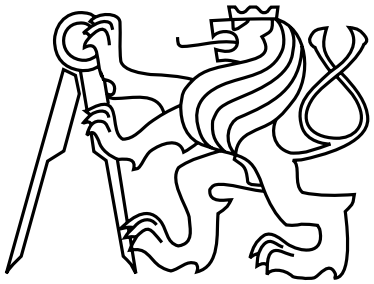
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Optimization of incident response vehicles placement

Bc. Lukáš Novotný

Supervisor: Ing. Ondřej Vaněk, Ph.D.

Study Programme: Open Informatics, Master

Field of Study: Artificial Intelligence

May, 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Novotný** Jméno: **Lukáš** Osobní číslo: **420018**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimalizace rozmístění zásahových vozidel bezpečnostní služby

Název diplomové práce anglicky:

Optimization of incident response vehicles placement

Pokyny pro vypracování:

Security companies which offer services in building protection and alarm detection also complement their services with alarm response. I.e., they operate a fleet of vehicles distributed around the region and in case an alarm is detected, a response vehicle is dispatched to investigate the alarm. The companies face the problem of vehicle allocation, i.e., how they should allocate the vehicles prior any alarm is detected and how the vehicles should behave in case one or more alarms are detected. The student should design and algorithm able to aid in the process of vehicle allocation and response management. Specifically, the student should do the following:
1. Understand the problem of fleet management in security companies
2. Study related literature to understand various models and approaches for vehicle allocation
3. Design an algorithm able to optimally allocate vehicles and maximize the coverage of buildings protected
4. Extend this algorithm to work efficiently in case one or more alarms are detected
5. Evaluate the algorithms on a set of real-world and synthetic scenarios

Seznam doporučené literatury:

[1] Li, X., Zhao, Z., Zhu, X. and Wyatt, T., 2011. Covering models and optimization techniques for emergency response facility location and planning: a review. Mathematical Methods of Operations Research, 74(3), pp.281-310.
[2] Yue, Y., Marla, L. and Krishnan, R., 2012, July. An Efficient Simulation-Based Approach to Ambulance Fleet Allocation and Dynamic Redeployment. In AAAI.
[3] Karasakal, O. and Karasakal, E.K., 2004. A maximal covering location model in the presence of partial coverage. Computers & Operations Research, 31(9), pp.1515-1526.
[4] Bošanský, B., Lisý, V., Jakob, M. and Pěchouček, M., 2011, May. Computing time-dependent policies for patrolling games with mobile targets. In The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3 (pp. 989-996). International Foundation for Autonomous Agents and Multiagent Systems.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ondřej Vaněk, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **20.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

Ing. Ondřej Vaněk, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Aknowledgements

First of all, I would like to thank my supervisor Ing. Ondřej Vaněk, Ph.D. for his valuable advices and willingness to consult approaches taken in this thesis. Secondly, I would like to thank Roman Novotný, certified Jablotron partner, for offering extensive knowledge of the domain background. Lastly, I would like to thank my family, my parents and my sister, for their support and patience during my studies.

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 25, 2018 .....

Abstract

Private security companies offer a service of incident response vehicle deployment to inspect the cause of an alarm in the client's building. Due to the increasing popularity of this service, security companies are facing a fleet management problem. They need to optimize idle positions of incident vehicles in a patrolled region such that the coverage of protected buildings is maximized while keeping the arrival time reasonably low. In this thesis, we present two solutions to vehicle allocation combining shortest path search and integer linear programming. First assigns a permanent idle position to each vehicle and the second one is able to dynamically change idle positions depending on the current operation of the whole fleet. An event-based simulator is used to evaluate these solutions both on synthetic and real-world scenarios. Results indicate the effectiveness of our solution and advantages of dynamic reallocation.

Keywords: fleet management, static resource allocation, dynamic resource allocation, maximum coverage problem

Abstrakt

Soukromé bezpečnostní společnosti nabízejí službu nasazení zásahového vozidla, které zkontroluje příčinu poplachu v budově klienta. Vzhledem k narůstající popularitě této služby se bezpečnostní společnosti potýkají s problémem řízení vozového parku. Potřebují optimalizovat rozmístění vozidel po hlídaném regionu tak, aby bylo pokrytí chráněních budov co největší a dojezdový čas byl přiměřeně nízký. V této práci uvádíme dvě různá řešení pro optimalizaci rozmístění vozidel, které kombinují hledání nejkratších cest a celočíselné lineární programování. První přiděluje každému vozidlu stálou pozici a druhé je schopné měnit pozice vozidel v čase v závislosti na aktuálním stavu celého vozového parku. K vyhodnocení těchto řešení na umělých a reálných datech používáme simulátor. Výsledky ukazují účinnost našich řešení spolu s výhodami dynamické realokace.

Klíčová slova: řízení vozového parku, statické přidělování zdrojů, dynamické přidělování zdrojů, problém maximálního pokrytí

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Private security companies nowadays offer affordable complete solutions for building security. The main purpose of such a solution is to trigger an alarm when a burglar illegally enters the property. However, it is not forbidding him from stealing valuable items and successfully escaping. This is a task for incident response vehicles. Their goal is to arrive at the location of an alarm as soon as possible, inspect the cause of an alarm, and potentially capture the criminal and handle him to the police. Nonetheless, it is not always an illegal activity that triggers the alarm, as it can also be activated by a fire, gas leak, water leak or other natural disasters, while inhabitants are away. In these scenarios, time also plays a crucial role when it is desired to minimize the damage. Since the service of building protection is becoming increasingly more popular, private security companies are faced with a problem of choosing idle positions for their incident response vehicles, such that they maximize the number of protected buildings while keeping the arrival time reasonably low.

In this work, we are interested in the optimization of idle incident vehicle allocation. Having said that, we are not particularly interested in the quality of the building's alarm system, distinguishing between true positive or false positive alarms and the inspection process. Given a set of buildings and a set of potential idle vehicle locations, our task is to manage a fleet of vehicles – that is to distribute them between potential idle locations and potentially reallocate them to a different idle location – in a way that the coverage of protected buildings is maximized and the arrival time of a vehicle to an alarm event is reasonably low.

A similar problem of resource allocation is researched in the domain of Emergency Medical Services. Their task is to allocate a set of stationary medical facilities or a set of mobile ambulances in such a way, that their response time to crucial incidents is minimized or the covered area is maximized. These problems are formulated as mathematical programs, especially linear programs. However, they either have too many hard constraints that are not satisfiable with a small fleet of vehicles or do not optimize both covered area and arrival time.

We present two approaches to optimize the idle incident vehicle allocation formulated as linear integer programs. Moreover, we introduce an event-based simulator, that evaluates the coverage of protected buildings and response time. First presented method associates each vehicle with its own idle location which does not change in time. Once the vehicle is deployed and finishes the inspection, it returns to its original idle place. Second presented method,

motivated by an intuition that temporal repositioning of available idle vehicles provides a better coverage, recomputes positions of idle vehicles each time a new alarm is detected or when an inspection of an alarm is completed. Both of those approaches are firstly analyzed in a small, controlled synthetic environment. Finally, we use these approaches to simulate a real-world scenario taking place in the city of Prague.

Situations with a varying number of vehicles and varying frequency of alarm events are explored. We conclude, that when the frequency of alarm events is high, such that the fleet of vehicles is too small to handle them all, there is no real benefit in continuous reallocation. Similarly, when the frequency is low, such that large fleet can handle them with ease, there is not much room for improvement via dynamic reallocation. However, there is a combination of fleet size and alarm frequency, where the dynamic reallocation helps significantly.

1.1 Thesis goals

The overall goal of this thesis is to design and evaluate an algorithm that is able to optimally allocate vehicles and maximize the coverage of protected buildings. Additionally, we also design and evaluate a dynamic scenario, where idle vehicles reallocate to better positions when an alarm event is detected. In order to do so, we solve related tasks in chapters as described below.

First, in chapter 2 we explore the building security domain background, which consists of securing a building with an alarm system, detecting when this system goes off, and finally dispatching an incident response vehicle to inspect the cause.

Secondly, in chapter 4 we study related literature on various approaches to vehicle allocation that usually come from Emergency Medical Services domain. Their goal is to minimize the response time of emergency vehicles. Additionally, we add a small introduction to patrolling games domain which formulates its problems as a two-player game.

Thirdly, in chapter 5 we define the allocation problem as a maximum coverage problem and present both static and dynamic integer linear programs, that are used to allocate incident response vehicles. We also present an event-based simulation that is used to evaluate the quality of the allocation.

Finally, in chapter 7 we use this simulation to evaluate both of our approaches on a set of small, controlled, synthetically generated data and real-world data consisting of a Prague road network.

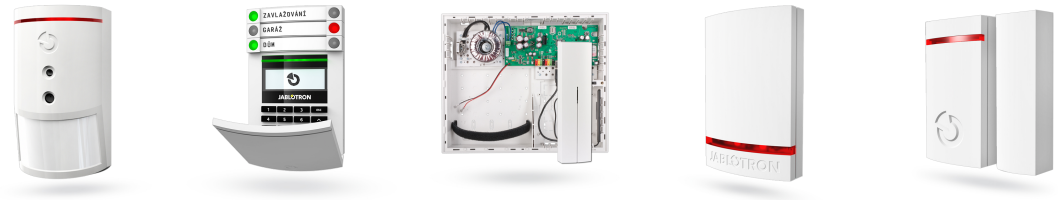
Chapter 2

Domain background

Criminality is undoubtedly a big worldwide concern and many resources are expended to reduce it. However, this work is interested only in burglary – an illegal entry into a building with intent to commit a crime, especially theft. Indeed, break-ins are not negligible crime offence, for instance, the Police of the Czech Republic released an annual 2017 criminality statistics [1] which registers 24 127 committed burglaries with only 5 714 marked as closed. This results in only 23.7% successful solve rate, which is together with other thefts one of the lowest one. In macro perspective, the statistical office of the European Union, Eurostat, states in its statistic report [2] that in the year of 2015 total number of 2 387 770 burglary offences were committed in EU-28. That year it was the lowest one since 2008 with 8.8% decrease.

The high percentage of unsolved housebreakings suggests that prevention might be more valuable than relying on the criminal's capture and that is not considering the potential loss of priceless personal belongings. Fortunately, private security companies with a strong background and many years of practice offer professional services related to the security of buildings. Furthermore, the rapid growth of this industrial sector opens affordable complete solutions for average citizens. One of the world's leading international security solutions group – as stated by data journalist Niall McCarthy [3] – is G4S[1] that also has its share on the Czech market together with local competitors such as Jablotron[2] or D.I.Seven[3].

Following sections briefly explain products sold by these companies. Firstly, section 2.1 describes how a typical professional building security system looks like, how an alarm is triggered and how the user is notified. Latter section 2.2 explains how a security company monitors your property and the protocol they follow once an alarm occurs. Finally, section 2.3 explains the deployment of an incident vehicle that inspects the cause of an alarm on the spot.



(a) JA-120PC (b) JA-114E (c) JA-106KR-3G (d) JA-111A (e) JA-111M

Figure 2.1: Various parts of building security system

JA-120PC 2.1a is an area motion detector with RGB camera to visually inspect the cause of the alarm.
JA-114E 2.1b is an authentication module with a keypad, LCD display and RFID reader.
JA-106KR-3G 2.1c is a control module with built-in 3G / LAN communicator and a radio module.
JA-111A 2.1d is an external siren designed to sound alarms and signal activation or deactivation of system outputs.
JA-111M 2.1e is a two-part magnetic detector designed for window or door opening detection.

Image and information source: [4]

2.1 Building security solutions

A typical modular building security solution, as product catalogue [4] of Jablotron suggests, consists of a wide variety of detectors, authentication devices and a control module 2.1c that connects the whole system together. Some examples of detectors are a motion sensor 2.1a that detects motions in the whole room, an opening detector 2.1e that triggers once a door, or a window is opened, a glass break sensor capable of recognizing a window-breaking sound or even not break-in related sensors such as flood or smoke detectors. Additionally, sensors may have integrated camera 2.1a, that takes a photo of the cause.

Once the system is activated, these devices will trigger an alarm once a monitored event occurs. This commonly means a loud sound siren 2.1d is set off to warn everyone about the intrusion and to scare the burglar. Additionally, property inhabitants are notified via various communication channels like SMS, phone call, email or push notification. To switch off this security system one requires a special remote control, control chip keychain or pin code that was previously paired with the authentication module 2.1b.

Unfortunately, currently used solutions cannot distinguish whether the source of an alarm is an actual threat. This means that the user may unintentionally set off their alarm if not careful. That usually happens when they forgot to deactivate the alarm, or they take too long to exit the building when they activate the alarm. Such activation is called false positive – it was triggered but there was no actual threat. On the other hand, a true positive is an alarm triggered by a burglar, fire caused by forgotten stove being on or a flood caused by a broken pipe et cetera.

After an alarm goes off it is useful to have a couple of CCTV cameras in place to immediately inspect the cause in real time and high quality. This is the fastest way to differentiate between false positive and true positive. However due to higher cost, they are

[1] <http://www.g4s.com/>, accessed on 3-April-2018
[2] <www.jablotron.com>, accessed on 3-April-2018
[3] <www.diseven.cz>, accessed on 3-April-2018

(a) Incident response vehicle

(b) Vehicle with crew

Figure 2.2: Incident response vehicle

Incident response vehicle is ready to be deployed at any time and its goal is to arrive at the location of an alarm and inspect the cause. The Kruh project has 215 of such vehicles all around the Czech Republic with the average arrival time of 10 minutes. Over the past 12 months, they were involved in around 85 000 interventions.
Image sources: <http://www.bezpecnostnicentrum.cz/en#krok4>, <https://www.diseven.cz/tiskove-centrum/> accessed on 3-April-2018
Information source: <http://www.zasahovasluzba.cz/o-projektu/>, accessed on 3-April-2018

not a usual part of home security systems but rather industrial, business or community buildings systems.

2.2 Alarm receiving centres

One of the latest service offered by previously mentioned companies [5], [6], are alarm receiving centres (ARC) that are connected to the control module of an alarm system and constantly monitor its state. Modern digital communication formats allow transmission of status signals, a full range of events – for example, photos or camera feed – or combination of both.

This information, once obtained by the ARC, is evaluated. If a suspicious situation occurs in the monitored building a standard protocol is initiated. This protocol obviously differs by the company, but in general, the first action is to inform the customer about the event. This helps to determine whether the alarm is false positive or true positive. Once a true positive alarm is confirmed by the client, or when the client is unreachable for some period, a mobile patrol is sent to inspect the location. Finally, a report is handed to the client and other designated parties.

2.3 Incident response vehicle deployment

Incident response vehicles – depicted in figure 2.2 – are deployed by ARC to inspect the cause of an alarm. Once at the location, the vehicle crew checks the perimeter for any signs of illegal entry including the fence, windows, doors or other possible entry points. If an unauthorized entry is confirmed the property is guarded until the police arrive. This also includes a possible capture of the intruder.

Since time plays such a crucial role, the ARC operators have only a couple of minutes to check with the owner of monitored property whether the alarm is true positive or false positive, to possibly prevent unnecessary and costly intervention. Another huge time factor is the route from current incident response vehicle location to the event alone.

One of the main concerns of private security companies and projects like Kruh[4] is to maximize the covered area while minimizing the travel time to the event. The common practice is, that inactive vehicles are waiting at a specific location – headquarters – and once engaged in an intervention event, they must fulfil all obligations before they can respond to other events. When multiple incidents happen in an area patrolled by a single vehicle, the response is not immediate but rather processed in a queue like order.

2.4 Area for improvement

This work explores a possible limitation of this system. We believe that computer aided design with appropriate algorithm will choose better idle locations of response vehicles that maximizes the covered area while minimizing the arrival time. Furthermore, after the dispatch of a single vehicle, the remaining idle vehicles may optimize their position to cover the newly uncovered sector.

[4] <http://www.zasahovasluzba.cz>, accessed on 3-April-2018

Chapter 3

Technical Background

In this chapter, we present mathematical problems we are dealing with in this work – namely maximum coverage problem in section 3.1 and shortest paths in section 3.2.1. Mathematical notation of these problems and notation of graphs in section 3.2 is given an is kept the same throughout this work. Additionally, each problem is placed in its corresponding complexity class.

