Master Thesis

Dynamic Emergency Ambulance Fleet Allocation and Management

Niklas Meinzer

Albert-Ludwigs-University Freiburg im Breisgau
Faculty of Engineering
Department of Computer Science
Chair of Algorithms and Data Structures

Bearbeitungszeitraum
15. 11. 2013 – 15. 05. 2014

Gutachter
Prof. Dr. Hannah Bast
Prof. Dr. Christian Schindelhauer

Betreuerin
Dr. Sabine Storandt

Danksagung

Ich möchte mich bei einigen Personen bedanken, die mich bei der Arbeit an dieser Masterarbeit unterstützt haben. Großer Dank gilt Frau Prof. Bast, die mich mit ihren hervorragenden Vorlesungen durch mein ganzes Studium motivieren und für das Themengebiet Algorithmen und Datenstrukturen begeistern konnte. Bei Dr. Sabine Storandt möchte ich micht bedanken für die hervorragende Betreuung über den gesamten Bearbeitungszeitraum. Vielen Dank auch an Herrn Prof. Schindelhauer, der sich freundlicherweise als Zweitgutachter für diese Arbeit zur Verfügung gestellt hat.

Besonderer Dank gilt Nikolaus Mayer und Fabian Nedic, die sich die Zeit genommen haben, meine Ausführungen nach inhaltlichen und grammatischen Fehlern zu durchsuchen.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum	Unterschrift

Contents

Abstract

Most countries have implemented some form of Emergency Medical Services (EMS) in order to help people with urgent medical needs. Typically a number of ambulances serve a specific geographic region by responding to emergency calls, treating patients and transporting them to a hospital. Since the number of ambulances is limited and emergency calls need to be responded to quickly by nature, EMSs have to be thoroughly managed and coordinated, which is usually done by human operators in special call centers. In this thesis we model that task and present and compare different strategies which could be employed by an automatic ambulance fleet management system. We show that continuous optimization of ambulance distribution over the region and dynamic reassignment of ambulances to incoming requests can benefit both the patients and, economically, the provider of the EMS.

Zusammenfassung

Die meisten Länder der Welt unterhalten einen medizinischen Rettungsdienst, um Menschen in medizinischen Notsituationen zu helfen. Ein Rettungsdienst besteht üblicherweise aus einer Gruppe von Rettungsfahrzeugen, die eine bestimmte Gegend bedienen und dort zu Notfallpatienten fahren, diese behandeln und in eine medizinische Einrichtung transportieren. Diese Rettungsfahrzeuge stellen eine limitierte Ressource dar und müssen daher sorgfältig koordiniert und gesteuert werden, um sicherzustellen, dass Notrufe schnell beantwortet werden können. Diese Aufgabe wird überlicherweise von speziell dafür ausgebildeten Disponenten in Rettungsleitstellen übernommen. Diese Arbeit stellt anhand eines Modells verschiedene Managementstrategien für den Rettungsdienst einer Region vor. Es wird gezeigt, wie durch gleichmäßiges Verteilen der Rettungsfahrzeuge auf dem Straßengraph und geschicktes Neuverteilen der Rettungsfahrzeuge bei neueintreffenden Notrufen, die Leistung des Rettungsdienstes unter medizinischen und ökonomischen Gesichtspunkten verbessert werden kann.

1 Introduction

1.1 Motivation

Emergency Medical Services (EMS) are a key component of modern societies and ensure that people with urgent medical needs quickly receive help. While the details vary widely from one country to another, transporting the patient to a hospital or medical center using ambulances is always a key part in the system.

The number of ambulances in a certain area (e.g. a city) is limited, but to respond to an emergency call a free ambulance is needed. Thus the ambulance fleet needs to be carefully managed. This is usually done by specially trained phone and radio operators in emergency dispatch centers. These operators have to decide which vehicle they want to send to a specific emergency, determine the destination hospital and possibly re-distribute the remaining ambulances. The environment in which they operate is characterized by a high degree of uncertainty, as little can be known about future emergency requests at any given time.

The main goal of an EMS system is to provide help in medical emergencies and save as many lives as possible. However the operation of an EMS system is also very cost intensive and being a part of traditionally tightly budgeted healthcare systems, economical aspects must also be taken into account when examining the performance of an EMS system. Some cost factors such as salaries for paramedics or purchase cost of medical equipment are subject to the location of the EMS system and its economical environment. A big cost factor, which applies to all regions of the world, is fuel, which can only be minimized if the ambulances drive optimal routes at all times.

In this thesis we present different strategies to manage an ambulance fleet and examine their performance in various scenarios. These strategies could eventually be implemented in a supervised automated form or as assistance systems to human operators. We model the EMS setting in a general way, such that it can be applied to all kinds of geographic regions. We created an event based simulator to test the strategies in different environments and under different settings. To visualize the model and results and to observe the simulation we created a graphical user interface (GUI) using a combination of traditional GUI programming and web tools.

1.2 Problem

The problem we seek to solve with our contribution is the automated management of an ambulance fleet. We are given a number of ambulances, ambulance bases and hospitals on a road network. Furthermore there is a stream of requests by patients that need to be brought to a hospital in as little time as possible, since they are in urgent need of medical treatment. Ambulances move around on an underlying road network graph on which we can perform shortest path queries to optimize routes and find the closest hospitals or ambulances to any point on the graph.
One major goal is to maximize the number of saved patients. A patient counts as saved if they reach a hospital alive. Other goals include the minimization of response times, i.e. the time an ambulance needs to reach a patient, and the minimization of overall distance the ambulances need to travel. We want to find a good online algorithm to solve this problem. For a formal problem definition see section 2.1

1.3 Related Work

Research in the field of route planning, vehicle fleet routing and more specific ambulance allocation and management has a rich history that goes back to the 1950s.

The basis of most state-of-the art route planning algorithms is Dijkstra's algorithm, which was introduced in 1959 [Dij59]. The algorithm gives a path of minimal cost between two nodes, given a graph with positive edge costs. Dijkstra's algorithm was later improved by a variety of techniques such as arc flags [Lau04] and contraction hierarchies [GSSD08]. Both of those improvements add precomputation steps to decrease the size of the search space.

When it comes to managing fleets of vehicles, one common problem is the *dial a ride problem* (DARP). In the DARP a number of users request pick-up and drop-off at two points on a graph to a vehicle or fleet of vehicles. The goal is to find good schedules for the vehicles to satisfy as many users as possible [CL03]. In a DARP setting there is not as much urgency as in the EMS setting: It is sometimes allowed to have multiple passengers in one car and it is acceptable to drive detours if this can maximize the total number of passengers. Furthermore the destinations in DARP are more diverse, whereas there is only a small set of hospitals in the EMS setting.

Another related subject is the coordination of a fleet of delivery vehicles, which transport goods from one point to another. In [AGP12] Azi et.al. describe a dynamic routing system for a fleet of delivery vehicles which makes decisions about incorporating new requests into the schedules. In [FBG13] Ferrucci et.al. present a management system for urgent delivery of goods. They derive a stochastic model from past requests and try to anticipate future requests with vehicle redeployment. In goods delivery past requests hold some information about future requests, for example if one factory wants to send something urgently, there is a chance that

it might want to do it again in the future. This does not apply to an EMS system where a request from one location holds little to no information about future requests from the same venue.

Other research has been made working with EMS settings. One field of interest is actual routing of ambulances on a road network. Panahl et. al. present an approach on dynamic ambulance routing in a city which takes real time traffic congestion data into account to adjust the routes and minimize the response times [PD09]. This approach focuses on the optimization of single ambulances instead of fleet management. Other work deals with scheduling of secondary ambulances, i.e. less urgent inter-hospital transfer of patients. In [CHS+13] a system to plan schedules for ambulance planes according to an existing set of request is presented by Carnes et.al. Given a set of patients with destinations and medical needs, they calculate optimal journeys for a minimal number of planes. In [Par09], Parragh also deals with optimizing of secondary ambulance transports and allows multiple patients to be transported in the same ambulance to save costs. These secondary ambulance settings lack the emphasis on urgency we have in our model and focus more on economic aspects.

Another often studied problem is optimization of ambulance base location within a certain area. Here, the task is not to manage an ambulance fleet, but to distribute bases on the map and allocate ambulances to them to be optimally prepared. Knight et. al. derive ambulance base location from a detailed model, taking different classes of patients into account [KHS12]. These different patient classes model different medical conditions and corresponding needs. For example a patient with a broken arm can be treated in a smaller hospital, while a patient with a failing heart probably needs to be brought to a more specialized medical facility.
In [Hal72] Hall uses statistical analysis based on real world data from the Emergency Services of Detroit. Not only EMS data, but also police emergency occurrences are incorporated into the analysis and also different operation modes, such as dual function ambulance police cars, are looked at.

Berlin et.al. divide the allocation problem into two sub-problems: Ambulance base location and allocation of ambulances to those bases [BL74]. A set covering algorithm is used to find a good distribution of ambulance bases which are then filled with ambulances according to the results of a simulation based calculation. However, no dynamic management is incorporated into the simulation.

The task of managing an ambulance fleet in operation has also been studied. Gong et. al. examine disaster relief operations [GB07]. In their setting they assume a disaster of some sort has struck a certain area. They divide the area into clusters and allocate and assign ambulances to those clusters. This allocation is dynamically adjusted as the situation changes.
A more general EMS setting is analyzed by Zhu et. al. in [ZM93]. They also make heavy use of real world data from the city of Shanghai, based upon which they develop a mathematical load balancing model. With this model they redeploy

ambulances to different stations as the demand changes. Their approach does not directly manage ambulances, but merely observes request frequencies in the system and makes recommendations on how many ambulances should be stationed at this time in each station. It also focuses solely on the real time data and environment in Shanghai and is not tested or benchmarked against other environments.

