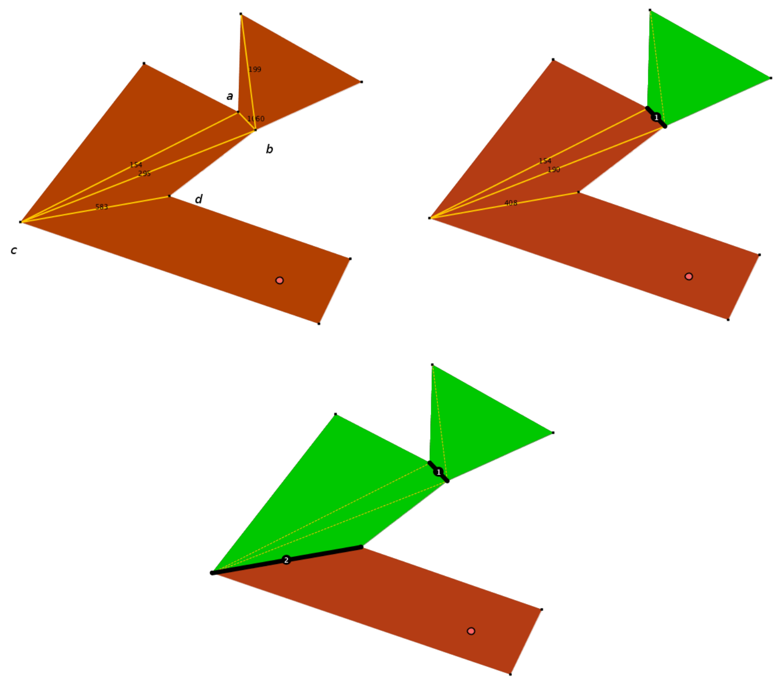


Figure 5.3: After the usage of barrier $(a, b)$, the priority of barrier $(c, d)$ decreases because $(a, b)$ already protects parts of the area related to $(c, d)$.

by a single barrier. In this case we can protect new areas $A'$ which are not related to single barriers. This is obviously not represented in the algorithm and the analysis.

In principle the algorithm could also run for intersecting barriers but there is one major problem. The execution time for building a barrier is not independent from the order of the barrier constructed so far. A barrier might have a release times that depends on the barriers constructed before. For example a barrier $b_1$ blocks the fire and extends the release time of a barrier $b_2$ that intersects $b_1$.

So there might be a barrier of the optimal solution, that contributes to the optimal profit with an arbitrary large amount of its profit but cannot be scheduled in the approximation since it depends on the construction of a very less important (w.r.t. area profit) barrier. The analysis might fail.

**Exercise 22** *Construct an example, where the construction of a barrier $b_2$, depends on the construction of a barrier $b_1$.*

**Exercise 23** *If intersections are allowed, where is the main problem in the proof of the above approximation result?*

Finally, we end the section with an example where the profit of a job is reduced by overlapping, as shown in Figure 5.3.