3.1 Maximum coverage problem

Since we formulate the optimal placement of incident response vehicles as a maximum coverage problem, we first present it in a general version, which is defined as follows. Given the finite universe $U = \{u_1, u_2, \ldots, u_n\}$, a finite family of finite sets $S = \{S_1, S_2, \ldots, S_m\}$, where each set contains elements from the universe $S_{i=1,\ldots,m} \subseteq U$, and a positive integer k, the goal is to select at most k sets from S

$$S' \subseteq S, |S'| \leq k,$$

such that the size of their union $|\bigcup_{S'_i \in S'} S'_i|$ is maximized. This union of sets S'_i cover elements from U and since we are trying to maximize the number of covered elements, this problem is named maximum coverage problem. In this work, we formulate this problem using following integer linear program.

$$max \quad \sum_{j=1}^{n} y_j \tag{3.1a}$$

$$\text{subject to} \quad \sum_{i=1}^{m} x_i \leq k \tag{3.1b}$$

$$\sum_{i:u_j \in S_i} x_i \geq y_j \qquad j = 1, \ldots, n \tag{3.1c}$$

$$x_i \in \{0, 1\} \qquad i = 1, \ldots, m \tag{3.1d}$$

$$y_j \in \{0, 1\} \qquad j = 1, \ldots, n \tag{3.1e}$$

This linear program has $m + n$ binary decision variables. Binary variable x_i (3.1d) determines, whether S_i is present in S'

$$x_i = \begin{cases} 1 & S_i \in S' \\ 0 & S_i \notin S', \end{cases}$$

while the binary variable y_j (3.1e) determines, if the element u_j is covered

$$y_j = \begin{cases} 1 & u_j \in \bigcup_{S'_i \in S'} S'_i \\ 0 & u_j \notin \bigcup_{S'_i \in S'} S'_i. \end{cases}$$

Moreover, the linear program has $n+1$ constraints. Constraint (3.1b) limits the number of selected sets in the cover $|S'| \leq k$, and n constraints of type (3.1c) saying that if element u_j is covered ($y_j = 1$), at least one S'_i must be selected, such that $u_j \in S'_i$. Finally, the objective function (3.1a) maximizes the number of covered elements.

In the following section, we prove that this maximum coverage problem belongs to $\mathcal{NP}-hard$ complexity class. The proof is done by a polynomial reduction from set cover, which is in $\mathcal{NP}-complete$ class [7].

3.1.1 Complexity class

In this section, we prove that the maximum coverage problem defined above is $\mathcal{NP}-hard$, but first, let us state the following lemmas about $\mathcal{NP}-hard$ problems and polynomial reductions.

Lemma 1 ($\mathcal{NP}-hard$ problems)**.** *If we know that any $\mathcal{NP}$ problem polynomially reduces to a problem $\mathcal{U}$ (or we know that there is a $\mathcal{NP}-$complete problem that polynomially reduces to $\mathcal{U}$), then we say that $\mathcal{U}$ is $\mathcal{NP}-hard$. Note that this means that $\mathcal{U}$ is at least as difficult as all $\mathcal{NP}-complete$ problems. [8, 1.8.7]* □

Lemma 2 (Reductions and polynomial reductions)**.** *Given two decision problems $\mathcal{U}$ and $\mathcal{V}$. We say that a problem $\mathcal{U}$ reduces to a problem $\mathcal{V}$, if there is an algorithm (a program for RAM, a TM) $\mathcal{M}$ that for every instance $\mathcal{I}$ of $\mathcal{U}$ constructs an instance $\mathcal{I}'$ of $\mathcal{V}$ such that*

$$\mathcal{I} \text{ is a YES-instance of } \mathcal{U} \text{ iff } \mathcal{I}' \text{ is a YES-instance of } \mathcal{V}.$$

The fact that $\mathcal{U}$ reduces to $\mathcal{V}$ is denoted by

$$\mathcal{U} \lhd \mathcal{V}.$$

Moreover, if the algorithm $\mathcal{M}$ works in polynomial time, then we say that $\mathcal{U}$ polynomially reduces to $\mathcal{V}$, and we denote it

$$\mathcal{U} \lhd_p \mathcal{V}.$$

Roughly speaking, $\mathcal{U} \lhd \mathcal{V}$ means that $\mathcal{U}$ is not more difficult than $\mathcal{V}$. [8, 1.8.1] □

If we can polynomially reduce some $\mathcal{NP}-complete$ problem to the maximum coverage problem, then by lemma (1) we prove, that maximum coverage problem is $\mathcal{NP}-hard$. For this purpose we chose the decision variant of set covering problem, that was proven by Richard M. Karp [7] to be $\mathcal{NP}-complete$. He defined the problem as follows:

> Given a finite family of finite sets R and a positive integer l, the goal is to determine, whether a subfamily $T \subseteq R$ with size bound $|T| \leq l$ exists, such that $\bigcup T = \bigcup R$.

Finally, following lemma (2), we formulate a YES-instance $\mathcal{I}'$ of maximum coverage problem from a YES-instance $\mathcal{I}$ of set cover trivially. Given a YES-instance of set cover $T_\mathcal{I} \subseteq R_\mathcal{I}$ and $l_\mathcal{I}$, we construct an instance $\mathcal{I}'$ of maximum coverage

$$\begin{aligned} U_{\mathcal{I}'} &= \bigcup R_\mathcal{I} \\ S_{\mathcal{I}'} &= R_\mathcal{I} \\ S'_{\mathcal{I}'} &= T_\mathcal{I} \\ k_{\mathcal{I}'} &= l_\mathcal{I}. \end{aligned}$$

It is easy to see, that $|S'_{\mathcal{I}'}| \leq k_{\mathcal{I}'}$, because $|T_\mathcal{I}| \leq l_\mathcal{I}$. Moreover $\bigcup S'_{\mathcal{I}'}$ covers the whole universe $U_{\mathcal{I}'}$, as $\bigcup T_\mathcal{I} = \bigcup R_\mathcal{I}$, therefore the coverage maximal. This reduction is indeed polynomial, hence given lemma (1), the maximum coverage problem is $\mathcal{NP} - hard$. □

3.2 Graph theory

Later in this work, we rely on graph structures that are studied in graph theory, namely shortest paths. We use *Graph theory with applications* by John A. Bondy et al. [9] as a reference book and since our graph notation is a bit different, we define it in this section.

John A. Bondy [9, sec. 1.1] defines a directed graph D as an ordered triple $D = (V(D), A(D), \psi_D)$ consisting of a non-empty set $V(D)$ of vertices, a set $A(D)$ – disjoint from $V(D)$ – of arcs and an incidence function ψ_D. The function ψ_D associates with each arc of D an ordered pair of vertices of D. We say, that arc a joins u to v if $\psi_D(a) = (u, v)$, where $a \in A(D)$ and $u, v \in V(D)$. The pictorial representation of an arc is an arrow $u \to v$ that can also be seen in figure 3.1.

In contrast to this, we define a directed graph G as an ordered pair $G = (V, E)$, where V is a non-empty set of vertices – also called nodes – and $E \subseteq V \times V$ is a set of arcs – also called edges. Each edge $e \in E$ is an ordered pair of vertices $e = (u, v)$, where $u, v \in V$. The meaning is the same as above, that is e joins u to v.

Furthermore, with the use of our notation, the book defines a *walk* W on graph $G = (V, E)$ as a sequence of alternating vertices and edges $W = v_0, e_1, v_1, e_2, \ldots e_k, v_k$, such that $e_i = (v_{i-1}, v_i) \in E$ holds for $1 \leq i \leq k$ and $v_j \in V$ for $0 \leq j \leq k$. A walk W is called a *trail* if the edges $e_1, \ldots, e_k$ of W are distinct. A trail T is called a *path* if all vertices $v_0, \ldots, v_k$ of T are distinct, and we call this path a (v_0, v_k)-path. We label a set of all paths from v_0 to v_k as $P(v_0, v_k)$.

3.2.1 Shortest paths problem

In order to formulate our shortest path problem, we need a graph $G = (V, E)$ defined in previous section 3.2. On top of this graph, the book [9] additionally defines a weight function

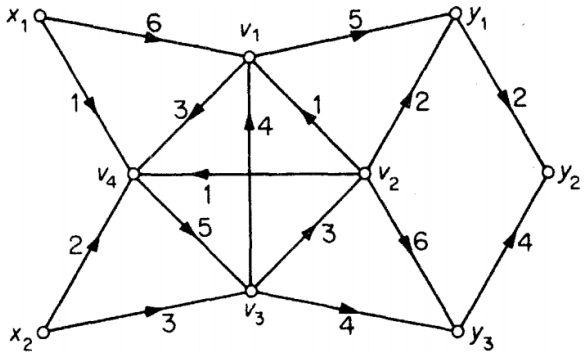


Figure 3.1: Example of a weighted directed graph

Reprinted from [9, fig. 11.1]

$w(e) \mapsto \mathbb{R}$, that associates each edge $e \in E$ with a real number. This extended graph is called a weighted graph (see figure 3.1).

One of many optimization problems defined on weighted graphs is the shortest path problem. The goal is to find a path of minimum weight connecting two vertices u and v. The cost of a path $p \in P(u, v)$ is equal to the sum of weights of its edges.

$$cost(p) = \sum_{e_i \in p} w(e_i) \tag{3.2}$$

$$SP(u, v) = \underset{p \in P(u,v)}{\operatorname{argmin}}\, cost(p) \tag{3.3}$$

As declared in [8, 1.7.9] shortest paths problem in a general directed graph – that is the graph defined above – is from a class of $\mathcal{NP}$ problems. However, shortest paths in graphs with positive weights $w(e) \mapsto \mathbb{R}_{>0}$ are solvable in polynomial time, for example by Dijkstra's algorithm as stated in [9]. In this work we use travel time as a weight function, therefore the weight function is positive, thus the problem can be solved in polynomial time and belongs in complexity class $\mathcal{P}$.

Chapter 4

Related work

This chapter is divided into two main parts. First, in section 4.1 we explore different approaches to solving resource allocation, especially vehicle allocation. Secondly, in section 4.2 we briefly summarize recent work in the patrolling games domain. As opposed to the allocation domain, where the majority benefits from the optimal distribution of resources, in the patrolling games domain, one party tries to maximize their profit at the expense of another party.

4.1 Resource allocation domain

The problem of vehicle allocation and possible reallocation has been drawing the attention of researchers for over 40 years. One of the most explored real-world application is Emergency Medical Services (EMS), systems that are making sure that the response time of emergency vehicles is minimal when faced with crucial life or death situations. In general, the challenge of EMS is to choose locations for ambulances or medical facilities, such that they are ready to handle emergency calls or substitute at places with high demand. The quality of an allocation and further reallocation is measured by an objective function which usually considers the arrival time, extent of covered population, limited available resources, local laws, strict state institution budgets or even unknowns like traffic situation. The biggest achievements and breakthroughs in this topic are nicely and categorically order up to the year 2010 in a review by Xueping Li et al. [10].

Following section 4.1.1 covers some of the influential research in the static allocation domain over the course of time. By static allocation, we mean one-time optimization of resource allocation that does not change in time. Further, in section 4.1.2 we explore research dealing with dynamic allocation and reallocation. In contrast to static allocation, the dynamic allocation and reallocation optimize resource allocation in the course of time. To provide a simpler and more understandable comparison between different approaches a common notation is presented in table 4.1.

4.1.1 Static covering models

The literature starts with the first formulation of covering model of emergency facilities proposed by Toregas et al. [11] in 1971. Given a set of demand points B, set of potential

Variable	Definition
B	set of demand points
W	set of potential facility locations
$t, t_1, t_2 \in \mathbb{R}$	maximal arrival time or distance
$W_i \subseteq W$	set of facility locations that cover demand point $i \in B$
$d_{ji} \in \mathbb{R}$	distance or travel time between locations $j \in W$ and $i \in B$
$s_i \in \mathbb{R}$	population size at demand point $i \in B$
$p \in \mathbb{N}_0^+$	number of available resources
$p_j \in \mathbb{N}_0^+$	maximum capacity of facility $j \in W$
$\alpha \in [0; 1]$	desired proportion of covered population
$x_j \in \{0, 1\}$	decision variable deciding the presence of facility at location $j \in W$
$y_i \in \{0, 1\}$	equals to 1 only if demand $i \in B$ is covered at least once
$z_j \in \mathbb{N}$	number of resources allocated to facility $j \in W$

Table 4.1: Common notation of variables

facility locations W and a maximum arrival time or distance t, the goal is to determine a minimum number of facilities (4.1a) such that all demand points are covered (4.1b) within the arrival time or distance. This formulation has many potential applications ranging from optimal positions of fire stations, hospitals, schools to private sectors such as positions of warehouses. The mathematical program written bellow is used to solve this problem.

$$min \sum_{j \in W} x_j \tag{4.1a}$$

$$\text{subject to} \sum_{j \in W_i} x_j \geq 1, \qquad i \in B \tag{4.1b}$$

$$x_j \in \{0, 1\}, \qquad j \in W \tag{4.1c}$$

The set of facility sites covering demand point i is $W_i = \{j \mid d_{ji} \leq t\}$ given that d_{ji} is a distance measured between sites $j \in W$ and $i \in B$. And x_j is a binary decision variable deciding the presence of a facility at j. However, this model does not allow to specify the number of available resources.

The result is indeed a minimum number of facilities providing full coverage, but all demand points are treated equally. This becomes a problem when considering town centres with a high concentration of requests and less dense rural areas. This complication was tackled by R. Church and Ch. ReVelle in *The maximal covering location problem* [12]. They gave each location $i \in B$ a population size s_i which serves as a weight of i, making this a weighted set cover. Also, a new binary decision variable y_i was introduced to ensure that

each point i is sufficiently covered by at least one facility (4.2b).

$$max \sum_{i \in B} s_i y_i \tag{4.2a}$$

$$\text{subject to } \sum_{j \in W_i} x_j \geq y_i, \qquad i \in B \tag{4.2b}$$

$$\sum_{j \in W} x_j = p \tag{4.2c}$$

$$x_j \in \{0, 1\}, \qquad j \in W \tag{4.2d}$$

$$y_i \in \{0, 1\}, \qquad i \in B \tag{4.2e}$$

The objective (4.2a) is to maximize the population reachable within distance t. Given the desired number p of facilities, this program can compute the largest amount of population covered together with the location of p facilities that grant this coverage.

This formulation was a great achievement as it was successfully applied to solve practical vehicle allocation problems. However, both previous formulations share a common drawback. It is the fact that the processing of an emergency request lasts some not negligible time. Once a facility is at its full capacity, meaning that all its resources are allocated to some demand points, other locations covered by this facility may no longer be covered.