A combination of fleet allocation and dynamic management is presented by Yue et.al. [YMK12]. This approach uses a simulation based strategy and historical data to manage the fleet in a large city. The results show that it works well for the region they have the data for, but no attempt to benchmark the system in a generalized setting is made.

2 Analysis

In this chapter we first formally define the set of problems which we deal with in this thesis. We then demonstrate the relation between them and scheduling problems and also give an NP-hardness proof for one of the ambulance allocation problems. Finally we introduce the concept of duty zones, which we then use to find upper bounds for the problems.

2.1 Problem Definition

The ambulance fleet allocation problem is defined as follows:

Definition 1. *We are given a street graph $G(E, V)$ with a set of edges E, a set of vertices V and a cost function $c : E \to \mathbb{R}^+$ assigning travel times to the edges, a set of hospitals $H \subset V$ ($|H| = h$) and a fleet of k ambulances. There is a stream of requests $r = (o, a, d)$, where $o \in V$ is the origin of the request (i.e. the location of the patient), $a \in \mathbb{R}^+$ is the starting time at which the request becomes visible and $d \in \mathbb{R}^+$ is the deadline at which point the patient has to be at one of the hospitals in order to be saved. Given this definition, a patient is saved if an ambulance at position $w \in V$ is assigned to the patient at time $t \in \mathbb{R}^+$, with $t \geq a$ and $t + c(w, o) + c(o, H') \leq d$ with $H' \in H$ being the hospital the patient is brought to. It is also possible to move idle ambulances freely around any time without sending them to patients. The goal is to manage and dispatch the ambulances in such a way, that the number of saved patients is as close to the number of total requests as possible.*

We call this general form of the problem the *Many Hospitals Many Ambulances* or *MHMA* problem. For formal analysis we look at the more restricted *One Hospital Many Ambulances* (*OHMA*) and *One Hospital One Ambulance* (*OHOA*) problems. There is also an offline version of the problem, where all requests are known in advance and can be taken into account while generating a schedule. We use this offline version when we try to come up with upper bounds.

2.2 NP-Hardness Proof

In this section we look at the *OHOA* problem where $k = h = 1$, which means there is only a single ambulance and a single hospital. We furthermore define *request intervals* for each request $r = (o, a, d)$ as $[a, d], l$ with l being the travel time of a round trip from the hospital to the patient and back ($l = c(H, o) + c(o, H)$). We do not allow requests with a request interval where $l > d - a$ as those requests can not be served. A request can be served, i.e., the patient can be saved, if an ambulance is assigned to the request at time $t : a < t < d - l$.

With this definition two requests (o, a, d) and (o', a', d') can be served in OHOA if $\exists t : t \in [a, d - l], \exists t' : t' \in [a', d' - l'] : [t, t + l] \cap [t', t' + l'] = \emptyset$, i.e. if there exist two intervals of length l and l' respectively within the request intervals that do not overlap (see Figure 2.1). This interpretation of the problem resembles a scheduling problem. Garey et.al. present a *dynamic scheduling problem* and prove that it is NP-hard [GJ79]:

Definition 2. *Given n jobs, each with a release time, a deadline and a processing time, decide whether all jobs can be scheduled on a single machine respecting the given time windows.*

We can reduce this problem to our *OHOA* problem: For every instance of the dynamic scheduling problem we define a star graph with the central node being the hospital. For each job j we add a node and connect it to the center node using and edge with travel time $\frac{1}{2}\cdot$ processing time of j. That way a round trip to the patient and back takes the same time as the processing time of j. Then we add a

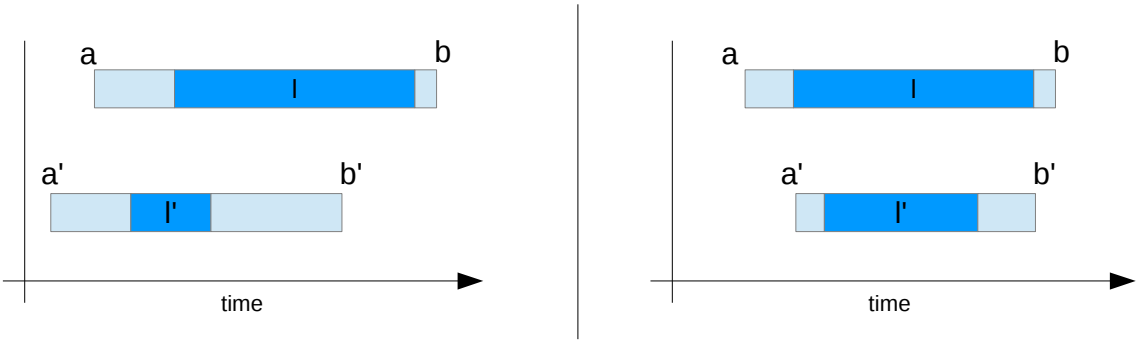

Figure 2.1: Visual representation of request intervals: To successfully handle a request a start time t_s can be selected, such that $a <= t_s <= d - l$, which corresponds to moving around the dark blue bar inside the light blue bar. Two requests can be handled if the dark bars can be arranged such that they do not overlap in the time axis. The left image shows an example of two requests which can be handled by one ambulance and the right image two requests which can not both be served by one ambulance.

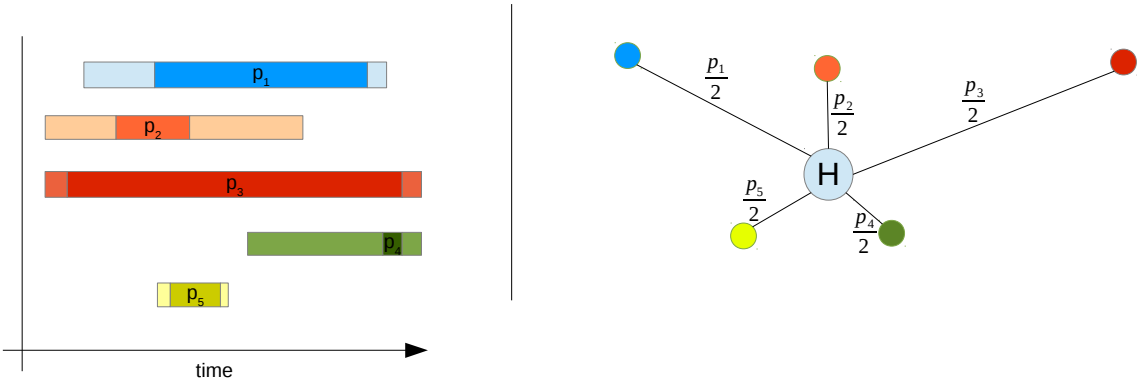

Figure 2.2: The left image shows an instance of the dynamic scheduling problem as described by Garey et.al. in [GJ79]. The light bars indicate the release time and deadline of each job and the width of the dark bars corresponds to the processing time. The right image shows the street graph of the reduction of the same problem to an *OHOA* problem. The travel times of the arcs are half the processing time of the corresponding job.

request from that node with a = release time of j and d = deadline of j. Clearly, every solution for the constructed ambulance allocation problem corresponds to a valid solution for the dynamic scheduling problem and the translation can be done in polynomial time. Figure 2.2 shows a graphical illustration of the problem reduction.

Since we have shown that the *OHOA* problem can be reduced to the dynamic scheduling problem in polynomial time, we can conclude that *OHOA* is NP-hard. Furthermore, since *OHOA* is a simplification of the other ambulance management problems, especially *MHMA*, we can deduce that they are NP-hard too.

2.3 Upper Bounds

When evaluating online algorithms it is helpful to have good upper bounds on the performance of the algorithms. In our case it would, for example, be interesting to know, if for a given set of requests, all patients can be saved or, if not, what the maximum number of patients is that can be saved. It is a common approach to compare online algorithms with their offline counterparts and then reason about their *competitiveness*.

For the simplified case where there is no flexibility in requests, we can easily find upper bounds. In this case for each request $r = (o, a, d)$ the travel time from the hospital to o is exactly $\frac{d-a}{2}$ and thus the request has to be served immediately. This eliminates the dynamic aspect and it remains to find a maximum size independent set among the intervals. This can be done with a greedy algorithm in $\mathcal{O}(n \cdot log\ n)$ as described in [SH05]. However, the fact that *OHOA* is NP-hard suggests that it

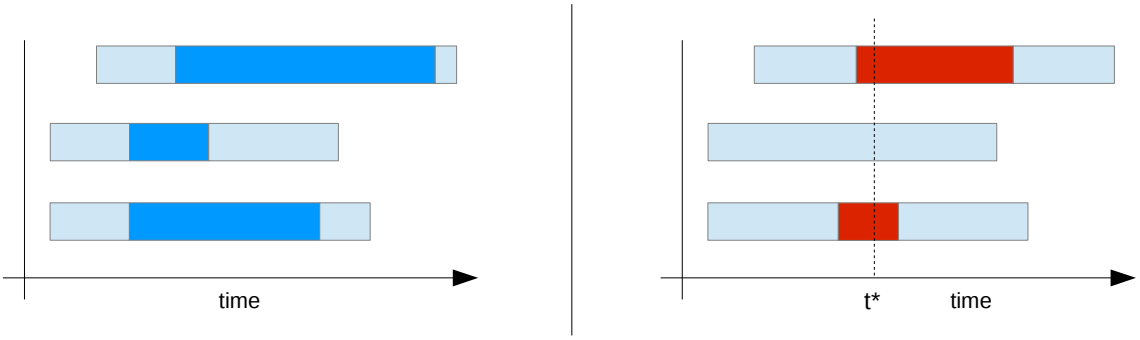

Figure 2.3: An illustration of duty zones: From regular request intervals (left) we compute the duty zones (red), i.e. the overlapping area when we move the dark bars all the way to the left and all the way to the right. Note that some requests might not have duty zones. Since the topmost and bottommost requests have overlapping duty zones, they can not be served by one ambulance. At time t^* the ambulance would have to be busy with both requests, which is impossible.

is much more difficult to find the optimal offline solution. So we instead try to find good upper bounds which can be computed more easily.