One of the possible solutions to this issue was proposed by M. Gendreau et al. [13]. Two arrival times $t_1 < t_2$ are introduced and the main objective (4.3a) is to maximize the number of points covered by at least two ambulances within smaller radius t_1 while covering all demands by some ambulance within arrival time t_2. This formulation even allows assigning multiple vehicles z_j to a single location $j \in W$, such that they can respond to multiple calls at the same time. The fact that demand point $i \in B$ is covered in time t_1 by a facility j, given travel time d_{ji} between j and i, is represented by a binary variable

$$\gamma_{ji} = \begin{cases} 1 & d_{ji} \leq t_1 \\ 0 & \text{otherwise} \end{cases}$$

Very similarly the variable δ_{ij} depicts the coverage of point i by j within t_2

$$\delta_{ji} = \begin{cases} 1 & d_{ji} \leq t_2 \\ 0 & \text{otherwise} \end{cases}$$

Similarly to y_i in previous mentions the variables y'_i and y''_i are equal to one only if the demand point i is reachable within t_1 by at least one ambulance or by at least two ambulances respectively. The flexibility of this model even allows specifying the total number of available ambulances p as well as maximum fleet size per each facility p_j, which is presented in constraints (4.3f) and (4.3g). However, there is no real benefit for assigning more than 2 ambulances to a single hospital, thus $p_j = 2$ can be imposed in practice. Another user-specified variable $\alpha \in [0; 1]$ specifies the proportion of the population that must be covered

by an ambulance located within t_1 time (4.3c).

$$max \sum_{i \in B} s_i y_i'' \tag{4.3a}$$

$$\text{subject to} \sum_{j \in W} \delta_{ji} z_j \geq 1 \qquad i \in B \tag{4.3b}$$

$$\sum_{i \in B} s_i y_i' \geq \alpha \sum_{i \in B} s_i \tag{4.3c}$$

$$\sum_{j \in W} \gamma_{ji} z_j \geq y_i' + y_i'' \qquad i \in B \tag{4.3d}$$

$$y_i'' \leq y_i' \qquad i \in B \tag{4.3e}$$

$$\sum_{j \in W} z_j = p \tag{4.3f}$$

$$z_j \leq p_j \qquad j \in W \tag{4.3g}$$

$$y_i' \in \{0, 1\} \qquad i \in B \tag{4.3h}$$

$$y_i'' \in \{0, 1\} \qquad i \in B \tag{4.3i}$$

$$z_j \in \mathbb{N}_0^+ \qquad j \in W \tag{4.3j}$$

The constraint (4.3b) states that all demands must be covered by at least one vehicle within the bigger time t_2. The correct behaviour of variable y_i' and y_i'' is enforced by (4.3d) that counts the number of vehicles covering demand point i on the left-hand side; the right-hand side is then equal to 1 or 2. Additionally, by (4.3e) the demand is covered two times only if it is covered at least once. However, this model becomes infeasible when it is not possible to cover all demand points within arrival time t_2. Additionally, it does not optimize the actual arrival time.

Karl F. Doerner et al. extended this work in 2005 with a paper called *Heuristic solution of an extended double-coverage ambulance location problem for Austria* [14]. A density of the population in different demand points is considered and the sought solution limits the ratio between the number of inhabitants and the number of available ambulances for the areas with a higher time limit. These specifications of their model are based on real data from

Austria that were published by the Austrian Red Cross.

$$max \sum_{i \in B} f - M_1 f_1 - M_2 f_2 - M_3 f_3 \tag{4.4a}$$

$$\text{subject to} \quad (4.3d), (4.3e), (4.3f), (4.3g), (4.3h), (4.3i), (4.3j) \tag{4.4b}$$

$$f = (4.3a) \tag{4.4c}$$

$$f_1 = |\{v_i \in B : \sum_{j \in W} \delta_{ji} z_j = 0\}| \tag{4.4d}$$

$$f_2 = \alpha - min\Big\{\alpha, \Big(\sum_{i \in B} s_i y_i'\Big) / \Big(\sum_{i \in B} s_i\Big)\Big\} \tag{4.4e}$$

$$f_3 = \sum_{i \in B} max(0, \omega_i - \omega_0) \tag{4.4f}$$

$$w_i = \frac{s_i}{\sum_{j \in W} \delta_{ji} z_j} \tag{4.4g}$$

The constraint (4.4d) counts the number of demands not covered within the larger time t_2 which is a modification of the original (4.3b). As an alternative to (4.3c), at the line (4.4e), the difference between desired coverage proportion α and the actual coverage is computed. Since an allocation of ambulances where some have to cover a large demand within t_2, while others cover only a small demand may be unrealistic, the final penalty (4.4f) is added to the model. It penalizes when w_i – the number of inhabitants per ambulance within t_2 – exceeds the predefined limit of w_0. Finally weight factors $M_1, M_2, M_3 \in \mathbb{R}_{>0}$ balance the impact of each of the three soft constraints.

In work by O. Karasakal et al. [15], they address the problem of binary decision between covered and not covered facility, which is based on a critical distance. When the facility is within this critical distance, it is covered otherwise it is not. This assumption is not always reasonable as the level of provided service does not change in a crisp way from fully covered to not covered at all. They introduce a new notion named *partial coverage* which is defined as a function of the distance of the demand point to the facility. They formulate the problem as follows

$$max \sum_{i \in B} \sum_{j \in W_i} C_{ij} u_{ij} \tag{4.5a}$$

$$\text{subject to} \sum_{j \in W} x_j = p \tag{4.5b}$$

$$u_{ij} \leq x_j \qquad i \in B, j \in W \tag{4.5c}$$

$$\sum_{j \in W_i} u_{ij} \leq 1 \qquad i \in B \tag{4.5d}$$

$$x_j \in \{0, 1\} \qquad i \in B \tag{4.5e}$$

$$u_{ij} \in \{0, 1\} \qquad i \in B, j \in W \tag{4.5f}$$

$$\tag{4.5g}$$

The binary variable u_{ij} is equal to one only if facility j covers demand point i. Constraint (4.5b) ensures, that total number of sited facilities is equal to the number of available

facilities, constraint of type (4.5c) limits u_{ij} according to selected facilities and (4.5d) require that the demand point i is covered at most once, where only the facility providing the best coverage level is selected. The coverage level C_{ij} is defined using two thresholds, t_1 is the maximum full coverage distance and t_2 is the maximum partial coverage distance ($t_1 < t_2$), as

$$C_{ij} = \begin{cases} 1 & \text{if} \quad d_{ji} \leq t_1 \\ f(d_{ji}) & \text{if} \quad t_1 < d_{ji} \leq t_2 \\ 0 & \text{otherwise} \end{cases}$$

where $0 < f(d_{ji}) < 1$ is called the partial coverage function. They conclude that introduction of partial coverage has a substantial effect on the optimal solution. Unfortunately, this model does not guarantee a backup facility for demand points, that is used, when the closest facility reaches its maximum capacity.

4.1.2 Dynamic allocation and relocation models

While static models presented in section 4.1.1 are useful on the coarse strategic level, they lack flexibility on the finer operational level. Suppose that a high-demand area has used up all emergency vehicles within its reach. Other idle ambulances located at places with a low probability of demand should be relocated to help where needed. In this section, models are able to redeploy facilities to provide better temporal and spatial coverage.

One of the possible improvements of static allocation of EMS is provided in a work by Yisong Yue et al. [16], where a simulation-based approach is used to substitute for mathematical models. In this paper, it is claimed that abstractions of mathematical programming usually fail to capture crucial temporal features such as time-dependent travel times, congestion patterns or the fact that a patient may specify the preferred hospital. Since the simulation was already commonly used for the measurement of quality of some mathematical models, the optimization via simulation is highly motivated. Nonetheless, a potential limitation is the reliability of the simulator, which requires a considerable amount of historical data. Unfortunately, this is our case, as we possess very limited data.

The quality of particular allocation is measured by seemingly black box simulator that is built upon circa ten thousand logged emergency requests over the course of one month. Firstly, a time-dependent sequence of emergency calls is generated and is processed in first-come first-served fashion. Whenever a request arrives, the dispatch officer assigns the closest available ambulance to handle the situation. This ambulance becomes unavailable for the processing time of the request, any other idle ambulances are candidates for possible redeployment. Given the initial distribution of resources and the redeployment strategy, the simulator outputs a fitness measure such as a number of requests served in 15 minutes. Given this objective function, a simple greedy allocation algorithm is used. It iteratively selects the best position for an ambulance with highest incremental gain relative to the current solution. Similarly, a redeployment strategy chooses such relocation that the expected utility of the next time interval is maximized.

4.2 Patrolling games domain

In contrast to Emergency Medical Services application, where the victim does not usually exploit the position of emergency vehicles, the allocation of resources may be exploited in the security domain. For example, if the burglar knew positions of incident response vehicles, he could target the building with longer arrival time.

As written by Branislav Bošanský et al. in [17], these situations are often modelled using game theoretical models. One defending party needs to protect an area in order to prevent an attack on high-valued targets from the other attacking party. Contrary to our work, where we assume – in assumption (5), section 5.3.1 – that the attacker does not know positions of incident response vehicles, they allow the attacker to observe defender's current position. This knowledge is then exploited when planning the attack. Additionally to previous work in the field of patrolling games, they allow the targets to move, however, they also formulate game with stationary targets. A typical real-world application is in the maritime domain, where moving vessels targeted by pirates need protection. The sought solution is found using non-linear mathematical programs.

Chapter 5

Technical approach

5.1 Informal problem definition

The main objective of this work is to improve the area coverage and response time of incident response vehicles in the building security domain – which was briefly introduced in chapter 2 – by modifying their idle locations. However, in such a complex domain where one subsystem depends on the reliability and response time of the previous one in the chain, one could argue that there are multiple ways to improve the response time. We will define the variables we are allowed to modify and other parts of the system that are treated as black boxes and assumed to work as expected without any problems.

The chain of events starts at the customer's building that is being monitored. A network of detectors, connected by a control module scans the property and triggers an alarm when a suspicious action happens. A reader can get a better insight into this process in section 2.1, where we briefly explain limitations of this subsystem. For instance, location and configuration of individual detector indoors or outdoors of a building matter. It might happen that an alarm triggers too late when it is no longer possible to catch the burglar. In this work, however, we are not interested in the proper configuration or placement of these detectors.

Another big concern is an alarm triggered by accident – called false positive. This usually happens when a person walks into a monitored area without turning off the alarm first. Deployment of incident response vehicle is unnecessary in such cases, yet it is hard for an alarm receiving centre worker to distinguish between true and false alarm. As described in section 2.2 he first informs the customer and asks whether the cause of the alarm is known. In case of any uncertainty, the vehicle is deployed. In this work, we are not interested in this problem and we treat each alarm as a proper one caused by a burglar or other deployment worthy event.

At this point, we have a black box that tells us about a burglary in progress at a certain location. All such possible locations are known in advance as a customer needs to register each property in order to subscribe to this type of service. Additionally, we need all possible locations, where an idle vehicle can be placed. We also require geographic data containing road network of the area we are interested in. This network encodes information about possible movements of vehicles and local motor vehicle laws, such as one-way streets, forbidden turns and speed limits. Vehicles cannot take a road that is not present in this network.

Variable	Definition
$l \in L$	locatable point $l = (\varphi(l), \lambda(l))$
$\varphi(x) \in [-\frac{\pi}{2}, \frac{\pi}{2}]$	latitude of x
$\lambda(x) \in [-\pi, \pi]$	longitude of x
$v \in V \subseteq L$	road network node, V is a finite set of all such nodes
$e = (i, j) \in E$	road network directed edge from i to j, where $i, j \in V$
	E is a finite set of all such edges
$speed(e) \in \mathbb{R}_{>0}$	average travel speed on edge $e \in E$
$distance(e) \in \mathbb{R}_{>0}$	the road distance of edge $e \in E$
$b \in B \subseteq L$	set of monitored buildings
$time(b) \in \mathbb{R}_{>0}$	maximum allowed arrival time to monitored building $b \in B$
$w \in W \subseteq L$	set of potential idle vehicle locations
$\hat{v_x} \in V$	representative of a location x snapped to road network V
$W_b \subseteq W$	set of idle vehicle locations that cover building $b \in B$ within $time(b)$
$B_w \subseteq B$	set of buildings that are covered by idle location $w \in W$
$SP(x, y)$	shortest path between nodes $x \in V$ and $y \in V$
$a \in A = \mathbb{R}_{\geq t_0} \times B$	alarm event $a = (a_t, a_b)$ at time a_t, building a_b and t_0 as the start of the simulation

Table 5.1: Description of mathematical notation

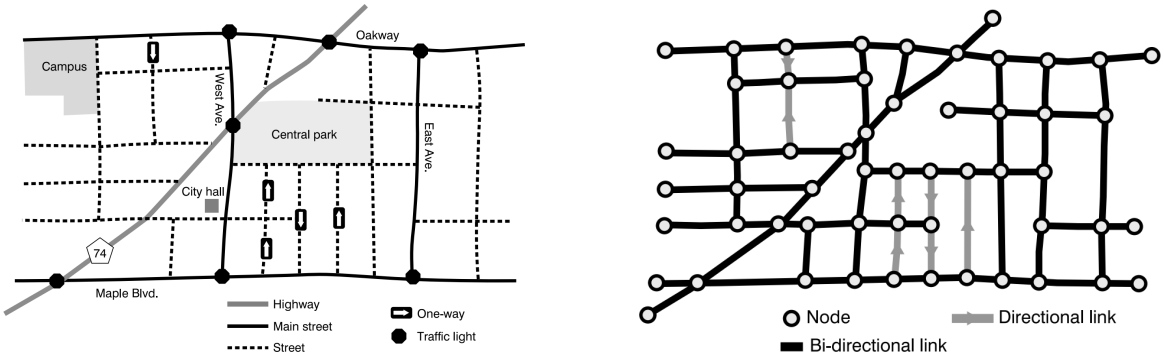
Furthermore, we expect that each monitored location is accessible by a road that is present in this network, otherwise, we assume that the vehicle crew can park on the nearest road and continue on foot.

To summarize, the objective of this work is to manage a fleet of incident response vehicles, given the number of those vehicles, road network, locations of monitored buildings and all possible idle locations of vehicles. Firstly, we explore a static allocation strategy, that selects idle locations of incident response vehicles only once. When an alarm is detected, the closest vehicle is deployed and returns to the original spot after the inspection. Secondly, a dynamic allocation strategy is presented. Initially, it works the same as static allocation, but when an alarm is detected, not only that the closest vehicle is deployed, but other idle vehicles move as well to optimize the coverage and substitute for currently intervening ones.

5.2 Formal problem definition

In this section, we decompose the problem of incident response vehicle placement into smaller problems. These are individually defined and a solution to each one is presented. Throughout following sections, we incrementally add to our mathematical notation that is presented in table 5.1.

Firstly we define a graph on top of a road network in section 5.2.1, then we tackle a real-world problem that is the consequence of this graph representation in 5.2.2. The navigation of incident response vehicles between two points is done via shortest paths in 5.2.3. Finally, in section 5.2.4 we formalize the allocation of incident response vehicles as maximum coverage problem, that was defined in section 3.1.



(a) Cartography of a network data model. Reprinted and adapted from [18, fig. 10.16]

(b) Topology of a network data model Reprinted and adapted from [18, fig. 10.15]

Figure 5.1: Difference between network data cartography 5.1a and topology 5.1b

Purpose of cartography is a visualization of a transport network that allows for simple navigation. Different elements have specific attributes, for example, a highway, main street or one-way. It can also include other features such as landmarks to provide better orientation.

Topology is an arrangement of nodes and links in the network. Nodes represent locations and links represent direction and connectivity. This topology abstraction should be as close as possible to the real world structure it represents.

5.2.1 Road network as a graph

Graph theory, discussed in section 3.2, is an extensively studied branch of mathematics with well-defined problems, their solutions and mathematical guarantees. Fortunately, it is possible to define a graph on top of a road network, which will allow us to apply well-known algorithms for solving shortest path problem. Such algorithm will provide a path from the current location of the incident response vehicle to the location that needs inspection with fastest arrival time guarantee. However, this does not hold in the real-world, where unpredictable traffic varies the duration of travel or changes the optimal route completely. In our case we assume ideal travel conditions.

Assumption 1 (Ideal travel condition)**.** *All speeds that were considered during planning are met on all possible paths during plan execution. In other words the transportation speed along all edges remains the same during the actual transport as during planning.*

The book by Jean-Paul Rodrigue, *The Geography of Transport Systems* [18], describes the process of transforming a real road network into its graph representation. A simple illustration of this process is depicted in figure 5.1. The number one rule used in this book is that every street dead end and an intersection becomes a node. If the node was connected to some other node in the real network, it remains connected by an edge in the graph representation.

For our purposes, a node v is an abstraction of a location such as an intersection or street dead end and has GPS coordinates associated with it – latitude $\varphi(v) \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and longitude $\lambda(v) \in [-\pi, \pi]$. A finite set of all such nodes is V. An edge $e = (i, j)$ is a directed link between nodes $i, j \in V$ and a finite set of all such edges is E. Edges represent the

direction of one way streets and hold other road properties such as $speed(e) \in \mathbb{R}_{>0}$ which is the average travel speed on this edge. Additionally, $distance(e) \in \mathbb{R}_{>0}$ is the road distance from node i to node j, using the edge $e = (i, j)$. On top of all nodes V and all edges E we define a directed graph $G = (V, E)$.

Other rules can also be applied such as an arbitrary node that is not a dead end or an intersection can be added to the graph if within that segment an attribute of the edge is changing. For instance, a speed limit or a number of lanes. Indeed, some attributes does not need to change at an intersection and an urban district speed limit is a good example. Additional 'dummy node' can be also added for aesthetic purposes so that the graph representation remains comparable to the real network.

Using this method, we are able to encode road graph information and traffic rules in a directed graph $G = (V, E)$. For example, if a road from i to j is bi-directional both edges $e_1 = (i, j)$ and $e_2 = (j, i)$ are in E. Otherwise, in a case of a one-way route, one of e_1 or e_2 is missing and it's the one corresponding to the forbidden direction. Another example is a roundabout with 3 exits, those are represented by nodes v_1, v_2, v_3 in clockwise direction. In the Czech Republic vehicles drive on the right side of the road and traffic on roundabouts travels in a counter-clockwise direction. The graph representation of such situation would include only edges $(v_1, v_3), (v_3, v_2)$ and (v_2, v_1) between the roundabout nodes. It is not possible to travel in the wrong direction and we cannot visit v_2 from v_1 without visiting v_3 first.

5.2.2 Points to road snapping

One of the reasons we defined the road network as a graph in section 5.2.1 was that we can use well-known shortest path algorithms to find optimal route between two nodes. Such algorithms often require a starting node $v \in V$ and a set of goal nodes $G \subseteq V$. Unfortunately this is not always met in our case, as we would like to find the shortest path between each possible vehicle location $w \in W$ and monitored building $b \in B$, but our definition of a road graph $G = (V, E)$ does not specify anything about the location of monitored buildings B and idle vehicle locations W. In general, they are not incident with graph edges $B \not\subseteq V$, $W \not\subseteq V$. This means that we are not able to find the shortest path to a monitored building using general shortest path algorithm.

Common approaches to this problem are to use the closest node $v \in V$ instead. However, there may be certain obstacles – fence, escarpment, etc. – that are not known, making this method inapplicable in some sporadic cases. In our case, we make following assumption.

Assumption 2 (Location reachability)**.** *We assume that each potential idle vehicle location $w \in W$ is accessible by a road that is connected with our road network G with negligible travel time. In case of monitored buildings $b \in B$, we assume that they are connected by a road as in the previous case, or that the crew of incident response vehicle can park on the nearest road and continue on foot, again in negligible travel time.*

The process – known as snapping and visualized in figure 5.2 – takes as an input a graph $G = (V, E)$ and a point $w \notin V$, where both w and $v \in V$ have associated GPS coordinates – latitudes $\varphi(v), \varphi(w)$ and longitudes $\lambda(v), \lambda(w)$. Using these coordinates, we can define a distance function d and find the node $\hat{v_w} \in V$ to w that lies on the graph and minimizes d.

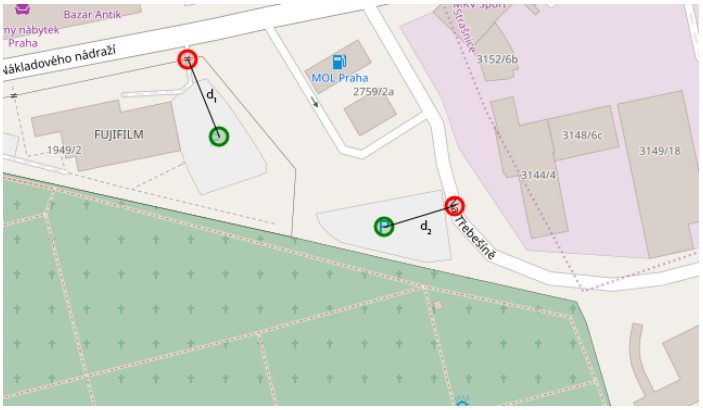


Figure 5.2: Snapping to road network

Snapping is a process where a point that is not incident with the road network (green circle) is represented by the nearest point from the road network (red circle). The distance (d_1, d_2) between those points is called snapping error. Visualization made by Leaflet [19].

$$\hat{v_w} = \underset{x \in V}{\operatorname{argmin}}\, d(w, x) \tag{5.1}$$

A standard method of calculating distances over the surface of the Earth is the Haversine formula (5.2), where the shape of the Earth is approximated by a sphere of radius r.

$$haversine(x, y) = 2r \arcsin \sqrt{\sin^2\left(\frac{\varphi(y) - \varphi(x)}{2}\right) + \cos(\varphi(x))\cos(\varphi(y))sin^2\left(\frac{\lambda(y) - \lambda(x)}{2}\right)} \tag{5.2}$$

Nonetheless, under certain conditions, we can assume that the surface of the Earth is flat and use the computationally more efficient method – the Pythagoras theorem (formula (5.3)).

Assumption 3 (Approximation by plane)**.** *When dealing with small distances between two points not located near geographic poles, we can assume that the surface of the Earth is flat, thus it can be approximated with a plane.*

In our case we expect measured distances to be small because the distance from a building to the nearest road is generally in meters. Since experimental data used in this work are located in the Czech Republic, we do not expect points around geographic poles, which make this approximation method more inaccurate. Furthermore, since square root is strictly increasing function it can be omitted from formula (5.3) when used in formula (5.1) as it does not change the argmin.

$$planar(x, y) = r\sqrt{(\varphi(y) - \varphi(x))^2 + \left[cos\left(\frac{\varphi(x) + \varphi(y)}{2}\right)(\lambda(y) - \lambda(x))\right]^2} \tag{5.3}$$

However, such process can be computationally demanding and requires distance computation between all (w, x) pairs for one query point w and graph nodes $x \in V$. The complexity of naive approach linearly depends on the number of nodes $|V|$ in the road graph. This query for the nearest point can be optimized using spatial indexing methods. They usually come

with initial build computational cost, but improve the average time of the query, making them useful for multiple queries. Since spatial indexing is not the focus of this work, we will point an interested reader to a book *Computational Geometry: Algorithms and Applications* [20] where a tree data structure called quadtree is discussed.

Ultimately, an even better approach is to snap to the closest edge, where a new 'dummy' node can be created, such that the distance to the original location is minimized, and appropriately connected to the rest of the graph.

5.2.3 Shortest paths

As preliminaries to the shortest paths computation, we defined a road network as a graph in section 5.2.1 and we snapped points of our interest to this graph in section 5.2.2. In this section, we will be finding the shortest path between any nodes on the graph. The term shortest path may be a bit misleading in vehicle routing since we are actually interested in the fastest path, that is to minimize the travel time, not the travel distance. This is due to the fact that the shortest path problem is a well-established term in graph theory, where edge has a cost associated. In our case, it is the time needed to traverse that edge.

In this work, we are interested in the shortest path between all pairs of potential idle vehicle locations $w \in W$ and monitored buildings $b \in B$, or rather their snapped representatives $SP(\hat{v_w}, \hat{v_b})$, where $\hat{v_w}, \hat{v_b} \in V$. Moreover, during dynamic reallocation, the vehicles may be on their way to their new idle location when another alarm is detected. In that case, we need to compute the shortest path between any snapped point $\hat{v_x}$ to snapped monitored building $\hat{v_b}$ (to determine the closest vehicle), or to snapped potential idle location $\hat{v_w}$ (to compute reallocation cost to other idle locations).

Since both $speed(e)$ and $distance(e)$ are positive in our graph, the resulting $time(e)$ will be also positive. Therefore, if we use $time(e)$ as the cost function of the edge $e \in E$ the graph would not contain any negative or zero cycles, which is beneficial for shortest path algorithms.

$$cost(e) = time(e) = \frac{distance(e)}{speed(e)} \tag{5.4}$$

Another observation is that each building b has maximum arrival time $time(b) \in \mathbb{R}_{>0}$. Any shortest path $SP(x, b)$ that takes longer than $time(b)$ is not interesting and can be omitted. This essentially prunes our search tree only to nodes, that are reachable in less than $time(b)$.

Given the facts above, we need a single-source shortest path algorithm to find accessible idle locations from any node $x \in V$, including snapped points, on the graph $SP(x, \hat{v_w})$. We also need a cost bounded single-destination shortest path algorithm to find all possible idle locations, such that the monitored building b is reachable within the arrival time $time(b)$ – we need to find all $SP(\hat{v_w}, \hat{v_b})$, such that $cost(SP(\hat{v_w}, \hat{v_b})) \leq time(b)$. Fortunately, the single-destination problem can be reduced to the single-source one just by reversing the orientation of all edges in the graph.

The chosen algorithm for this task is a modification of uniform cost search that is presented in [21]. Our modification – algorithm (1) – adds bounds to the cost of the explored paths and reverses the edges of the graph in the beginning. As an input, it takes

a road graph $G = (V, E)$, source node $\hat{v_b} \in V$, arrival limit $time(b)$ and *true* flag for *singleDestination*. New graph $G' = (V, E')$ with reversed edges is constructed, such that $E' = \{e' = (j, i) \mid e = (i, j) \in E, i, j \in V\}$.

Algorithm 1: Time bounded uniform cost search algorithm

```
1  function uniformCostSearch (graph, source, timeThreshold, singleDestination);
   Input  : G = (V, E), source node ∈ V, maximum arrival time ∈ ℝ>0, true if
            single-destination
   Output: Map where keys are all explored nodes within timeThreshold and values
            contain parent node and time from source.
2  if singleDestination then
3  |  graph.reverseEdges();
4  end
5  Q ← emptyPriorityQueue(); ▷ contains objects (node, parent of node, arrival time)
6  Q.add((source, null, 0));
7  explored ← emptyMap();      ▷ node → (parent of node, time from source to node)
8  while Q.isNotEmpty() do
9  |  node, parent, time ← Q.popMin();                  ▷ minimum by arrival time
10 |  if explored.contains(node) then
11 |  |  continue;
12 |  end
13 |  explored.put(node, (parent, time));
14 |  foreach child in graph.childrenOf(node) do
15 |  |  if notValidTransition(node, child) then
16 |  |  |  continue;                                   ▷ checks for U-turns, etc.
17 |  |  end
18 |  |  childTime ← time + graph.getTime(node, child);
19 |  |  if childTime ≤ timeThreshold and explored.notContains(child) then
20 |  |  |  Q.add(child, node, childTime);
21 |  |  end
22 |  end
23 end
24 return explored
```

This search finds all nodes that are reachable from $\hat{v_b}$ in the given time limit $time(b)$. It returns a mapping $f(x) \to \{(p_x, t_x) \mid t_x \in \mathbb{R}_{>0}, p_x \in X\}$ defined on all explored nodes $x \in X \subseteq V \cup \{null\}$, where t_x is the arrival time to x from the source node $\hat{v_b}$ and p_x is the immediate predecessor of x on the shortest path $SP(\hat{v_b}, x)$. Note that $f(\hat{v_b}) = (null, 0)$ every time, where *null* is a special identifier stating that the start of the path does not have any immediate predecessor.

However, since the edges were reversed in the beginning it returns all nodes x from whose the destination node v_b is reachable within the time limit $time(b)$. The mapping $f(x)$ must be also treated differently since the edges were reversed. It is easy to see that without the reversal of graph edges, the algorithm can be used for single-source shortest path queries.

5.2.3.1 Proof of completeness

This section proves that the algorithm terminates for any input. This proof for general uniform cost search is available in [21], however, our proof is a bit different since we modified the algorithm. As the cost of an edge e is defined as traversal time $time(e)$ (equation 5.4), it is always positive, thus the cost of a child (line 18) is always greater than the cost of its parent (line 9). This means that we either hit the time threshold and stop adding new nodes to the queue (lines 19 - 21) or we explore all nodes in the finite graph, which would also stop adding new nodes to the queue (lines 19 - 21). Finally, in each iteration of the main loop (line 8) we remove one node from the queue, therefore the queue will be empty at some point.

5.2.3.2 Proof of correctness

The proof that this algorithm indeed returns shortest paths is also similar to the one in [21]. It states that once the search selects a node n for expansion for the first time (lines 9 - 12), the optimal path to that node has been found. If that was not the case, shorter path to n must exists. Let us say that S is a set of all explored nodes (line 7) that contains source node $s \in S$ and some arbitrary node $a \in S$ and the shortest path $SP(s, a)$ is known. Once a node is explored, all of its valid successors are added to the frontier queue (lines 14 - 22).

Let the contents of a frontier queue Q be a triplet (node, its parent, its cost).

$$Q = \{(n, s, cost((s, n))), (n, a, cost(SP(s, a)) + cost((a, n)))\}$$

The algorithm then correctly selects the tuple with the shorter arrival time first (line 9), because it extracts the tuple with the smallest cost from Q. Further expansions of node n are ignored (lines 10 - 12).

Now let us suppose that the frontier queue Q contents are

$$Q = \{(n, s, cost((s, n))), (n', s, cost((s, n')))\}$$

and the algorithm incorrectly expands node n because the shortest path to n is via nodes s, n', n.

$$cost((s, n)) > cost((s, n')) + cost(SP(n', n)) \tag{5.5}$$

Since the cost of all edges in the graph is positive, $cost(SP(n', n))$ must be positive as well, and we can transition from (5.5) to (5.6).

$$cost((s, n)) > cost((s, n')) \tag{5.6}$$

But now $cost((s, n)) > cost((s, n'))$, thus the algorithm will not expand node n, but node n' instead. This leads to contradiction and completes the proof.

Finally, no path will take longer than specified time threshold $t \in \mathbb{R}_{>0}$. The first element added to the queue (at line 6) has cost 0, and $0 < t$. Any additional elements are added to the queue only if their cost is less or equal to threshold t (lines 19 - 21). Only nodes from the queue appear in the final result, thus if they are not added to the queue, they are not in the final result.

5.2.4 Maximum coverage problem

Now that we can compute arrival times between possible idle vehicle locations $w \in W$ and monitored buildings $b \in B$ using the procedure defined in previous section 5.2.3, it is possible to formulate the problem of maximizing the coverage of buildings protected as a maximum coverage problem.

One run of uniform cost search algorithm (1) for monitored building b and its arrival time limit $time(b)$ yields a result from which a set $W_b \subseteq W$ that consists of all possible idle vehicle locations that are within arrival time $time(b)$ is extracted.

$$W_b = \{w \mid w \in W, cost(SP(w, b)) \leq time(b)\}, \text{ where } b \in B \tag{5.7}$$

After obtaining W_b for each building $b \in B$, we can construct sets $B_w \subseteq B$ that contains all buildings b that are covered from idle location w.

$$B_w = \{b \mid b \in B, cost(SP(w, b)) \leq time(b)\}, \text{ where } w \in W \tag{5.8}$$

The maximum coverage problem, as defined in section 3.1, requires a universe U, collections of sets S and integer k. In our case, we are covering monitored buildings B using coverages B_w with a limited amount of vehicles k.

$$U = B \tag{5.9a}$$
$$S_w = B_w, \text{ where } w \in W \tag{5.9b}$$
$$S = \{S_w \mid w \in W\} \tag{5.9c}$$
$$k = \text{number of available vehicles} \tag{5.9d}$$

The objective of the maximum coverage is to find a subset $S' \subseteq S$, such that $|S'| \leq k$ and $|\bigcup_{S_w \in S'} S_w|$ is maximized. In our domain, this means to find locations of idle vehicles w, where $S_w \in S'$, such that we use at most k vehicles and maximize the number of covered buildings $|\bigcup_{S_w \in S'} S_w|$.

This formulation alone would maximize the coverage of protected buildings, as the building b is considered covered if at least one incident vehicle is located within arrival time $time(b)$. However, it comes with a drawback, such as we are not able to control the actual arrival time, we only know if the building b is covered within $time(b)$. Additionally, it is sufficient to cover a building only once, leaving some available vehicles unused. Once a vehicle is deployed, it is likely to leave a significant amount of buildings uncovered, which could be prevented by using more available vehicles. These problems are tackled in upcoming sections.

5.3 Main components

In order to evaluate the quality of incident response vehicle allocation and monitored building coverage, we need to replicate the process consisting of an alarm detection 2.1, vehicle deployment 2.2 and vehicle intervention 2.3. We designed a system consisting of 4 independent components, each one dealing with its own task. Top level architecture can be seen in figure 5.3.