Upper bound using Duty Zones

Since it is hard to compute an optimal solution for a given *OHOA* problem, we try to look at the requests and prune some of them. For this we define the *duty zone* of a request with the request interval $[a, d], l$ as the time interval $[a + l, d - l]$, i.e. the time interval in which an ambulance has to be occupied by this request if it is to be served at all. In other words, if an ambulance is dispatched to the request right away $t = a$, if it is dispatched in the last possible moment $t = d - l$, and if it is dispatched any time in between it will be busy during the duty zone. See Figure 2.3 for a graphical illustration of duty zones.

We can use these duty zones to identify impossible combinations of requests. If two requests r and r' have overlapping duty zones, i.e. $\exists t : t \in dz(r) \wedge t \in dz(r')$, it is impossible to serve both requests with only one ambulance, because at time t the ambulance would have to be occupied with both requests. It is of course possible for the duty zone to be empty in cases where the earliest and latests possible processing times do not overlap.

Given a number of requests with their respective duty zones, we can compute an upper bound for our ambulance allocation problems. We start with the upper bound $U = |R|$, the number of requests, and continue to decrease it as we find conflicts in the request stream and assume that we have k ambulances at our disposal. We look through the requests until we find the first t where more than k duty zones overlap. It is clear that at this point we can not be serving more than k requests and thus

we must decrease U by the number of conflicting duty zones minus k. This way we sweep through all of the requests and decrease the upper bound every time we encounter such conflicts. We disallow one request being in more than one of those conflict sets and thus remove all requests from one conflict set after the set has been processed.

A different approach to an upper bound is taking the set of duty zones as an instance of regular interval scheduling and computing an optimal solution for it. This gives us an upper bound for the number of jobs a single ambulance could serve while only respecting requests with duty zones. We multiply this value with k and add the number of requests without a duty zone to get the alternative upper bound U'. We then obtain the final upper bound $U^* = min(U, U')$.

This approach of finding an upper bound can also be applied to the *MHMA* problem, with many ambulances and many hospitals. For this we simply use the travel time to the closest hospital when calculating the interval length for each requests.

3 Model

In this chapter we describe the model we designed for the problem at hand. An EMS system and its ambulances operate on a road network and so we use a road network graph as the basis of the model. Then we introduce and describe all the objects we model on the graph and finally outline the capabilities of our EMS operator agents and their abilities to interact with the modelled objects.

3.1 Road Network

The road network on which the ambulances operate is modelled as an undirected graph $G = (E, V)$ with edges and vertices.

The edges E in the graph represent the roads or parts of them. We use the word "road" to refer to any kind of surface capable of being travelled on by an ambulance. We do not differentiate between different types of roads, which means a vehicle moves with the same speed on all roads. Furthermore we assume vehicles can drive on all roads in both directions. Each road in the model has a length in meters.

The vertices or nodes V represent junctions or turning points on roads. Each node has a latitude and longitude representing its position on the earth's surface. We use these coordinates to calculate the length of the edges by calculating the distance between the two nodes which are connected by the edge. The position of all objects in the simulation is given by node id. This discretization of ambulances and the implied fact that an ambulance can not be positioned part-way on an edge could lead to problems on long edges, as an ambulance's position when travelling on an edge is the source vertex until the entire length of the edge has been travelled and then changes to the destination vertex. However the data we use for our graphs is very fine grained and so edges are usually very short (see subsection 6.2.2), which minimizes this effect.

We make sure that all road network data that we use as input represents one connected graph. To assume a connected road network, i.e., any node can be reached from any other node, is reasonable, since ambulances could also not reach a sealed off area in reality.

3.2 Objects on the Graph

The road network serves as an underlying reference system for all objects and entities that are part of an EMS system. We modelled the following objects and entities:

- **Ambulances:** Ambulances are the resource managed by the different strategies. They have a capacity of one, which means, they can carry only one patient at a time. Each ambulance has a constant speed and a home base at which it will start. Ambulances are always in one of the following states.
  - Free.
  - Free and at its base.
  - En route to a patient.
  - At a patient.
  - En route to a hospital with a patient.
  - At a hospital; dropping off a patient.
- **Ambulance Bases** $B \subseteq V$: The buildings where ambulances are stationed. They do not have any additional function in this model.
- **Hospitals** $H \subseteq V$: Hospitals or comparable medical institutions, which provide medical care to injured and sick people. The only way to save a patient is to transport them to one of the hospitals. We do not differentiate between different classes of hospitals and assume any medical condition can be treated by any hospital. This assumption is a simplification we make not to overcomplicate the model, but taking different hospital classes into account could provide further interesting results. See subsection 7.1.3 for more details.
- **Patients:** The main goal of any EMS system is to transport patients to hospitals. The simulation will generate EMS requests, each of which will spawn a patient at a position represented by a node and a time to live (TTL). The patient must be transported to a hospital before the TTL runs out.

3.3 Request Generation

From the EMS operators point of view the stream of incoming requests appears to be very random, and unpredictable as little to nothing can be known about the amount and nature of requests in the future. To simulate this we use a random request generator which generates a stream of requests which we then use as the basis of our simulation. The requests have two random components: time and location.

To simulate the time of a request, we use the parameter expected time between requests (e_{tbr}) to be able to simulate different workloads. Given this parameter

request number	time of request	time since last request
0	01:43	01:43
1	05:39	03:56
2	16:30	10:51
3	20:13	03:43
4	28:42	08:29
5	34:58	06:16
6	36:29	01:31
7	36:54	00:25
8	43:11	06:17
9	50:08	06:57

Table 3.1: An example of a request stream with ten requests. The e_{tbr} is set to 300 seconds, i.e. five minutes. This example demonstrates that the used generation method can produce very short intervals such as between requests six and seven, but also rather long intervals such as between requests one and two. In this case the average time between consecutive requests is exactly five minutes.

we generate n requests with the first requests occurring at $t_0 = X_0$ where X_0 is a random variable whose value is subject to exponential distribution with $\lambda = \frac{1}{e_{tbr}}$. All other requests n occur at time $t_n = t_{n-1} + X_n$ where X_n are again random variables following the same distribution as X_0.

With those generation rules we end up with a series of requests where the average time between requests n and $n + 1$ goes to e_{tbr} as n goes to infinity.

The second random component of requests is the location or origin on the graph. Since medical emergencies can occur anywhere in form of a road accident on a motorway or a stroke in a residential area in a city, we treat all nodes of the graph equally and select the origin of each request from the set V of all nodes using normal distribution.

3.4 Agents

In the real world, ambulances are coordinated by operators or a team of operators in a dispatch center. In our model, EMS operators are represented by agents. In our simulation, agent objects define a set of methods which they can use to subscribe to specific events (see section 4.1). They also have a persistent state so that they can keep track of past events. In particular all agents have a *request queue*, a priority queue in which they can store requests which they are currently unable to respond to.

Once a subscribable event comes up, the respective method will be invoked. The agent is then allowed to inspect the observable state of the world, e.g., the position

of all ambulances, and must then decide if and how to react to the current event. To make this decision, an agent has full access to the road network information and can run arbitrary route planning or other algorithms to gain information, just as a real operator could do with the help of a computer.

Agents will try to respond to as many requests as possible in a *first-in-first-out* fashion, which means if requests r is registered before request r', there will be an ambulance assigned to r before one is assigned to r'.

Orders

As a reaction to all types of subscribable events, agents can issue a list of orders to ambulances. An order consists of the ID of the ambulance and the ID of the target node. This list of orders is then passed to the simulator, where it is validated to ensure that no orders are given to, for example, an occupied ambulance which cannot take new orders. After that, the ambulances are assigned to their new targets. For the sake of simplicity we chose not to model the short delay that the propagation of orders would create in the real world. Orders are assigned immediately and the ambulances need no time to react to them. Ambulances automatically chose the shortest path to their new target, therefore the path is not part of the order.

Ambulances do not keep any state about orders. They can only process one order at a time and immediately forget about an order once a new one is given. An order queue for each ambulance could however be emulated by the agent.

4 Strategies

In the previous chapter we described the model our simulations are based on and the general way the agents work and interact with the world. In this chapter we present the management strategies which can be employed by those agents in detail. We first give some more information about the task of the agents, followed by further defining the methods of interaction with the world. Finally, we present the three main strategies we developed to manage an ambulance fleet in detail.

The main goal to all our strategies is to help as many patients as possible while staying economically viable. Three distinct sub-problems can be described which the agents need to address in order to achieve these goals:

Planning ahead

In a situation where no new EMS requests are created, for example, at the beginning of the simulation the agents can try to prepare themselves and the ambulance fleet in a way that might make it more easy for them to deal with future requests, for instance, redistributing the fleet in a certain way.

Initial response to requests

Once a new emergency call comes in the agent must decide, based on the location of the patient, which of the available free vehicles it wants to order to respond to the emergency. If there is no free ambulance available, the agent needs to save the request in a queue and try to answer it again once an ambulance becomes available.

Reassignment

During the time an ambulance needs to reach a patient, it is still a free ambulance and can potentially be diverted to a different patient. As more requests come in, it can be very beneficial to reassign the ambulance and send another one to the patient it was on its way to previously. Reassignment is not part of the core strategies, but can be added to any of the three. It is described in detail in section 4.5.

In our simulation we assume that all decisions are made instantly, i.e., the simulation is halted while the simulator waits for the decisions by the agent. However, we make sure that all decision making processes take negligible amounts of time.

4.1 Reaction Slots

In our simulation we define a set of situations, where actions can be taken by the agent. This set covers all situations in which the state of the simulation has changed in a way that could possibly influence the actions taken by the agent, e.g., a new EMS request or an ambulance that is available again after delivering a patient to a hospital. An ambulance that has travelled another kilometer along its path while nothing else has happened can not possibly change the decisions made by a deterministic agent in a deterministic simulation and so this is not a situation where agents can make decisions. Other non reactable situations include events that do change the simulation but are assumed not observable by the agent, such as the death of a patient before an ambulance has reached it.

To make it possible for the agent to react to those situations we define a *reaction slot* for each of them. A reaction slot is a method for a specific situation which all agents must implement. This method will be invoked by the simulator whenever the respective situation occurs. The return value of all reaction slot methods is a (possibly empty) list of orders that the agent gives to ambulances. If an agent does not want to react to a certain class of event the respective method just needs to return an empty list.

The list of reaction slots contains the following situations:

- **Simulation Begins:** This is a unique event that only occurs once at the beginning of each simulations, to enable agents that want to do repositioning of ambulances to do so.
- **New Request:** A new EMS request occurs. The agent learns the origin and can dispatch an ambulance.
- **Patient in Ambulance:** The patient is aboard the ambulance and it can be sent to a hospital.
- **Ambulance Free:** An ambulance has delivered a patient to a hospital and is now available again

4.2 Greedy

The most straightforward approach to the ambulance management problem is a greedy or lazy algorithm. A greedy approach is the base of what is currently used by most real world EMS systems. Ambulances are stationed in hospitals, fire departments or separate ambulance depots where supplies are kept and a repair and maintenance infrastructure is present. This makes it a reasonable approach for an operator to leave the ambulances in these depots when not needed and send them back once an order has been processed.

This widely used approach has been sudied before and found to perform well in common EMS settings [YMK12].

The greedy approach does not employ any kind of precomputations and does not do any redistribution of the initial fleet. Every ambulance has a fix assignment to a home base. It can only be on its way to a patient, hospital or back to its base at any time. There is no other event that will cause Greedy to move an ambulance.

4.2.1 Event Handlers

Greedy subscribes to the following events and reacts as described:

- **New Request:** A list of all available ambulances and their positions is obtained. This list includes ambulances idle in their bases and those on their way to a base. Using multi-source-Dijkstra with the ambulance's positions as sources and the request origin as target, the closest ambulance is determined and assigned to the request. No other orders are issued.
- **Patient in Ambulance:** Once an ambulance reports as ready to leave with a patient, Greedy will run a multi-target-Dijkstra with the ambulances position as source and all hospitals as targets to determine the closest hospital and order the ambulance to go there.
- **Ambulance Free:** An ambulance that reports as free is immediately ordered back to its home base.

4.2.2 Characteristics

The Greedy approach does not do any kind of redistribution of the initial ambulance fleet, therefore we can assume that it is very sensitive to the initial distribution of the ambulance fleet. The time it takes for an ambulance to reach a patient is dependent on how close the nearest ambulance base is to the patient, which is something the algorithm has no control over. We could confirm this in our experimental results (see subsection 6.4.2).

Economically, Greedy can be expected to perform rather well, since it does no moving around of ambulances other than to respond to requests and always uses the shortest possible routes.

It is also the least computationally complex approach, as no precomputation has to be done. While responding to requests, Greedy limits itself to two multi-source/target Dijkstra runs: one to find the closest ambulance and one to find the closest hospital. All other strategies do this too in addition to their own precomputation and redistribution, leaving Greedy as the approach with the lowest expected runtime.

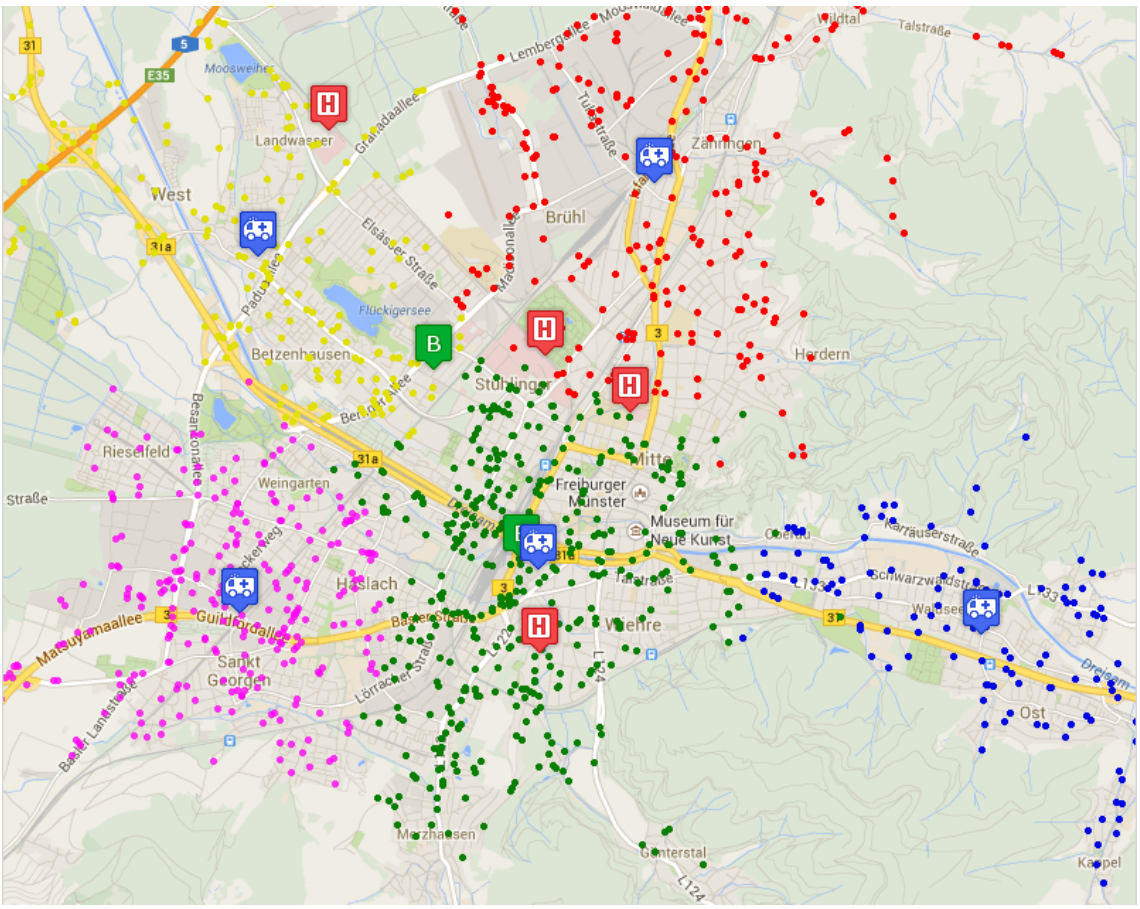

Figure 4.1: The initial distribution of K-Medoid with 5 Ambulances (blue). Each group of colored dots represents one cluster. The ambulances are positioned at the medoids of these clusters.

4.3 K-Medoid

The most significant drawback of Greedy is its dependence on the initial distribution of ambulances. To overcome this we introduce K-Medoid. This strategy tries to optimize the distribution of free ambulances at any time and therefore minimize the response time to the next requests. This way we hope we can both minimize the average response time and rescue more patients overall.

To accomplish this, we use what we call the K-Medoid distribution, where ambulances are distributed according to the results of the K-Medoid clustering algorithm.

4.3.1 Modified K-Medoid Algorithm

The K-Medoid algorithm is a relative of the K-Means clustering algorithm, which is a widely used algorithm in data analysis. K-Means is based on work by Lloyd

who first introduced the basic principles in 1982 [Llo82]. Given a number of n points in space, a distance metric and a set of k initial means, K-Means computes a k-clustering seeking to minimize the average distance of a point to the mean of its cluster. The result is not guaranteed to be globally optimal, but can be a local optimum arbitrarily worse than the global optimum. While in practice the runtime of the algorithm is often acceptable, it has been shown to be theoretically superpolynomial [AV06].

Algorithm 1: The K-Means algorithm

Data: A set of points V, a distance metric and k initial means
Result: An assignment $V \rightarrow 1, ..., k$

```
1 while Clusters have changed do
      // Assignment step
2     for v ∈ V do
3         Assign v to the cluster with the closest mean;
4     end
      // Update step
5     for i ∈ 1, ..., k do
6         Calculate mean m of cluster i;
7     end
8 end
```

In K-Medoids, the clusters are formed around medoids rather than means, i.e., the cluster centers have to be part of the set of data points. K-Medoids is also know as *PAM* (Partitioning around Medoids) and was introduced by Kaufman et.al. [KR90]. The algorithm is similar to K-Means, as it also has an update and assignment step in each iteration (see algorithm 3). However, the update step is more costly in K-Medoids since for every cluster, every member is tested as a new medoid. There are techniques to improve the speed of the algorithm, for example, the *CLARA* method which only considers a randomly selected subset of each cluster as medoid candidates [HKP06]. This method sacrifices optimality for speed, as the found clustering can never be optimal if any of the optimal medoids are not part of the randomly selected subsets.

We use a different modification to K-Medoids to improve its running time. Instead of evaluating every data point as new medoid during the update step, we calculate the mean of the cluster and define the cluster member with the lowest distance to the mean as the new medoid (see algorithm 2). This modification requires there to be a meaningful distance measure from data points to arbitrary values instead of or in addition to a distance matrix. We also require mean calculation to be possible and meaningful. In the case of geocoordinates both these requirements are satisfied as we can use Euclidean distance as a distance measure and define the mean of a set of coordinates as the mean of the latitude values and the mean of the longitude values.