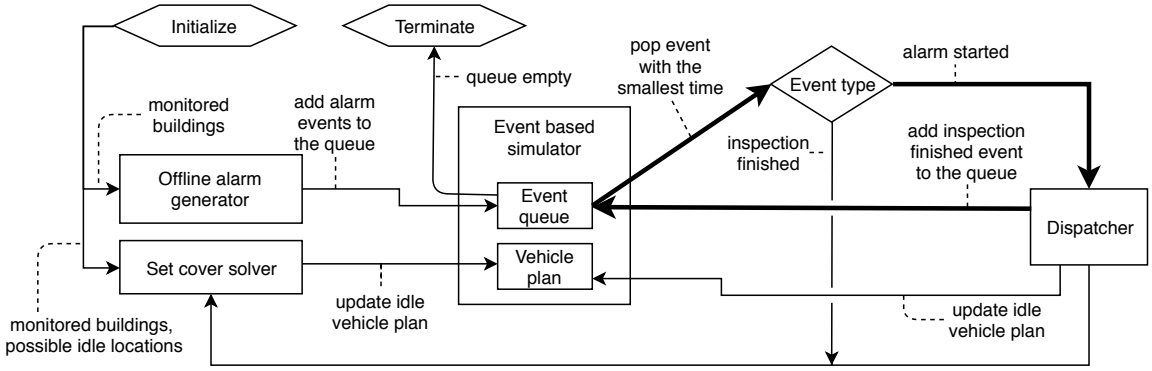


Figure 5.3: Diagram of our system structure

Our system consists of 4 independent components. *Offline alarm generator* creates a sequence of alarms at monitored buildings that are triggered over a period of time. *Set cover solver* optimizes the next position of idle vehicles given monitored buildings and the current position of idle vehicles. *Event-based simulator* computes the actual positions of all vehicles at given time. *Dispatcher* assigns an idle vehicle to alarm location. The main system loop is highlighted with bold arrows.

First of all, the *offline alarm generator* generates a sequence of alarms and the *set cover solver* computes the starting locations of idle incident response vehicles. Those two inputs are fed into the *event-based simulator* that simulates the 'world' and computes the location of all vehicles in given time. When an alarm occurs, the *simulator* delegates the choice of incident response vehicle to the *dispatcher*, which then asks the *set cover solver* to reallocate remaining idle vehicles. When a vehicle finishes with the building inspection, the *simulator* tells the *set cover solver* that there is a new vehicle available and re-computation of idle vehicles allocation is needed.

This decomposition of the problem into smaller parts allows for easier understanding, programming and overall maintenance of the whole system. Following sections 5.3.1 - 5.3.4 describe each component in detail.

5.3.1 Alarm generator

Purpose of an alarm generator is to simulate alarms at monitored buildings B. As described in section 2.1, there are two types of alarms – true positive and false positive. One of the tasks of alarm receiving centre employee is to distinguish these two and prevent unnecessary deployment of a vehicle to false positive alarms when possible. In order to eliminate this variable from our system, we assume that all alarms are true positive.

Assumption 4 (Alarm detection reliability)**.** *We assume that each triggered alarm is true positive and requires an intervention of an incident response vehicle.*

We further assume alarm independence.

Assumption 5 (Alarm independence)**.** *Sampling an alarm event depends only on external factors, such as location or time, and is independent of other alarms along with allocation,*

dispatch strategy and operation of incident response vehicles.[1]

This independence between alarms and operation of vehicles allows comparing multiple approaches to solving maximum building coverage on the same sequence of alarms. Moreover, the sequence of alarms can be pre-sampled for a given time span and if required, this time span can be incrementally prolonged.

Alarms are defined as

$$A = \mathbb{R}_{\geq t_0} \times B = \{(time(a), building(a)) \mid time(a) \in \mathbb{R}_{\geq t_0}, building(a) \in B\}, \tag{5.10}$$

where $time(a)$ stands for a time of the alarm since the beginning of the simulation t_0 and $building(a)$ stands for the monitored building b, where the alarm occurs. Finally, the output of alarm generator is an ordered sequence of alarms $a_0, a_1, \ldots, a_n$, where $a_i \in A$,

$$t_0 \leq time(a_0) \quad \text{and} \quad time(a_i) \leq time(a_{i+1}) \quad \text{for} \quad 0 \leq i < n.$$

.

5.3.2 Simulator

Since the state of our system depends on time, we need a component – the simulator – that determines this state given time. This simulation happens in an idealized world that is free of real-world complications.

Assumption 6 (Ideal simulation)**.** *During our simulation, we assume ideal world conditions. This means that we do not model situations such as our incident response vehicle having a car accident, malfunctioning or running out of fuel. Additionally, we do not take into consideration needs of the vehicle crew, such as working hours, and assume that the vehicle is available at all times.*

There are two types of events that are particularly interesting. Firstly, when an alarm is triggered, a vehicle should be deployed immediately and the rest of idle vehicles can be reallocated for better coverage. Secondly, when a vehicle finishes the inspection of a building, it should be moved to its new idle location. Other time instants are not of our interest, hence we define our simulator as event-driven. Because events cannot affect the past, it is sufficient to store them in a data structure that specializes in extract minimum operation – such as min-heap. This process is visualized in figure 5.4.

In each iteration, an action from this heap will be extracted, such that this action is the nearest one in the future. This action will either be an alarm detection or completed inspection. In both cases, the position of all vehicles is determined using process later described in section 5.3.2.1. Afterwards, in case of alarm detection, the alarm handler (see section 5.3.3) is executed in order to dispatch a vehicle. This process will also provide the route of the dispatched vehicle to the inspected building. An entry that represents the completion of this inspection is added to the heap. Finally, in both cases, the set cover solver – section 5.3.4 – is run and paths to new idle vehicle locations are obtained.

[1] Please note that this assumption is not suitable for situations when the burglar monitors the operation of vehicles. In such cases, a more game theoretic approach should be taken into consideration. We offer a starting point on this topic in section 4.2.

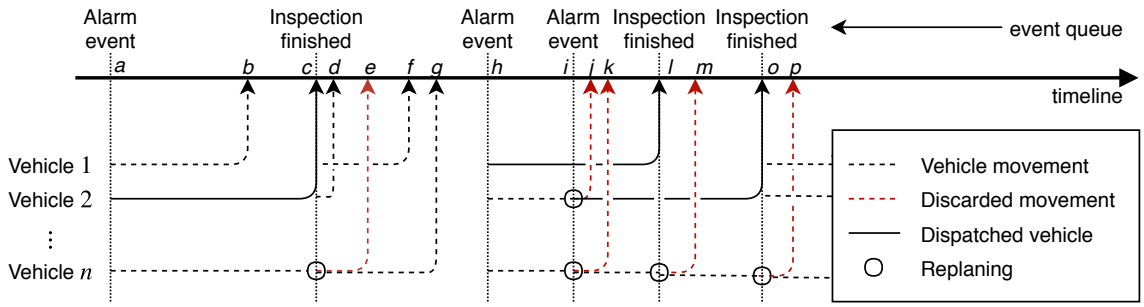


Figure 5.4: Simulator timeline

Our event-based simulator stores time ordered events and routing of idle vehicles (black dotted arrows). These idle vehicles are reallocating to a different location and arrive when the arrow meets the timeline. When an important event occurs, plans of idle vehicles may be disrupted (red dotted arrows) and replaned (black circles). If the event is an alarm event, one of the idle vehicles is dispatched (black solid arrow).

Interpretation: At time a, an alarm is triggered and vehicle 2 is dispatched to handle it. Simultaneously vehicle 1 and n start reallocating to better idle positions. At b, vehicle 1 arrives at its new idle location. At c, vehicle 2 finishes the inspection of the alarm and becomes idle again. All vehicles start reallocating to their new location, to which they arrive at times d, f and g. Note that vehicle n was still moving. This route, that would end at time e is cancelled. At time h another alarm is detected and the nearest vehicle 1 is dispatched, while other vehicles 2 and n start reallocating. At i, the idle vehicles are 2 and n, we dispatch vehicle 2 (its current route ending in j is cancelled). Again route of vehicle n ending at k is cancelled because it is reallocated to a different location. At time l vehicle 1 finished the inspection. It does not have to reallocate, because it is already in the best position, however vehicle n reallocates again and the route ending in m is cancelled. Vehicle 2 is handling an event, therefore it is not idle and does not reallocate. Finally, at o vehicle 2 finishes inspection and all 3 idle vehicles start reallocating.

5.3.2.1 Determining position of vehicle at time t

During simulation, one of the most requested operations is to determine a position of a vehicle in time $t \in \mathbb{R}_{\geq t_n}$, where t_n is the current time of the simulation. In other words, we do not need to determine the position of the vehicle in the past, but only now or in the future. Since after every important event the action of each vehicle is computed and the ideal travel conditions assumption (1), it is possible to answer such request.

The only thing we need to store for each vehicle is the time when a new path was scheduled t_v, and the actual path P. Let us extract the sequence of vertices $v_0, \ldots, v_p$ from P and set $time(v_0) = 0$. Now we can incrementally compute arrival time to each node using the following formula

$$time(v_i) = time(v_{i-1}) + \frac{distance((v_{i-1}, v_i))}{speed((v_{i-1}, v_i))} \tag{5.11}$$

As a result of both $distance(e)$ and $speed(e)$ being positive, the sequence

$$time(v_0), \ldots, time(v_p)$$

is in increasing order, such that $time(v_i) \leq time(v_{i+1})$ for $0 \leq i < p$.

This ordered property can be utilized by the binary search algorithm. In general, the binary search finds exact matches in sorted sequence in logarithmic time but can be easily

modified to return approximate matches. In our case, we would like to find the predecessor – next smaller value. Given a query time $t = t_v + t_q$, our binary search locates v_j such that

$$time(v_j) \leq t_q \leq time(v_{j+1}).$$

When $t_q \geq time(v_p)$, there is no such node as v_j, instead v_p is returned as the position of the vehicle at time t and the following step is skipped.

Now that we have nodes v_j and v_{j+1}, we can use linear interpolation under the approximation by plane assumption (3). The position of vehicle $x = (\varphi(x), \lambda(x))$ at time t is then computed as

$$y = \frac{t_q - time(v_j)}{time(v_{j+1}) - time(v_j)} \tag{5.12a}$$

$$\varphi(x) = y[\varphi(v_{j+1}) - \varphi(v_j)] + \varphi(v_j) \tag{5.12b}$$

$$\lambda(x) = y[\lambda(v_{j+1}) - \lambda(v_j)] + \lambda(v_j) \tag{5.12c}$$

When there is no path for the vehicle (e.g. after initialization, or when standing still), we can assume that there is a path of length 1 starting and ending at the location of the vehicle.

5.3.2.2 Corner cases

It may happen, that there is an alarm that requires attention, but there is no idle incident response vehicle. In that case, it is investigated by the next idle vehicle.

Due to the nature of an alarm sequence generation (see section 5.3.1), there is a possibility that multiple alarms at the same building are triggered before a vehicle arrives. It is common sense that this situation can be covered by a single vehicle and does not require additional deployment.

5.3.3 Dispatcher

The dispatcher is an abstraction of an alarm receiving centre – described in section 2.2. Similarly to its real-world counterpart, it reacts to alarms that are triggered in monitored buildings. Due to our alarm reliability assumption (4) it does not have to distinguish between false positive and true positive alarms, because the alarm generator – section 5.3.1 – only generates valid alarms.

Given positions of $n_i \geq 1$ idle vehicles $L_i = \{l_{ij} \mid l_{ij} \in L, j \in 1 \dots n_i\}$ at time t_i and the building $b_i \in B$ that requires attention, the goal of a dispatcher is to deploy a single vehicle $d_i \in L_i$, that inspects the building b_i. The choice of d_i is called deployment strategy and may be different at each iteration of the system cycle. However, we use greedy strategy (5.13), that always selects the vehicle with fastest arrival time.

$$d_i = \underset{j \in 1 \dots n_i}{\operatorname{argmin}} \, cost(SP(\hat{v_{l_{ij}}}, \hat{v_{b_i}})) \tag{5.13}$$

It is worth noting, that positions of idle vehicles are in general not incident with possible locations of incident response vehicles $L_i \not\subset W$, because, in dynamic allocation strategy, a vehicle can be on its way from previous idle location to the next one. This does not happen during static allocation, however vehicle that just finished an intervention may be returning to its original idle location when another alarm occurs.

The computation of greedy deployment strategy (5.13) in general requires snapping of all vehicle positions L_i onto road network and running algorithm (1) once. Even better is to run a slight modification of uniform cost search, that terminates after the first one of $\hat{v_{l_{ij}}}$ is popped from the queue (line 9 of algorithm (1)).

5.3.4 Set cover solver

Finally, the set cover solver optimizes positions of idle vehicles. By idle vehicle, we mean a vehicle that is not inspecting an alarm. Once this inspection is done, the vehicle becomes idle again. In general idle vehicles are parked at selected location $w \in W$ or they are driving towards one.

We formulate this optimization task as an integer linear program. In this program, we combine a maximum coverage problem, as defined in section 5.2.4, with the constraint from [13] that tries to cover a building at least twice, which helps in situations, where multiple alarms are triggered. Additionally, we also minimize arrival time.

Two scenarios are presented. Static allocation, in section 5.3.4.1, computes idle positions of vehicles only once and vehicles always return to this position after finished alarm inspection. On the other hand, dynamic allocation – section 5.3.4.2 – optimizes positions of idle vehicles during the simulation such that they substitute for currently intervening vehicles.

5.3.4.1 Static allocation

Given a set of monitored buildings B, a set of potential idle vehicle locations W, a set of idle vehicle locations W_b that cover building b within its arrival time $time(b)$ for each building $b \in B$, and total number of vehicles p, the objective of static allocation is to assign an idle location w to each vehicle. We solve this as an integer linear program, that maximizes

the coverage while minimizing the arrival time.

$$max\ M_1 \sum_{b \in B} y'_b + M_2 \sum_{b \in B} y''_b - M_3 \sum_{w \in W} x_w \bar{p_1}(w) \tag{5.14a}$$

$$\text{subject to} \sum_{w \in W} x_w = p \tag{5.14b}$$

$$\sum_{w \in W_b} x_w \geq y'_b + y''_b \qquad b \in B \tag{5.14c}$$

$$y''_b \leq y'_b \qquad b \in B \tag{5.14d}$$

$$y'_b \in \{0, 1\} \qquad b \in B \tag{5.14e}$$

$$y''_b \in \{0, 1\} \qquad b \in B \tag{5.14f}$$

$$x_w \in \{0, 1\} \qquad w \in W \tag{5.14g}$$

In order to minimize the arrival time, we define a cost function $p_1(w)$ that associates a potential idle location $w \in W$ with a penalty. It is equal to a sum of squared arrival times to covered buildings $b_w \in B_w$.

$$p_1(w) = \sum_{b_w \in B_w} cost(SP(\hat{v_w}, \hat{v_{b_w}}))^2 \tag{5.15}$$

However, the range of this cost function depends on the arrival times, that may vary in different instances of this problem. Therefore, we normalize the outcome of this function to $[0, 1]$ interval and present it as function $\bar{p_1}(w)$.

$$\alpha = \max_{w \in W} p_1(w)$$
$$\bar{p_1}(w) = \frac{p_1(w)}{\alpha} \tag{5.16}$$

The integer linear program has $2|B| + |W|$ binary variables and $2|B| + 1$ constraints. Following the maximum coverage problem, defined in section 3.1, variable y'_b (5.14e) is equal to one only if building $b \in B$ is covered at least once and variable x_w (5.14g) is equal to one if a vehicle is located at $w \in W$. Similarly to y'_b, variable y''_b (5.14f) is equal to one only if building b is covered at least twice. This variable is taken from [13] as it improves the coverage in a case where multiple alarms are detected.

Most of the hard constraints define a correct behaviour of decision variables. Constraint stating, that a building b must be covered at least once before it can be covered at least twice (5.14d) is the same as in (4.3e). Similarly, constraint (5.14c) is a variant of (4.3d). Finally, constraint (5.14b) enforces the total number of used vehicles.