Algorithm 2: The K-Medoids algorithm

Data: A set of points V, a distance metric or matrix and k initial medoids $m_1, ...m_k \in V$

Result: An assignment $V \to 1, ..., k$

```
while Clusters have changed do
    // Assignment step
    for v ∈ V do
        Assign v to the cluster C_i with the closest medoid m_i;
    end
    // Update step
    for i ∈ 1, ..., k do
        for m' ∈ C_i do
            Calculate the cost c(m') of the cluster with m' as medoid;
            if c(m') > c(m_i) then
                m_i = m' // Define m_i as new medoid of C_i
            end
        end
    end
end
```

Algorithm 3: Modified version of K-Medoids

Data: A set of vertexes V with a disctance function $dist : V \times V \to \mathbb{R}_+$ and k initial medoids

Result: An assignment $V \to 1, ..., k$

```
while Cluster have changed do
    for v ∈ V do
        Assign v to the cluster with the closest medoid
    end
    for i ∈ 1, ..., k do
        Calculate mean m of cluster i;
        Define the vertex m_new with m_new = argmax_v dist(v, m)
    end
end
Calculate the closest medoid of each vertex on the actual road network;
Reassign each vertex to the closest medoid;
Recalculate medoids;
```

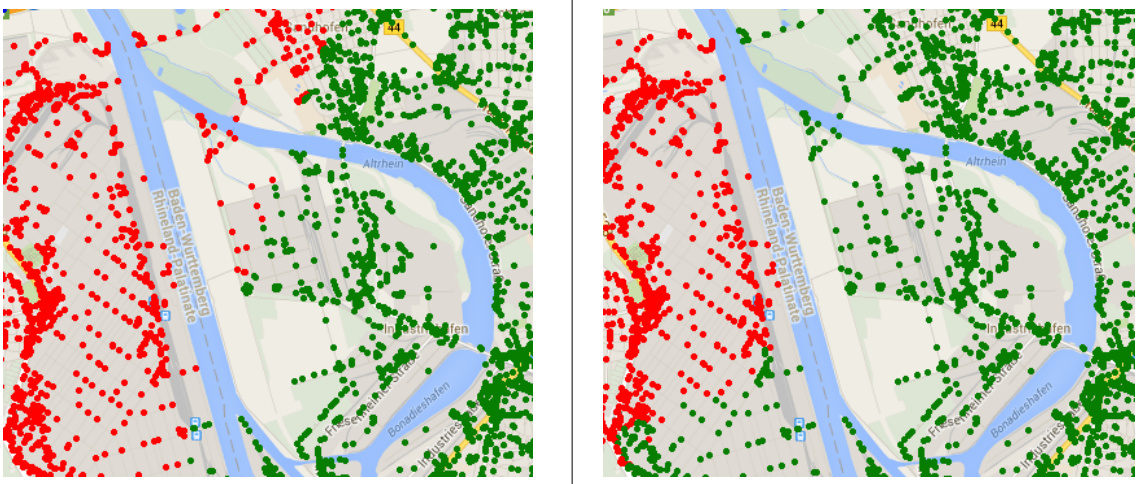

Figure 4.2: If run without the refinement step (left), some nodes on the right hand side of the river are added to the red cluster, although, because of the lack of bridges in this area, they are closer to the green medoid on the street graph. With the refinement step (right) they are added to the green cluster.

Refinement Heuristic

To decrease the complexity of the algorithm, we use a straight line heuristic during the iterative computation of the clusters. That is, we assume there is a direct straight line road between two vertices, which allows us to use the Euclidean distance as distance measure. Because our geographical data is so fine grained, and especially in cities street graphs are highly connected, this is usually not simplifying the problem too much. However, it can introduce unfortunate assignments, for example, at rivers with no crossing in the area (see Figure 4.2).

To compensate for this, we do a refinement step after the clusters have settled, where we calculate the closest medoid of each vertex on the actual road network using a multi-source complete Dijkstra run. We then reassign the vertices accordingly and do one last recalculation of the medoids.

Result

The result of this algorithm gives an approximation of the k medoids in a graph that minimizes the average distance of a vertex to the closest medoid. In the EMS setting this means that if we have x ambulances and place them on the calculated medoids, we can hope to minimize the time an ambulance needs to respond to the next request. To achieve this we need to precompute the medoids for each $k \in 1, ..., x$ in order to be able to work with any number of free ambulances. Since the resulting

medoids only depend on the underlying graph and not on the position of ambulances, hospitals or other objects, they can be cached on a per graph basis.

4.3.2 Event Handlers

To minimize the response time of ambulances, the K-Medoid strategy tries to place the free ambulances on the k medoids of the graph, where k is the number of free ambulances at the current time. Response handling is similar to Greedy.

- **Simulation Begins:** Once the simulation begins and before any requests have occurred, all available k ambulances are spread across the graph, each being ordered to one of the medoids, where they will remain until a request comes in.
- **New Request:** A list of all available n ambulances and their positions is obtained, including ambulances idle at medoids or en route to one. Using multi-source-Dijkstra with the ambulance's positions as sources and the request origin as target, the closest ambulance is determined and assigned to the request. Since the number of free ambulances has now changed to $n - 1$, the remaining free ambulances are repositioned to the $n - 1$ medoids.
- **Patient in Ambulance:** Just as in Greedy, ambulances with patients are sent to the nearest hospital using a multi-target-Dijkstra.
- **Ambulance Free:** Once an ambulance reports as free again, the number of free ambulances is increased and the ambulances are repositioned.

4.3.3 Characteristics

The main goal of this approach is to minimize the time it needs to respond to requests while not depending on the placement of ambulance bases on the map. Since nothing is known about the location of future patients, we compromise and try to cover all nodes the best we can. The heavy redeployment involved in this makes it rather uneconomic, as the total distance travelled by ambulances can be expected to be significantly higher than in Greedy.

The approach is also not very practical as, in reality, ambulances do need to return to their bases in order to be re-equipped and cleaned. Since medoids can be any node in the graph the can also be on highways and other points where it is in practice hard or impossible to station an ambulance.

Computationally, this approach is the most complex of the presented ones as it involves rather costly precomputation (although it can be cached) and most event handling involves redeployment of ambulances.

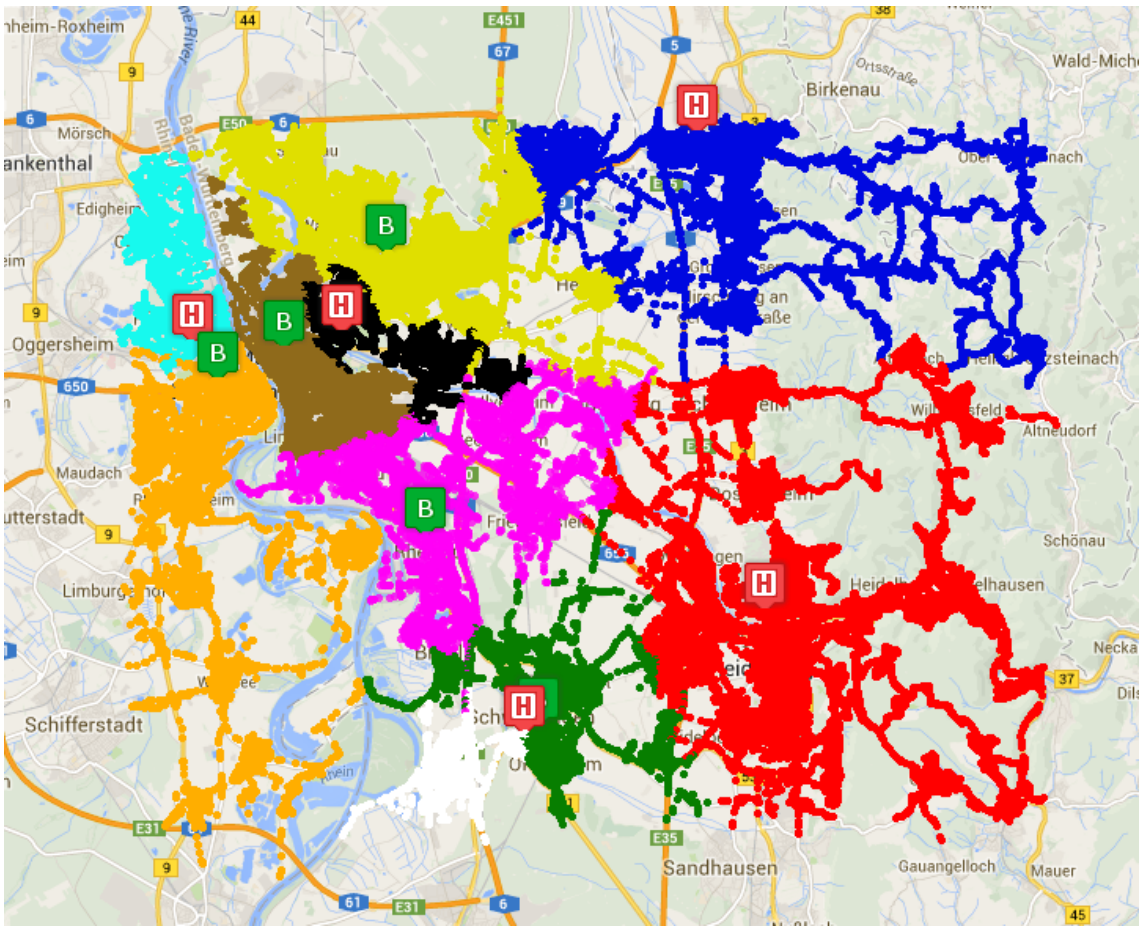

Figure 4.3: The voronoi cells of a street graph. Each hospital and ambulance base is the seed of one cell.

4.4 Voronoi

To overcome the practicability problems of K-Medoid and make the approach more economical, we introduce a third strategy, which we call Voronoi, named after Voronoi diagrams which represent the underlying concept [Aur91].