The only remaining part is the objective function (5.14a), that is composed of 3 different soft constraints. First, the $\sum_{b \in B} y'_b$ counts the number of buildings covered at least once, secondly $\sum_{b \in B} y''_b$ counts the number of buildings covered at least twice. Thirdly,

$\sum_{w \in W} x_w \bar{p_1}(w)$ summarizes the penalty of chosen idle vehicle locations. The importance of each component can be tuned by M_1, M_2 and M_3 constants. We chose a cascading pattern, first of all, maximize the number of buildings covered at least once, then maximize the number of buildings at least twice and then minimize the arrival time penalty. This corresponds to

$$M_1 = M_2 \cdot |B|$$
$$M_2 = M_3 \cdot p$$
$$M_3 = 1$$

Because $\bar{p_1}(w)$ is normalized to the interval $[0, 1]$ and constraint (5.14b), the maximum value of $\sum_{w \in W} x_w \bar{p_1}(w)$ is equal to p, therefore $M_2 = p$. Similarly, the maximum value of $\sum_{b \in B} y''_b$ is equal to all buildings covered at least twice, which is $|B|$, thus $M_1 = M_2 \cdot |B|$.

In the end, we are interested only in values of decision variables x_w which determine positions of idle incident response vehicles. In the static allocation, this program is solved only once and each vehicle is associated with one idle location. Vehicles do not move away from those locations apart from when they are dispatched. Once they finish an alarm inspection, they return to their associated idle location. One possible limitation of this formulation is that it cannot assign more vehicles to the same idle location.

5.3.4.2 Dynamic allocation

Dynamic allocation is an extension of a static allocation. The difference is, that the dynamic allocation re-computes positions of idle vehicles when the set of idle vehicles or the set of idle monitored buildings updates at time t_i. By idle monitored building, we mean a building where no alarm is currently triggered.

To distinguish a set of all buildings B from a set of idle buildings at time t_i, we label this set of idle buildings as $\bar{B_i}$. Similarly the number of idle vehicles changes over time, so we label it as n_i instead of p. The optimization problem is formulated as an extension of static allocation (5.14a) - (5.14g) integer linear program.

$$max \quad M_1 \sum_{\bar{b} \in \bar{B}_i} y'_{\bar{b}} + M_2 \sum_{\bar{b} \in \bar{B}_i} y''_{\bar{b}} - M_3 \sum_{w \in W} x_w \bar{p_{1i}}(w) - M_4 \sum_{w \in W} \sum_{h=1}^{n_i} z_{hw} \bar{p_{2i}}(h, w) \tag{5.17a}$$

$$\text{subject to} \quad \sum_{w \in W} x_w = n_i \tag{5.17b}$$

$$\sum_{w \in W_{\bar{b}}} x_w \geq y'_{\bar{b}} + y''_{\bar{b}} \qquad \bar{b} \in \bar{B}_i \tag{5.17c}$$

$$y''_{\bar{b}} \leq y'_{\bar{b}} \qquad \bar{b} \in \bar{B}_i \tag{5.17d}$$

$$\sum_{w \in W} z_{hw} \leq 1 \qquad h \in 1, \ldots, n_i \tag{5.17e}$$

$$\sum_{h=1}^{n_i} z_{hw} \geq x_w \qquad w \in W \tag{5.17f}$$

$$y'_{\bar{b}} \in \{0, 1\} \qquad \bar{b} \in \bar{B}_i \tag{5.17g}$$

$$y''_{\bar{b}} \in \{0, 1\} \qquad \bar{b} \in \bar{B}_i \tag{5.17h}$$

$$x_w \in \{0, 1\} \qquad w \in W \tag{5.17i}$$

$$z_{hw} \in \{0, 1\} \qquad w \in W, h \in 1, \ldots, n_i \tag{5.17j}$$

The meaning of penalty function (5.15) stays the same, however, it is defined only on idle buildings $\bar{B}_i$.

$$p_{1i}(w) = \sum_{b_w \in \bar{B}_{iw}} cost(SP(\hat{v_w}, \hat{v_{b_w}}))^2 \tag{5.18}$$

Similar holds for its normalized variant.

$$\alpha_i = \max_{w \in W} p_{1i}(w)$$
$$\bar{p_{1i}}(w) = \frac{p_{1i}(w)}{\alpha_i} \tag{5.19}$$

To minimize the reallocation time, we define a cost function $p_{2i}(h, w)$ that assigns a reallocation cost to each idle vehicle $h \in 1, \ldots, n_i$ and potential idle location $w \in W$.

$$p_{2i}(h, w) = cost(SP(\hat{v_{l_{ih}}}, \hat{v_w}))^2 \tag{5.20}$$

Where the location of vehicle h at time t_i is l_{ih} and $\hat{v_{l_{ih}}}$ corresponds to its snapped location to the road network graph. Again, since the range of $p_{21}(h, w)$ depends on current positions of vehicles and road network distances, we normalize it to $[0, 1]$ interval.

$$\beta_i = \max_{w \in W; h \in 1, \ldots, n_i} p_{2i}(h, w)$$
$$\bar{p_{2i}}(h, w) = \frac{p_{2i}(h, w)}{\beta_i} \tag{5.21}$$

In this linear program extension, we added $n_i \cdot |W|$ new binary variables (5.17j). The variable z_{hw} equals one only if the vehicle $h \in 1, \ldots, n_i$ reallocates to idle location $w \in W$. The proper behaviour of this variable is enforced with $|W| + n_i$ additional hard constraints. The constraint (5.17e) assures that each vehicle is reallocated to at most one idle location. Lastly, the constraint (5.17f) is satisfied only if the idle location $w \in W$ is selected when at least one idle vehicle reallocates to w. All remaining hard constraints are covered in static allocation in section 5.3.4.1.

As for the objective function (5.17a), there is one additional soft constraint

$$\sum_{w \in W} \sum_{h=1}^{n_i} z_{hw} \bar{p_2}_i(h, w)$$

which minimizes the reallocation time. Similarly to static allocation importance of each soft constraint can be tuned by M_1, M_2, M_3 and M_4. However, we use greedy cascading pattern

$$\begin{aligned} M_1 &= M_2 \cdot |\bar{B}_i| \\ M_2 &= M_3 \cdot n_i \\ M_3 &= M_4 \cdot n_i \\ M_4 &= 1 \end{aligned}$$

that is described in static allocation, section 5.3.4.1.

The result obtained by solving the linear program (5.17a) - (5.17j) is a new idle location $loc(h)$ for each vehicle $h \in 1, \ldots, n_i$ that is defined as

$$loc(h) = w \quad \text{iff} \quad z_{hw} = 1.$$

Similarly to the static allocation, this formulation cannot allocate multiple vehicles to one idle location.

Chapter 6

Implementation

The system defined in section 5.3 was implemented and evaluated using technologies described in section 6.2. In order to run real-world scenarios, we use publicly available road network data as pointed out in section 6.1.

6.1 Data collection

Unlike synthetic scenario, where the road network was made purposefully simple, in real-world scenario we use map of Prague. We use OpenStreetMap (see section 6.1.1) as it offers all needed map features and is compatible with some of the technologies that we use.

6.1.1 OpenStreetMap

OpenStreetMap [22] is an editable map database maintained by volunteers. This geospatial information is collected through car trips, jogs, photos, videos or GPS traces. This project was motivated by the limited availability of maps, that required a university-level degree and expensive equipment to keep them accurate and current, and rapid growth of low-cost GPS receivers [23]. Our main use of OpenStreetMap data is to extract locations of buildings and parking lots using OSMonaut (see section 6.2.4) and to compute shortest paths between multiple locations using Graphhopper (see section 6.2.3).

6.1.2 Security data

We were able to collect data from an unnamed security company, but due to the sensitive nature of the data, that could affect business model of this company, it remains classified. It was used to better understand the security domain and problems they are dealing with. Our simulation is sufficient as it captures the most important properties of incident response vehicle operations.

6.2 Technologies

In following sections, we introduce technologies that were used in this work. Our system, as described in section 5.3, is implemented in Kotlin and tested using JUnit. We use Graphhopper and OSMonaut libraries when working with OpenStreetMaps and Gurobi for solving integer linear programs. Finally, we use JSON format to transfer data to Leaflet and Matlab® which both serve for data visualisation.

6.2.1 Kotlin

Most of this work was implemented in Kotlin. Kotlin is a fairly new programming language, the first version was released on February 15, 2016 [24]. It is developed as an open source language, but the language design and the overall steering of the project are done by JetBrains team. One of its key focuses is interoperability with popular Java language that allows mixing Java and Kotlin together in one project. We benefit from this property as we use several Java libraries and frameworks: Graphhopper, OSMonaut, Maven or JUnit. Some of Kotlin benefits are null-safety, operator overloading, extension functions, range expressions or data classes.

6.2.2 JUnit

JUnit is a Java unit testing framework. It is used to assure proper functionality of the application. This testing is done by targeting small pieces of code where external dependencies are removed.

6.2.3 Graphhopper

Graphhopper [25] is an open source routing library written in Java. It tackles many problems, one of them is described in our work as snapping, in section 5.2.2, and provides simple user web interface called Graphhopper maps[1]. It can be configured to use different graph exploring algorithms such as Dijkstra or A*. In this work, we have integrated a custom version of uniform search algorithm (1) and used this library to find shortest paths between multiple locations. By default, it uses OpenStreetMap data.

6.2.4 OSMonaut

OSMonaut [26] is an OpenStreetMap data parser framework written in Java. It offers a way of converting raw OpenStreetMap data into other file formats. We use this tool to extract locations of buildings and parking lots.

[1] <https://graphhopper.com/maps/>

6.2.5 Gurobi

Gurobi [27] is a commercial state-of-the-art mathematical programming solver. It covers optimization problems such as linear programming, mixed-integer programming, quadratic programming, mixed-integer quadratic programming and more. It offers application programming interfaces into many languages amongst which is also Java. We use Gurobi version 7.5.2 to solve integer linear programs formulated in section 5.3.4.

6.2.6 Maven

Apache Maven was used as a software project management tool. It manages project's dependencies, build and documentation from central piece of information. Maven is mainly used in Java projects, but it is also supported by Kotlin.

6.2.7 Matlab®

Matlab® [28] – an abbreviation for Matrix laboratory – is a matrix-based programming language and programming environment. It specializes in matrix operations which allow most natural expression of computational mathematics. It offers interactive graphical output. The main use of Matlab® in this work was for plotting simulation data.

6.2.8 Leaflet

Leaflet [19] is an open source JavaScript library used to build maps for web applications. It provides a way of displaying interactive layers, such as markers, points, polygons, paths and more, on top of maps. If offers basic controls such as zooming or dragging, that is supported on multiple platforms. Leaflet was used to visualize routing data in an easy to read form.

Chapter 7

Evaluation

7.1 Synthetic scenarios

The ability of vehicles to respond to triggered alarms is first measured in a small, controlled and synthetically generated environment. Description of this environment is given in section 7.1.1, followed by results from static and dynamic approaches, in sections 7.1.2 and 7.1.3 respectively.

7.1.1 Environment

Synthetic road graph contains 25 nodes that are arranged into a 5 by 5 grid. This grid contains only vertical and horizontal edges. While all horizontal edges are bidirectional, all vertical edges are unidirectional and the direction alternates each column. This pattern can be seen in figure 7.1. The length of each edge is 1 km with 60 km/h travel speed resulting in 1 minute travel time. This property causes discontinuities in following graphs, due to the discrete arrival times of vehicles that are located directly at nodes. Moreover, each node is considered as a monitored building $b \in B$ and a possible idle incident vehicle location $w \in W$. Both alarm time and location are drawn from the uniform distribution.

7.1.2 Static scenario

In the static scenario, each incident response vehicle is associated with and placed in its idle location, in the beginning, using approach defined in section 5.3.4.1. Vehicles do not move apart from when they are dispatched to an alarm event or when they return from an alarm event to their associated location.

Firstly, we present examples of chosen idle vehicle positions in figure 7.1. The solution for one vehicle 7.1a is trivial and the location is in the middle. Positions for two vehicles 7.1b might seem a bit counter-intuitive as one would expect different locations (same as in 7.1d). However this is caused by the one-way middle vertical road, the other placement would result in higher arrival times in the bottom row. In the case of four vehicles 7.1c, the two locations in the middle vertical road already cover every building twice (as seen in 7.1b), therefore additional vehicles are placed such that their sum of squared arrival times is minimal.

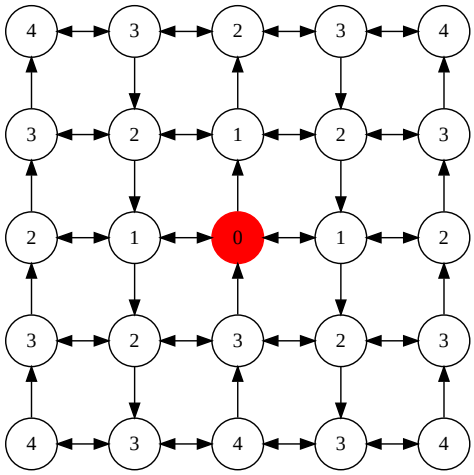
Secondly, a 14 days worth simulation is run with 1 vehicle covering the varying amount of triggered alarms 7.2. This process was repeated 50 times to minimize the impact of 'very good' or 'very bad' alarm sequences. An example of a 'bad' alarm sequence would be alternating alarms on the opposite side of the grid, while 'good' alarm sequence would be alarms appearing in the close neighbourhood. As expected, one vehicle performs well when the alarm frequency is low – it can cover about 96% of all alarms within arrival time threshold. With increasing frequency , the vehicle start falling behind, since it processes the alarms in the order they were triggered.

Lastly, we compare the performance of varying amount of vehicles 7.3 in a similar simulation as described above. Again, one vehicle fails to handle high demand and covers only 26% of requests in time. However, one additional vehicle boosts the coverage to 88%. Three vehicles manage to handle 98% of requests and adding additional vehicles does not increase the performance much.

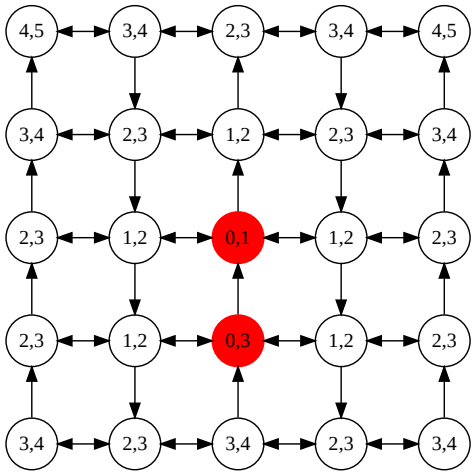
7.1.3 Dynamic scenario

In contrast to the static scenario, in the dynamic scenario, we allow idle vehicles to move from their original position to some other potential idle location. The idea behind this is that they would fill in for currently intervening vehicles which would increase the coverage. As seen in the results 7.4, there is a slight improvement – about 6% more alarms covered within 2 minutes arrival time. However this assumes that we have more vehicles at our disposal, otherwise, as in the case of only 2 vehicles, the performance is the same.

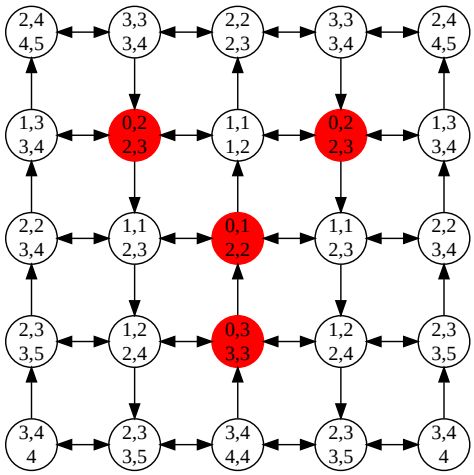
Other that that, we measured distance driven by idle vehicles 7.5. That is the distance they drive when reallocating from one idle position to another or when they return from an inspected alarm event. In the case of dynamic optimization, the mileage is almost 3 times higher than in the static case. In real-world, this approach requires a lot of extra organization. Due to our assumptions about ideal simulation (6) (section 5.3.2) and ideal travel conditions (1) (section 5.2.1), the application of the dynamic approach to the real-world would bring additional disturbance because longer driven distance means higher error caused by assumptions.