In general, a Voronoi diagram is a way of partitioning a space into n cells, given n fixed points in space that will each become the center of a cell. Each point in space is assigned to the cell of the center which is closest to it according to a distance measure, for example, Euclidean distance.

Voronoi is a variant of K-Medoid with a restriction on the medoids: Only hospitals and ambulance bases are allowed as points where idle ambulances are stationed. In a precomputation step, we define each hospital or ambulance base as the center of a Voronoi cell. Each vertex is then assigned to the cell belonging to the closest hospital or base. Instead of using Euclidean distance when determining the closest center, we use the actual distance on the street graph. We do this by using the result of one

complete Dijkstra run with all hospitals and ambulance bases as seeds. This way we obtain for each node the closest seed and can add this node to the Voronoi cell of that hospital or ambulance base. In other words, each node belongs to the Voronoi cell of the center from which an ambulance would have the shortest distance to that node, if all centers had at least one free ambulance available. Figure 4.3 shows a visualization of Voronoi cells for a graph.

Each time the number of free ambulances changes, they are redistributed according to algorithm 4, where the number of ambulances stationed in a Voronoi cell is proportional to its size. Since it is possible that there are more cells than ambulances, the smallest cells might be assigned no ambulances.

Algorithm 4: Assignment of free ambulances to Voronoi cells

Data: A number of free ambulances, a graph g and a set C of Voronoi cells
Result: An assignment of the ambulances to the cells

```
for c ∈ C do
    x = ⌊numNodes(c) / numNodes(g)⌋ ;
    assign x ambulances to c;
end
while ambulances remain do
    assign one ambulance to the cluster with the fewest ambulances;
end
```

4.4.1 Event Handlers

The event handlers of Voronoi are similar to those in K-Medoid:

- **Simulation Begins:** As in K-Medoid, ambulances are repositioned directly after the beginning. The repositioning is done according to algorithm 4.
- **New Request:** A list of all available ambulances and their positions is obtained. This list includes ambulances idle in bases and those on their way to a base. With a multi-source-Dijkstra the closest ambulance to the request is determined and assigned to the request. The remaining free ambulances are repositioned according to algorithm 4.
- **Patient in Ambulance:** As in the other strategies, patients are sent to the nearest hospital using a multi-target-Dijkstra.
- **Ambulance Free:** After an ambulance becomes free all free ambulances are repositioned using algorithm 4.

4.4.2 Characteristics

Voronoi represents a compromise between Greedy and K-Medoid. It is more practical since ambulances are only idle in hospitals or ambulance bases, where they can be re-equipped and ready-rooms for the crews can be maintained.

Voronoi involves less driving around than K-Medoid, since if an ambulance stays in the same cell, it does not have to move at all. In K-Medoid every redistribution step means all free ambulances have to move, since the set of medoids is different for every k.

Like Greedy, Voronoi's performance is not completely independent from the setting, as positions of hospitals and ambulance bases are predefined. It is, however, more flexible, since hospitals can be used as bases and the initial distribution of ambulances can be changed.

Although it is computationally more complex than Greedy, the precomputation and computation during redeployment is less expensive than in K-Medoid.

4.5 Reassignment

In all of the previously described strategies, only ambulances which are not assigned to a request are considered available when a new request comes in. This means that once an ambulance has been assigned to a request, this assignment cannot be changed. This behaviour can lead to bad assignments for unfavourable sequences of requests (see Figure 4.4).

We also did not re-evaluate the situation once a new ambulance becomes free, although it might be much closer to a patient than the ambulance currently responding to them. This, too, can lead to unnecessarily bad response times.

To overcome these weaknesses, we designed a reassignment mechanism which can be incorporated in all previously described strategies in order to boost their performance.

4.5.1 Assignment Problem

To be able to effectively reassign ambulances to open requests, we need to find an assignment of ambulances to requests, such that the sum of the distances between each ambulance and its assigned request is minimized. This is a fundamental problem of combinatorial optimization and known as the *Assignment Problem*. The problem can be visualized with a complete bipartite graph where each vertex from the left side needs to be assigned to one vertex on the right, such that the cost of the used edges is minimized.

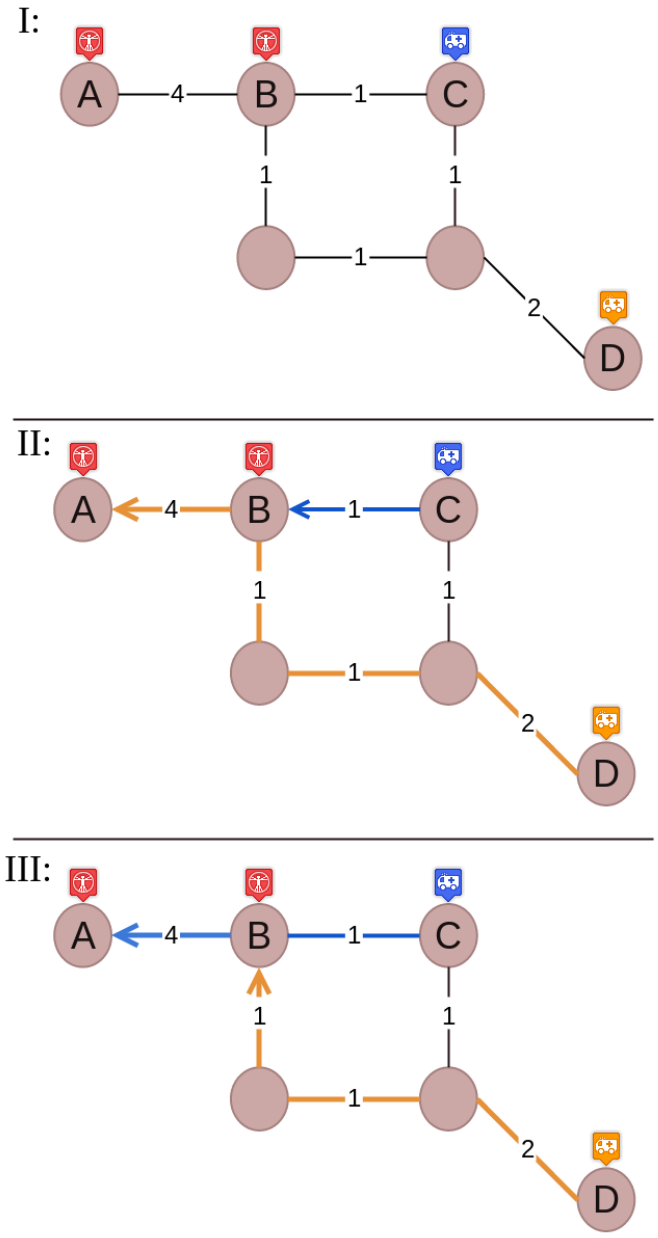

Figure 4.4: This situation demonstrates the benefits of reassignment. In the starting position **I**, two ambulances (blue and orange) are positioned at C and D, when two requests come in, *first* from B and then from A. Without reassignment (**II**) the blue ambulance is sent to B when the request comes in, as it is the closest, then, when the request from A comes in, only the orange ambulance remains free and is assigned to A. This results in a maximum response time of 8. When reassignment is enabled (**III**), the blue ambulance is reassigned to A and the orange ambulance is sent to B. This results in a maximum response time of 5.

The algorithm we use to solve this problem in polynomial time is known as the *Hungarian Algorithm*, *Kuhn-Munkres algorithm* or just *Munkres algorithm*. It was initially developed by Kuhn [Kuh55] and later studied and improved by Munkres [Mun57]. This algorithm allows us to find an optimal assignment of ambulances to patients in $\mathcal{O}(n^3)$ time, given a cost matrix. To obtain this cost matrix we use a complete mulit-target-Dijkstra for each ambulance, which gives us the distance to each patient. This way the final assignment will be the one which minimizes the sum of the distances which all ambulances need to travel.

Munkres algorithm only works with square matrices. If there are more free ambulances than patients, we add dummy patients with longer distances to all ambulances than the maximum distance in the matrix and disregard the assignments of ambulances to them. There will never be more patients than free ambulances, because requests are served in a first-in-first-out manner and we only take as many requests from the queue as there are ambulances available.

All our strategies did not differ in the way they responded to requests, but only in management of idle ambulances. That means that we can apply the Munkres reassignment strategy to all of them (see algorithm 5).

Algorithm 5: Ambulance reassignment

Data: All ambulances and the request queue

Result: An assignment of ambulances to requests with minimal overall travel time

1 Let A^* be the set of free ambulances and those currently en route to a patient;
2 Let R^* be the set of requests ambulances are responding to plus x more from the queue, such that $|A^*| = |R^*|$;
3 **for** $a \in A^*$ **do**
4 | Calculate the travel time to all $r \in R^*$;
5 **end**
6 Calculate the optimal assignment using Munkres' algorithm;

Munkres algorithm style assignment is also used in K-Medoid and Voronoi when free ambulances need to be assigned to medoids or bases/hospitals. Instead of patients the free ambulances are then assigned to the respective points on the map or hospitals/ambulance bases.

5 Implementation

In this chapter we focus on the implementation of our strategies, the simulator and the graphical user interface. We list the technologies we use and schematically describe the way the simulation works.

When selecting the development stack, the most important requirements were:

- **Performance:** To be able to easily run a large number of simulations with different scenarios, a highly performant system is required.
- **Flexibility/Ease of development:** We want to be able to easily adapt the implementation to new ideas during the development and to minimize the development time in general.
- **GUI capabilities:** The selected development stack needs to provide an easy way to create a graphical user interface.

With these requirements in mind we opted for Python as the main programming language, as it provides both the ability to rapidly develop and adapt code, and also the tools to quickly develop a GUI. At the time of writing, Python usually falls short in performance of compiled languages such as C++. It is however possible to implement computation intensive parts in C++ and use them within the Python code. There are different means of achieving that. We chose the Boost.Python[1] library, which is part of the well known Boost library collection for C++ [AGK03]. With C/C++ extensions it is possible to expose single functions or a whole object written in C or C++ to Python and use them in Python code, i.e., after the code is compiled the Python interpreter can directly access the binary code and run it like a pure C/C++ program would. Furthermore, in the C/C++ code the programmer can use and return Python's basic data structures like lists or dictionaries.

We first created a pure Python implementation and then gradually re-implemented those parts in C++, which a profiler showed to be the most time consuming during a simulation run. First we wrote Dijkstras algorithm and its variations, like multi-source and multi-target-Dijkstra, as a C++ extension and later ported the whole graph data structure to C++. With these optimizations we were able to improve the running time by about a factor of ten.

[1] `http://www.boost.org/doc/libs/1_55_0/libs/python/doc/`

5.1 Simulation

There are some simulation libraries for Python available. The most prominent among which is SimPy [Mat08], a feature rich framework for various types of simulation. We decided against using a framework, since the simulation we want to implement is rather simple and we want to avoid introducing computational overhead.

We designed the model in such a way that it is suitable for discrete event based simulation (DES) and we can save computational complexity compared to time discrete approaches. The simulator we designed consists of an event queue, which is a priority queue where events are ordered by time, and a main loop which iteratively handles the events until no more events are registered (see algorithm 6).

Algorithm 6: Main Simulation Loop

```
1  Initialize event queue;
2  Generate events and add to queue;
3  Initialize Objects;
4  while queue not empty do
5  |   e = next event;
6  |   update ambulance positions to e.time;
7  |   handle event specific logic;
8  |   inform agent and retrieve orders;
9  |   relay orders to ambulances;
10 end
```

We designed the simulator and underlying data structures in such a way that they can easily be parallelized. This means we can run multiple simulations simultaneously which operate on the same graph, but with different settings in a way that there is only one copy of the graph in memory. While Python in general has difficulties with traditional threading due to the global interpreter lock (GIL) which prohibits more than one thread to execute a section of code, similar behaviour can be achieved by using the multiprocessing module from the standard library which offers the same interface as threading, but internally works with multiple processes.

5.2 Dijkstra's Algorithm

When computing shortest paths on our road network, we rely on Dijkstra's algorithm. We implemented the algorithm as a C++ extension to our Python code. Although the unoptimized version of Dijkstra's algorithm is not the most effective choice for larger graphs, it proved sufficient for our simulations, since a distinct EMS system usually only operates within a smaller geographic region such as a city. It

is worth noting however that the available optimization techniques for Dijkstra's algorithm, such as contraction hierarchies [GSSD08], could be used to improve the performance of our simulations.

5.3 Caching

We use many costly calculations in our programs. To reduce the burden of those, we make heavy use of caching, both run-locally and globally. During a run we cache all Dijkstra calls. For example, during a simulation there might be multiple requests for the shortest path between two points. To avoid calculating it multiple times, we employ a key/value store which uses node ID pairs as keys and store the results of all Dijkstra runs in there. Furthermore, we have a persistent file level cache which caches the results of K-Medoid calculations. Here we must carefully chose the key for the cache. If we only use the graph name or file name, changes in the graph, or using two graphs with the same name, would lead to illegal cache hits. For this reason we generate a JSON representation of the graph, which consists of a list of all the nodes with their latitude and longitude values and all the arcs with to and from nodes. We then calculate the SHA1 hash of this string and use it as the cache key. That way, if the graph is altered in some way, the SHA-1 digest will change and the K-Medoids calculation will have to be repeated.

5.4 Graphical User Interface (GUI)

When we run many simulations in a batch for benchmarking, we run the simulator in headless mode without any user interaction. However, we also provide a GUI to debug and visualize the simulations. The GUI is also written in Python using Qt[2] as a GUI framework and PyQt[3] as a binding library, since Qt is written in C++. Qt is an open source library published under the GNU General Public License (GPL) or GNU Lesser General Public License (LGPL) for versions > 4.5 respectively.

The main reason to use Qt is the WebView object which is a complete Webkit rendering engine for web content, which can be embedded into the application. Webkit[4] is an open source rendering engine for HTML content. It is used by a number of web browsers including Safari, Midori and until recently Google Chrome[5].

To properly display the location of objects on a street graph, we require a framework for map applications. We opted to use Google Maps API[6], because it is easy to use

[2] `http://blog.qt.digia.com/`
[3] `http://www.riverbankcomputing.co.uk/software/pyqt/intro`
[4] `http://www.webkit.org/`
[5] In 2013 Google announced that they would fork Webkit and continue development under the name Blink
[6] `https://developers.google.com/maps/`

and the API is well documented. There is also a similar API for OpenStreetMap, which we use as data source for geographical data. Being a Web based tool, Google Maps API is a JavaScript library. Therefore we wrote the code for our map area in JavaScript and are able to call JavaScript functions from Python code via Qt Webkit.

The GUI Application consists of the following components (see Figure 5.1 and Figure 5.2):

- **Map View:** The Map View displays the position and routes of ambulances as well as all other relevant objects, like patients and hospitals. We also use the map area to visualize results of K-Medoid and Voronoi cells. The map is interactive, users can zoom and pan around and tool tips provide additional information such as the ambulance's IDs.
- **Text log:** The text log is used to output all relevant information about current events, such as the time and type of event. Any new orders to the ambulances are also displayed here.
- **Control Area:** The user interacts with the application via the control area. The simulation can be stepped through event by event. The parameters of the simulation can also be adjusted here.

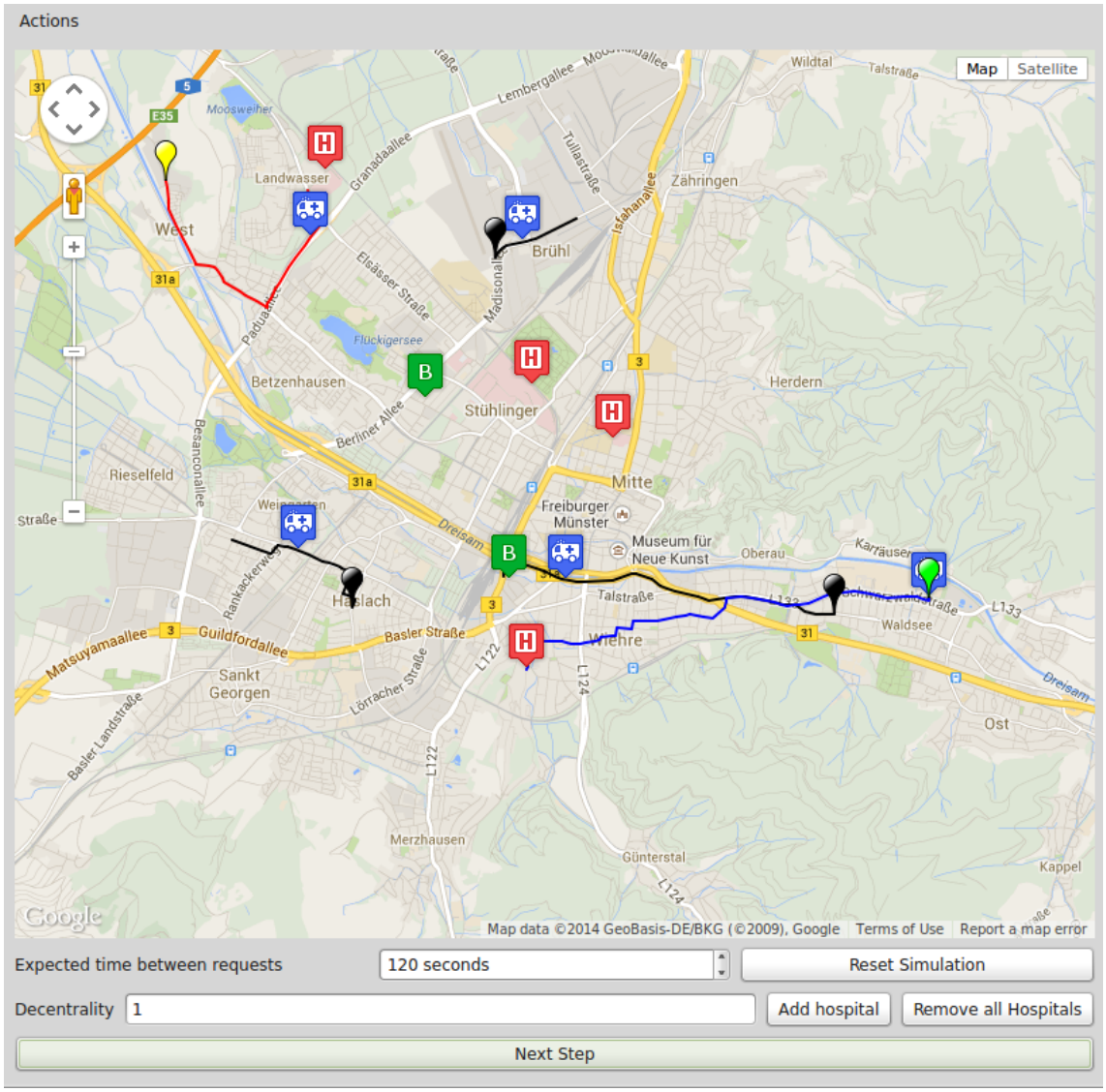

Figure 5.1: The map and control section of the GUI: The map shows the current position and routes of the ambulances. Red markers represent hospitals, green markers represent ambulance bases and blue markers represent ambulances. Ambulances on red paths are responding to a request by the patient represented by a yellow marker. Ambulances on a blue path are taking a patient to a hospital and ambulances on a black path are currently redeploying. The control area underneath the map is used to step trough the simulation, change parameters and restart the simulation.