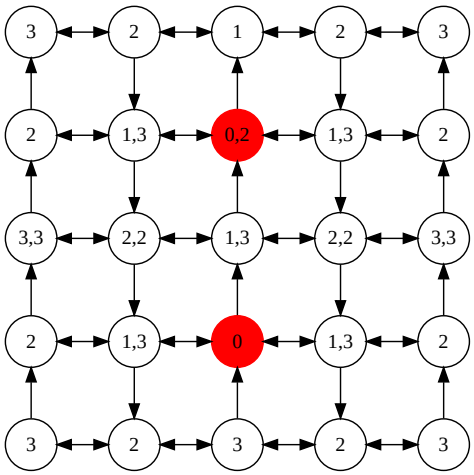
(a) Solution for 1 vehicle, 5 minute threshold



(b) Solution for 2 vehicles, 5 minute threshold



(c) Solution for 4 vehicles, 5 minute threshold



(d) Solution for 2 vehicles, 3 minute threshold

Figure 7.1: Optimal static idle incident response vehicle positions for synthetic graph

The figure shows optimal positions of idle incident response vehicles (in red) and arrival times within time threshold (node labels) computed using static allocation. Each node is considered both as a monitored building $b \in B$ and a potential idle vehicle location $w \in W$. Please note that horizontal roads are bi-directional and vertical roads are unidirectional. Positions in figures 7.1a, 7.1b, 7.1c are optimized for arrival time within 5 minutes, whereas figure 7.1d for arrival time within 3 minutes.

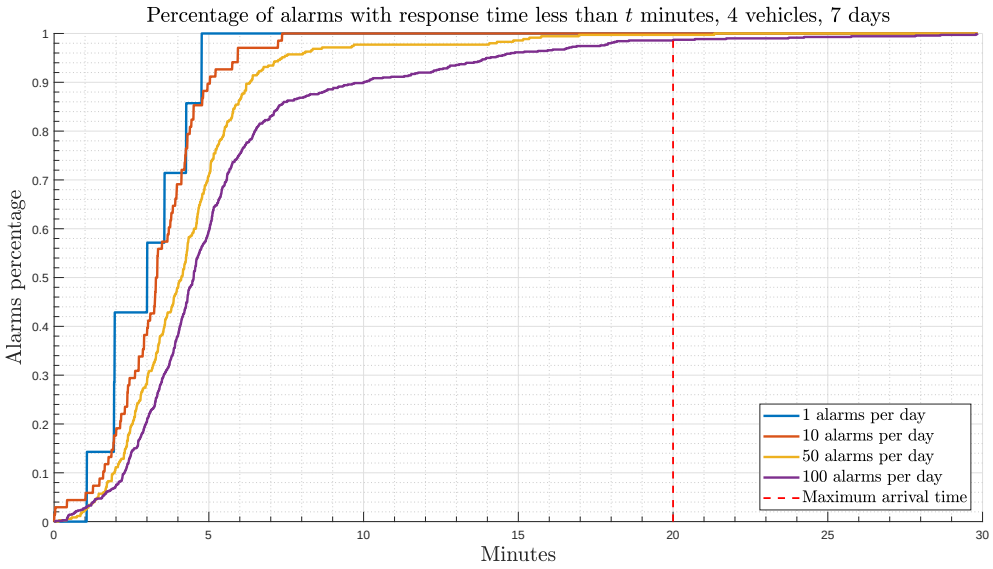


Figure 7.2: Arrival times on synthetic data with varying alarm frequency

Varying alarm frequency is evaluated in a simulation with 14 days time span and 1 incident response vehicle. This simulation is repeated 50 times with different alarm sequences and results are averaged. The inspection time is set to constant 2 minutes. Results show, that 1 vehicle is able to respond to 96% of alarms within the arrival time threshold of 5 minutes when the alarm frequency is low (25 alarm per day on average). With increasing frequency, the percentage of covered alarms within 5 minutes drops. Discontinuities are caused by an idle vehicle on a node, because it has discrete arrival times of 0, 1, . . . , 5 minutes.

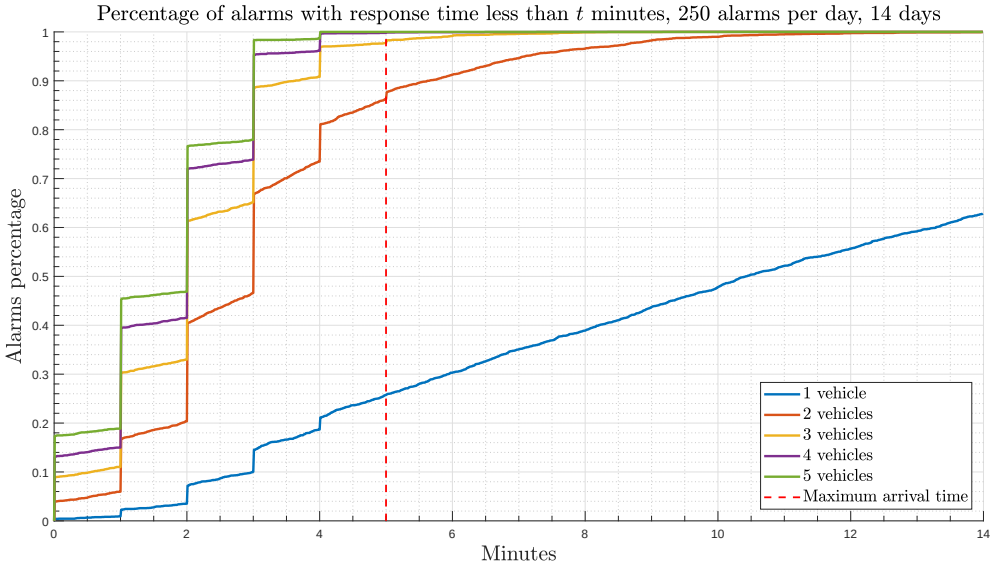


Figure 7.3: Arrival times on synthetic data with varying number of used vehicles

The varying number of vehicles is evaluated in a simulation with 14 days time span and 250 alarms, on average, triggered each day. This simulation is repeated 50 times with different alarm sequences and results are averaged. The inspection time is set to constant 2 minutes. Results show, that 1 vehicle is able to respond to only 26% of alarms within 5 minutes arrival time threshold. When 2 vehicles were used, 88% of those alarms are covered. Finally, the coverage was above 98% when 3, 4 or 5 vehicles were used. Discontinuities are caused by an idle vehicle on a node, because it has discrete arrival times of 0, 1, . . . , 5 minutes.

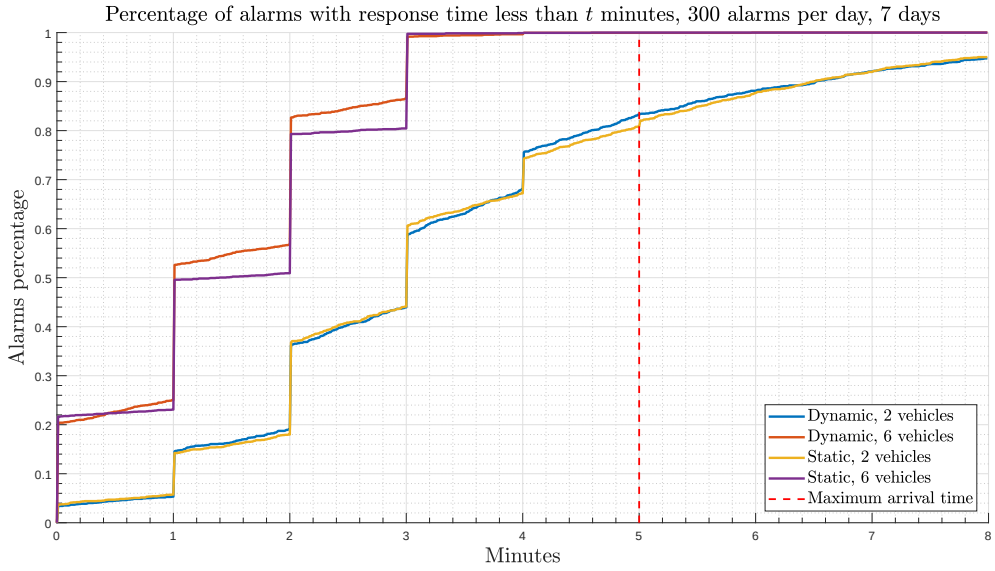


Figure 7.4: Difference between static and dynamic approach

In dynamic approach, we allow reallocation of idle vehicles to different potential idle locations to substitute for currently intervening vehicles. The increase in coverage only happens when we have enough vehicles to work with. In case of 2 vehicles, they fall behind and results are very similar. On the other hand, dynamic approach managed to cover 6% more alarms within 2 minute arrival time. However at the cost of mileage (see figure 7.5). Discontinuities are caused by an idle vehicle on a node, because it has discrete arrival times of 0, 1, ..., 5 minutes.

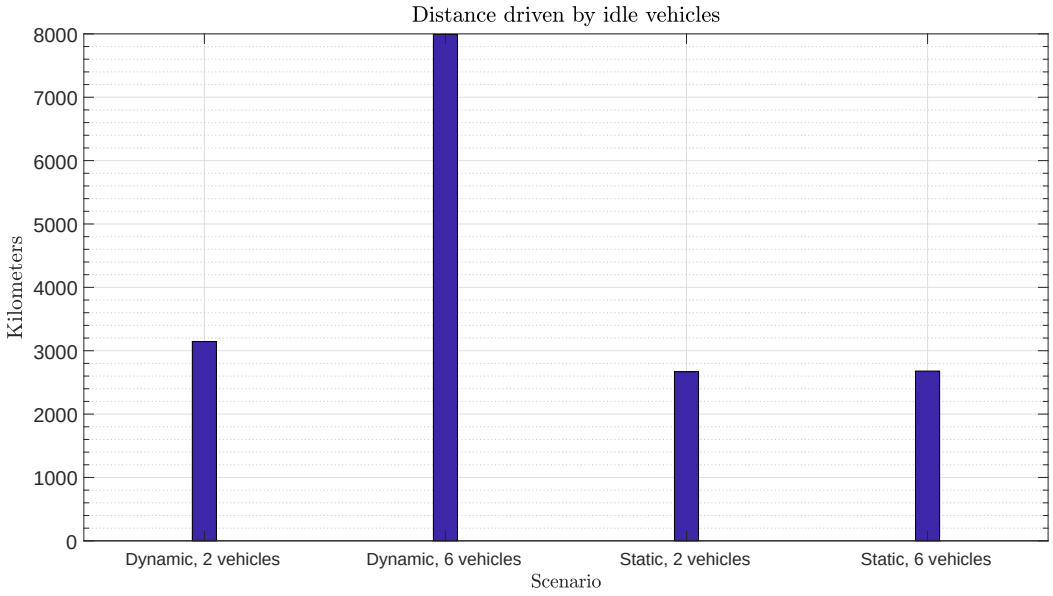


Figure 7.5: Distances driven by idle vehicles in static and dynamic approaches

The slight coverage increment of the dynamic approach shown in figure 7.4 comes at a high price – additional mileage that is almost 3 times higher than in the static approach.

7.2 Real-world scenarios

Rather than using unit distances and grid graph layout as in synthetic scenarios, in real-world scenarios, we utilize real road connections, distances and speed limits. Moreover, we use real buildings for monitoring and real parking lots as locations for idle vehicles. Furthermore, we employ arrival and inspection time that is close to reality.

7.2.1 Environment

We defined two real-world scenarios on a road network of the Prague city – specifically smaller red instance and bigger blue instance in sections 7.2.2 and 7.2.3 respectively. In order to represent this city as close to reality as possible, we used publicly available OpenStreetMap data – more about OpenStreetMap in section 6.1.1. This map contains road network with speed limits, buildings and parking lots. We used this information to construct a set of monitored buildings B and set of possible idle vehicle locations W.

However, due to incompleteness or generalisation in the data, it is not trivial to conclude, whether a parking lot is available for public use. In following experiments, we assume, that we are able to park idle incident vehicles in all of them.

We set arrival threshold to 20 minutes and the inspection time – that is the time the vehicle needs to wait in order to properly inspect the cause of an alarm – also to 20 minutes[1], as these values closely correspond to reality. Additionally, we use four different frequencies of alarm sequences, namely uniformly sampled number of alarms corresponding to 0.1%, 1%, 5% and 10% of monitored buildings daily. Please note, that 5% and 10% are extreme cases and very rarely happen in reality.

7.2.2 Red instance

The smaller red instance roughly corresponds to Prague 3 city district (see figure 7.6), more specifically it is a bounding box with top left tl and bottom right br corners specified by following GPS coordinates in degrees.

$$tl = (50.0913156, 14.4421000)$$

$$br = (50.0779717, 14.5035300)$$

It contains 59 parking lots, that were selected as the set of potential idle vehicle locations W. Furthermore, we uniformly sampled 1000 out of extracted buildings as the set of monitored properties B. Therefore the number of sampled alarm each day of the simulation equals to $1, 10, 50$ and 100.

[1] The inspection time is not known before or during the inspection, similarly as in real-world. We only get notified, once the inspection ends. It could easily be a different value for each inspection, however, we use a constant for simplicity.

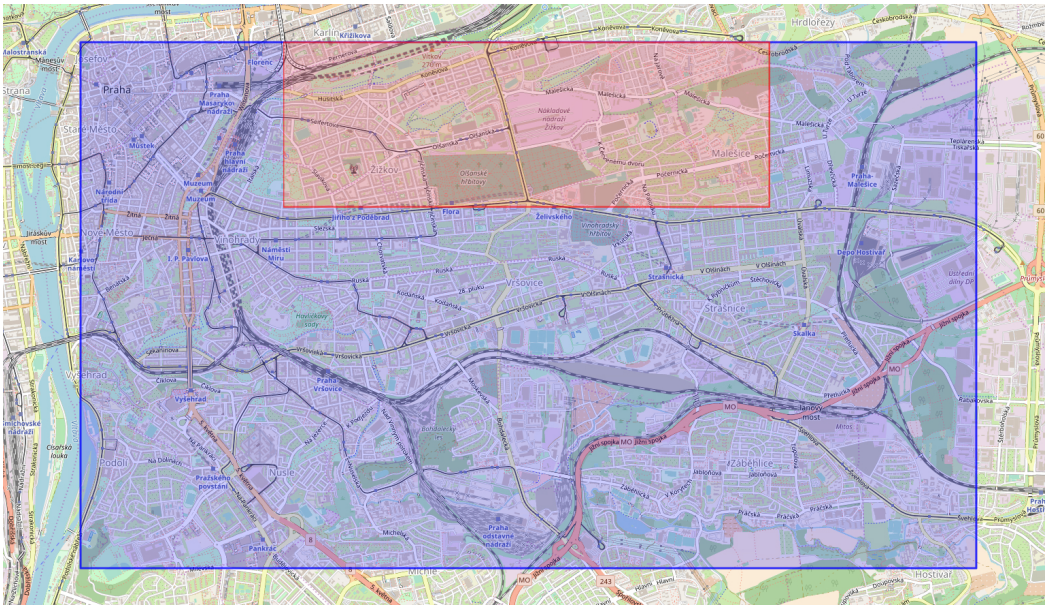


Figure 7.6: Real-world instances

We evaluated our approach on two real-world instances. First, a smaller one (red bounding box) that roughly corresponds to Prague 3 city district which has 59 parking lots and 1000 monitored buildings. Second instance (blue bounding box) roughly corresponds to city districts Prague 2, Prague 3, Prague 10 and partly Prague 1 and Prague 4. It contains 247 parking lots and 4000 monitored buildings.

7.2.3 Blue instance

The bigger blue has a bounding box with top left tl corner and bottom right br corner that is specified by GPS points in degrees below. As depicted in figure 7.6, it roughly corresponds to city districts Prague 2, Prague 3, Prague 10 and partially Prague 1 and Prague 4.

$$tl = (50.0913156, 14.4166511)$$

$$br = (50.0487414, 14.5296522)$$

From this bounding box, we extracted 247 parking lots as the set of potential idle vehicle locations W and uniformly chose 4000 buildings as the set of monitored buildings B. The number of daily sampled alarms equals to $4, 40, 200$ and 400.

7.2.4 Static scenario

In the static scenario, each vehicle is assigned one parking lot $w \in W$ and due to the definition of static linear program in section 5.3.4.1, there can be at most one vehicle on a single parking lot. Once an alarm is activated, the closest vehicle is dispatched and after arriving at the location of the alarm, it performs a 20-minute inspection. Then, it returns to its assigned parking lot.

First, we evaluated this scenario on the red instance. The percentage of alarms, that was handled within 20 minutes is shown in table 7.1. A single vehicle performs well when

the frequency is low (0.1% and 1%), but the performance drops significantly at and above 5%. This is caused by the duration of an inspection, 5% alarm frequency corresponds to 50 alarms daily, each one requires a 20 minutes long inspection which equals 1000 minutes in inspections only. Provided that a day has 1440 minutes, there is no doubt about the poor performance.

Intuitively, adding more vehicles helps. Three vehicles can handle up to 86.84% of alarms with the highest frequency within 20 minutes. Ultimately, five vehicles handle 100% of all alarm frequencies, therefore adding more vehicles yields no additional improvement.

Secondly, the results from the blue instance in table 7.2 show similar phenomena for 5 vehicles as in the red instance.

	Percentage of buildings with daily alarms			
Number of vehicles	**0.1%**	**1%**	**5%**	**10%**
1	100%	91.18%	18.34%	0.17%
2	100%	98.53%	85.43%	30.07%
3	100%	100%	98%	86.84%
4	100%	100%	99.71%	98.57%
5	100%	100%	100%	100%

Table 7.1: Percentage of alarms handled within 20 minutes on the red instance, static allocation

Since the red instance contains 1000 buildings, the percentage 0.1%, 1%, 5%, 10% correspond to 1, 10, 50 and 100 alarms daily.

	Percentage of buildings with daily alarms			
Number of vehicles	**0.1%**	**1%**	**5%**	**10%**
5	92.86%	91.43%	58.60%	0.32%
10	100%	98.93%	98.28%	80.02%
15	100%	100%	99.71%	98.96%

Table 7.2: Percentage of alarms handled within 20 minutes on the blue instance, static allocation

Since the blue instance contains 4000 buildings, the percentage 0.1%, 1%, 5%, 10% correspond to 4, 40, 200 and 400 alarms daily.

7.2.5 Dynamic scenario

In dynamic scenarios, idle vehicles may re-allocate to a different idle location as opposed to static allocation where they use only the one, which was assigned to them at the start.

The percentage of covered buildings within arrival time of 20 minutes is presented in tables 7.5 and 7.6 for red instance and blue instance respectively. More importantly, the absolute increase in the favour of dynamic scenarios over static ones is shown in tables 7.3 and 7.4.

In latter tables, there are three distinguishable regions that share the same properties. Firstly, when the fleet size is big enough, such that the static allocation handles most of the

request in the given alarm frequency in time, there is no real benefit in dynamic allocation as there is a little room for improvement. This sector corresponds to the entries on or below the main diagonal.

Secondly, when the fleet size is small and alarm frequency high, the inspection time becomes a bottle neck. Since all vehicles are probably deployed at the same time, there are no remaining vehicles that could substitute for intervening ones. This sector corresponds to the upper right corner.

Thirdly, there is a combination of the fleet size and alarm frequency, that lies in between the first and the second region. Namely 2 vehicles with 5% alarm frequency and 3 vehicles with 10% alarm frequency on the red instance and 5 vehicles with 5% alarm frequency and 10 vehicles with 10% alarm frequency on the blue instance. In these experiments, there remains enough idle vehicles that the dynamic approach handles more alarms within the arrival time, more precisely by +4%, +5.72% more on the red instance and by +21.7%, +15.22% more on the blue instance.

	Percentage of buildings with daily alarms			
Number of vehicles	**0.1%**	**1%**	**5%**	**10%**
2	0%	+1.47%	+4%	+2.35%
3	0%	0%	+1.43%	+5.72%
4	0%	0%	−0.28%	+0.29%
5	0%	0%	0%	−0.29%

Table 7.3: The increase in the percentage of alarms handled within 20 minutes in the favour of dynamic allocation, red instance

	Percentage of buildings with daily alarms			
Number of vehicles	**0.1%**	**1%**	**5%**	**10%**
5	+7.14%	+8.57%	+21.7%	+1.73%
10	0%	+1.07%	+1.72%	+15.22%
15	0%	0%	+0.29%	+1.04%

Table 7.4: The increase in the percentage of alarms handled within 20 minutes in the favour of dynamic allocation, blue instance

	Percentage of buildings with daily alarms			
Number of vehicles	**0.1%**	**1%**	**5%**	**10%**
2	100%	100%	89.43%	32.42%
3	100%	100%	99.43%	92.56%
4	100%	100%	99.43%	98.86%
5	100%	100%	100%	99.71%

Table 7.5: Percentage of alarms handled within 20 minutes on red instance, dynamic allocation

Since the red instance contains 1000 buildings, the percentage 0.1%, 1%, 5%, 10% correspond to 1, 10, 50 and 100 alarms daily.

	Percentage of buildings with daily alarms			
Number of vehicles	**0.1%**	**1%**	**5%**	**10%**
5	100%	100%	80.30%	2.05%
10	100%	100%	100%	95.24%
15	100%	100%	100%	100%

Table 7.6: Percentage of alarms handled within 20 minutes on blue instance, dynamic allocation

Since the blue instance contains 4000 buildings, the percentage 0.1%, 1%, 5%, 10% correspond to 4, 40, 200 and 400 alarms daily.

7.3 Road network

In order to speed up computation, in section 5.2.2, we assumed (assumption 3) that distances between two points are small. We support this assumption by measuring distances between relevant pairs of points, that is all snapping distances and distances between nodes lying on an edge. These distances are displayed in figure 7.8.

Another possible error that is introduced with the real-world road network and Graphhopper happens during a computation of vehicle's location – this process is described in section 5.3.2.1. Since we are using linear interpolation to determine the position and then snapping this position to the graph, it is possible, that the snapped point will not lie on the same edge. This may happen when the true location is on a bridge, but the snapped location is under a bridge as depicted in figure 7.7a. A similar problem may occur when the resolution of the graph is low as the snapped point might end up on different road – figure 7.7b. In both of these cases, the vehicle might spend a not negligible time to recover and return to its original route.

(a) Snapping failure on a bridge

(b) Snapping failure in a turn

Figure 7.7: Possible snapping failures

In some special cases, the process of determining a vehicle position as described in section 5.3.2.1 might fail. In situation depicted in 7.7a the vehicle is driving over a bridge (blue line), but when we try to determine its position shown by a red circle, the closest node might be under the bridge. A similar situation happens in a sharp turn, figure 7.7b, where the position (red circle) might end up in the opposite direction. Visualizations made by Leaflet [19].

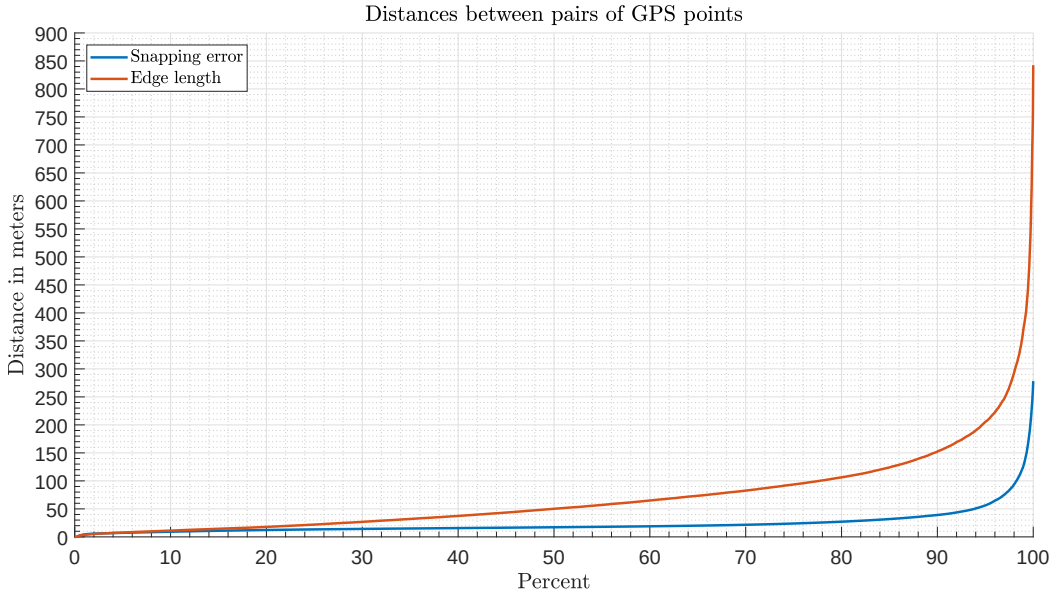


Figure 7.8: Distances between pairs of points

In the assumption (3) (section 5.2.2), we assumed, that distances between two points are small. To support this assumption, we calculated both snapping distances and edge lengths on our road network. The graph shows, that approximately 94% of snapped distances is below 50 meters. The distances between nodes on edges are a bit larger, circa 90% is below 150 meters.

Chapter 8

Conclusion

The goal of this thesis was to propose and evaluate two approaches for incident response vehicle fleet management in building security domain. These vehicles are distributed around the region and in case an alarm is detected, one of the vehicles is dispatched to investigate a cause of the alarm. A good fleet management should maximize the coverage of protected buildings while keeping the arrival time reasonably low. First approach – called static allocation – should allocate the vehicles prior any alarm is detected. On the other hand, the second approach – dynamic allocation – should react to alarm events, such that they are able to substitute currently intervening response vehicles.

We explained the process of alarm detection up to the point of incident vehicle deployment in chapter 3. Additionally, in chapter 4, we studied related problems of facility and vehicle allocation, especially in the Emergency Medical Services domain. Static models in the form of mathematical programs, presented in related work, are either too restrictive and infeasible with a small fleet of vehicles or do not optimize both coverage and arrival time. Finally, we briefly introduced a game-theoretic approach that is to be taken in order to drop the alarm independence assumption (5, section 5.3.1), as there exists a possibility, that the burglar is monitoring positions of incident vehicles and plans its attack based on this observation.

Our approach was formulated in chapter 5, where we also reason about complications that arise when dealing with real-world data. A system consisting of four independent components was designed and implemented in Kotlin programming language. This system, consisting of offline alarm generator, linear integer program solver, event-based simulator and greedy dispatcher, is capable of using both static and dynamic optimization models and work with real-world or synthetic data.

Finally, in chapter 7, we evaluated both of our approaches on synthetic and real-world data. Small and comprehensible synthetic data is used to analyze the two algorithms in greater depth. On the other hand, real-world scenario, taking place in the city of Prague, is evaluated to explore possible benefits of our approach to current building security systems. We show, that in two situations, where the static allocation of a large fleet can handle alarm events easily and where the static allocation of a small fleet cannot handle alarm events at all, there is no real benefit of dynamic allocation. However, between those two extrema, the potential benefit of the dynamic approach is noticeable, as the percentage of covered alarms increased by 15.22% and 21.7% on the larger region with bigger fleet size and by 4% and 5.72% on the smaller region with smaller fleet size.

Bibliography

[1] Policie České republiky. Statistické přehledy kriminality za rok 2017, 2017. URL <http://www.policie.cz/soubor/2017-12-prosinec-sest-01a-xlsx.aspx>. [Online; accessed 3-April-2018].

[2] Eurostat. Crime and criminal justice statistics, 2017. URL <http://ec.europa.eu/eurostat/statistics-explained/images/4/4e/Crime_and_criminal_justice_tables_2017_v2.xlsx>. [Online; accessed 3-April-2018].

[3] Niall McCarthy and Felix Richter. Infographic: Where private security outnumbers the police, Sep 2017. URL <https://www.statista.com/chart/10925/where-private-security-outnumbers-the-police/>. [Online; accessed 27-April-2018].

[4] Jablotron. Product catalogue, JABLOTRON 100, 2018. URL <https://www.jablotron.com/en/katalog-produktu/alarms/jablotron-100/>. [Online; accessed 3-April-2018].

[5] D.I.Seven. Arc – alarm receiving centre, 2018. URL <https://www.diseven.cz/en/arc-alarm-receiving-centre/>. [Online; accessed 3-April-2018].

[6] Jablotron. Security center, 2018. URL <http://www.bezpecnostnicentrum.cz/en>. [Online; accessed 3-April-2018].

[7] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[8] Demlová Marie. Theory of algorithms. University Lectures, 2017. URL <http://math.feld.cvut.cz/demlova/teaching/e-tal/e-tal.pdf>. [Online; accessed 15-May-2018].

[9] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Citeseer, 1976.

[10] Xueping Li, Zhaoxia Zhao, Xiaoyan Zhu, and Tami Wyatt. Covering models and optimization techniques for emergency response facility location and planning: a review. *Mathematical Methods of Operations Research*, 74(3):281–310, 2011.

[11] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations research*, 19(6):1363–1373, 1971.

[12] Richard Church and Charles ReVelle. The maximal covering location problem. In *Papers of the Regional Science Association*, volume 32, pages 101–118. Springer, 1974.

[13] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Solving an ambulance location model by tabu search. *Location science*, 5(2):75–88, 1997.

[14] Karl F Doerner, Walter J Gutjahr, Richard F Hartl, Michaela Karall, and Marc Reimann. Heuristic solution of an extended double-coverage ambulance location problem for austria. *Central European Journal of Operations Research*, 13(4):325, 2005.

[15] Orhan Karasakal and Esra K Karasakal. A maximal covering location model in the presence of partial coverage. *Computers & Operations Research*, 31(9):1515–1526, 2004.

[16] Yisong Yue, Lavanya Marla, and Ramayya Krishnan. An efficient simulation-based approach to ambulance fleet allocation and dynamic redeployment. In *AAAI*, 2012.

[17] Branislav Bošanskỳ, Viliam Lisỳ, Michal Jakob, and Michal Pěchouček. Computing time-dependent policies for patrolling games with mobile targets. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 989–996. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[18] Jean-Paul Rodrigue, Claude Comtois, and Brian Slack. *The geography of transport systems*. Routledge, 2016.

[19] Vladimir Agafonkin. Leaflet version 1.3.1, an open-source JavaScript library for mobile-friendly interactive maps, 2017. URL <https://leafletjs.com/>.

[20] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. ISBN 3540779736, 9783540779735.

[21] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[22] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . <https://www.openstreetmap.org>, 2017.

[23] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[24] Kotlin 1.0 Released: Pragmatic Language for JVM and Android, Feb 2016. URL <https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/>. [Online; accessed 19-May-2018].

[25] Peter Karich and S Schröder. Graphhopper. <http://www.graphhopper.com>, 2018.

[26] OSMonaut. URL <https://github.com/MorbZ/OSMonaut>. [Online; accessed 19-May-2018].

[27] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016. URL <http://www.gurobi.com>.

[28] *MATLAB version 9.2.0.556344 (R2017a)*. The Mathworks, Inc., Natick, Massachusetts, United States, 2017. URL <http://www.mathworks.com/>. [Online; accessed 19-May-2018].

Appendix A

CD content

The attached CD contains following directories:

maps/ road map network used in real-world scenarios

maps/graphhopperCache/ map data cached by GraphHopper

matlab/ Matlab® functions used to generate graph visualisations

persistence/ precomputed location data

src/ Kotlin source code

thesis/ thesis LaTeXsource code

thesis/images/ figures in high resolution

A.1 Evaluation description

The evaluation is implemented in JUnit test environment and located in

src/test/kotlin/thesisdata.

In order to compile and run the evaluation, one needs to provide a path to Gurobi Java Archive in the file **pom.xml** on line *25*. For example in order to run changing frequency experiment on synthetic data, one runs a test

src/test/kotlin/thesisdata/SyntheticScenarios.changingFrequency().

The evaluation creates a new directory **results**/ and stores the results of an evaluation in an appropriate directory structure that corresponds to the experiment. In this folder, files in JSON format will be generated and their structure corresponds to

src/main/kotlin/logging/SimulationLog.

Additionally, one JSON file **experiment.json** is generated, that corresponds to

src/main/kotlin/logging/ResponseExperiment.

The script also automatically generates a visualisation using MATLAB®, if a path to the matlab executable is provided in

src/main/kotlin/Constants.kt

and Matlab contains folder **matlab/** in its path.