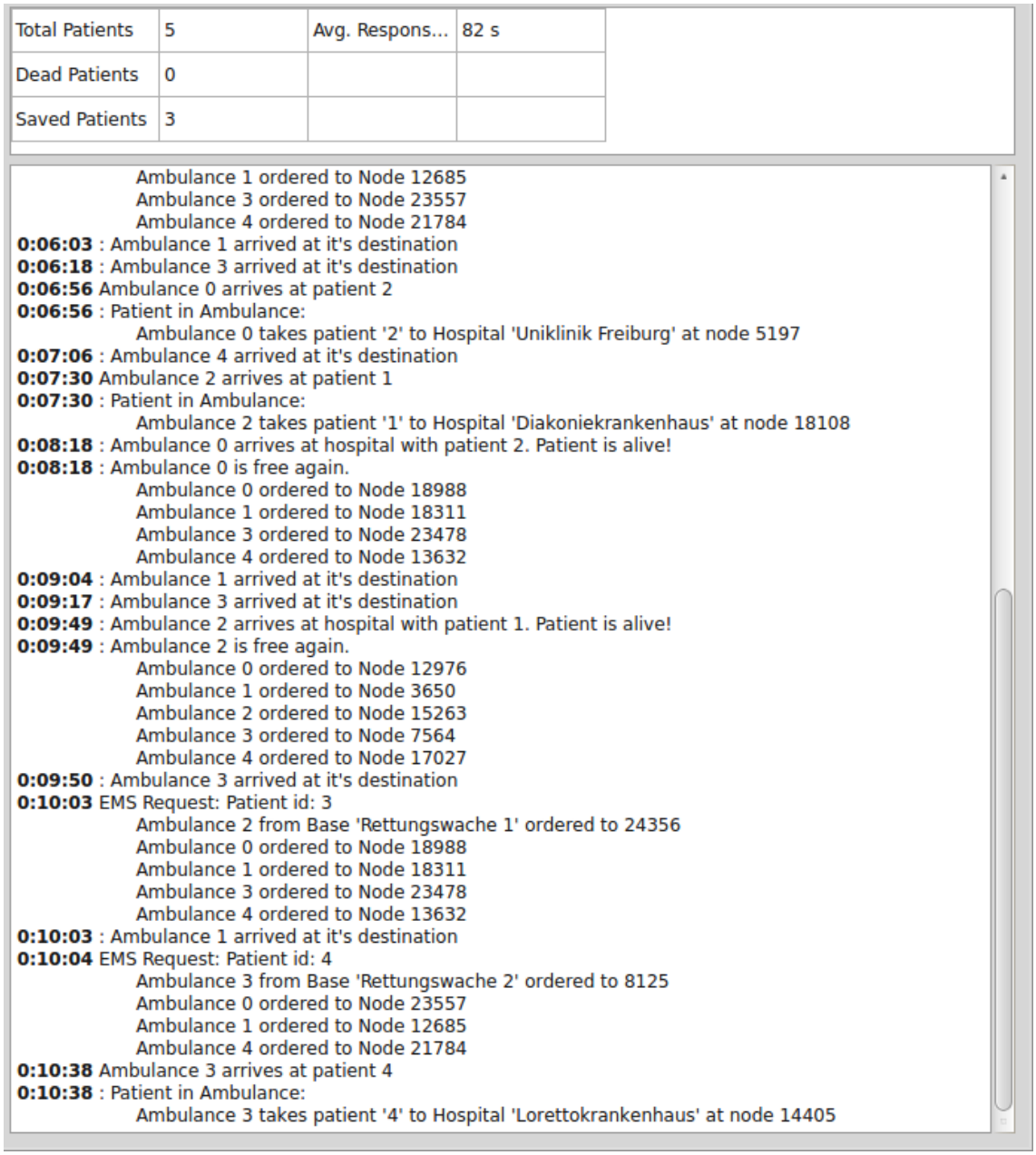

Figure 5.2: The statistics and log section of the GUI: The statistics field gives an overview of current patients and response times. The log underneath gives a detailed description of all events that happened and orders that were issued.

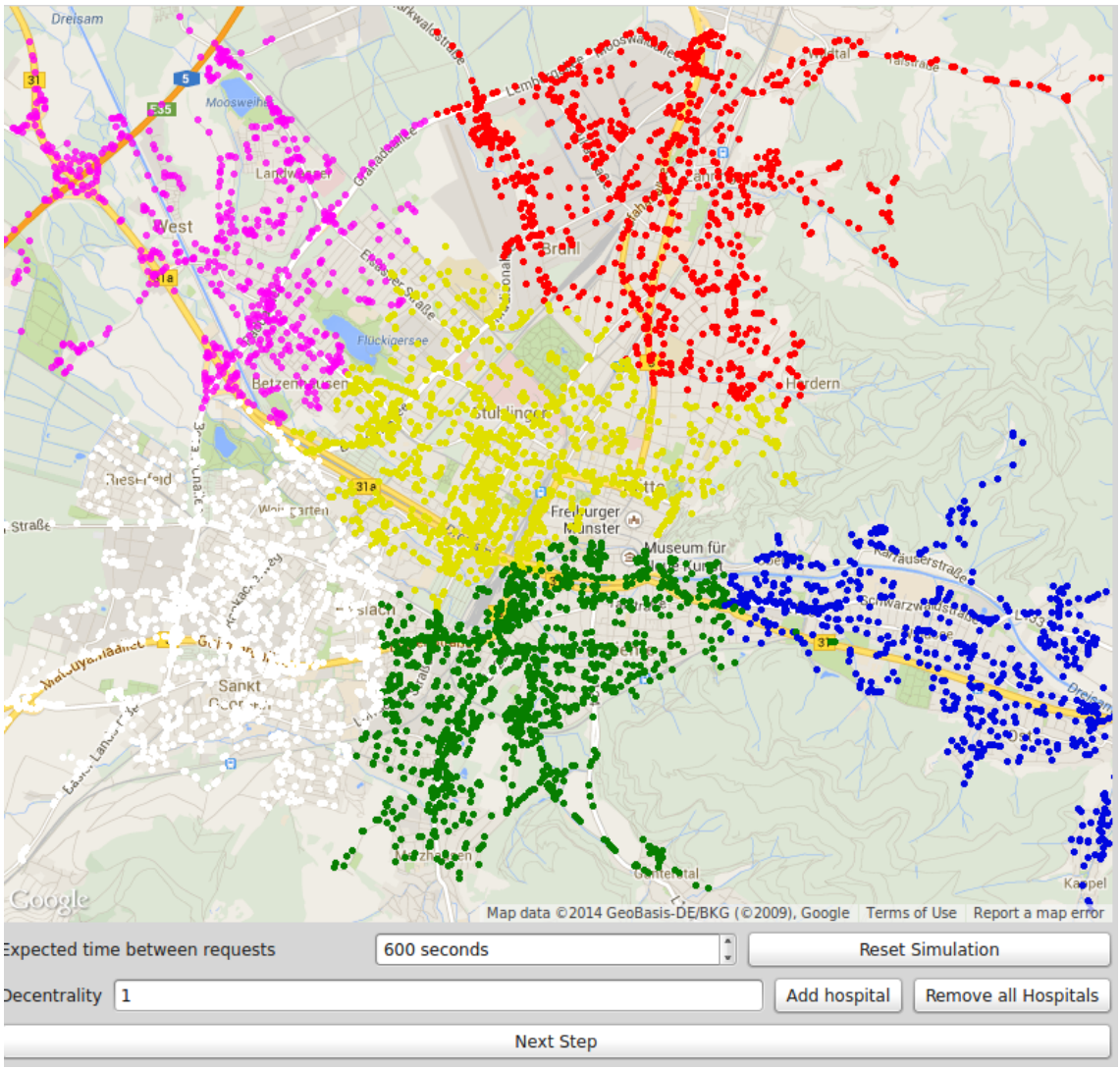

Figure 5.3: The GUI can also be used to display the results of a K-Medoids run. In this case $k = 6$.

6 Experimental Results

In this chapter we present and discuss the results of the simulations we ran. We first list the criteria we used when evaluating our strategies, then briefly discuss the running time of the simulations and finally present the benchmark results of our strategies in different environments.

6.1 Criteria

The question, what measures to use when evaluating the quality of an EMS system is not easy to answer, as – at least in the western world – EMSs are taken for granted and are expected to work perfectly. If a person calls an ambulance, it is expected, in Germany even required by law, that the ambulance be there promptly and if that person does not live to see the hospital, it is usually not due to the tardiness of the ambulance, but the medical condition of the patient.

Nevertheless we need to come up with measures to examine the quality of our strategies. Each of these measures corresponds to the result of one simulation, i.e., the strategy has processed one stream of n requests.

- **Number of patients saved:** The total number of patients that were saved by the EMS system, i.e., they were transported to a hospital before their "deadline". The deadline is not known to the EMS agent.
- **Average Response Time:** The average time an ambulance needed to respond to a request. All requests are counted, even if the patient is already dead when the ambulance arrives.
- **Average Time to Hospital:** The average time it took for a patient to arrive at a hospital after the request was made. This only takes saved patients into account.
- **Total distance travelled:** The sum of the distances all ambulances travelled during the simulation. This measure is introduced to acknowledge the fact that EMS providers often operate with tight budgets and want to operate in a money and energy saving way.

Time

In this section we present some timing information about our simulation runs for the sake of completeness. It is of no real relevance to the quality of the proposed strategies, as any one request handling operation takes less than 0.1 seconds and is thus faster than any human operator could be.

On a laptop with a 2.4 GHz Intel i5 processor and 8 GByte of RAM we ran simulations with 10 ambulances and 500 requests. As a Python interpreter we use CPython 2.7.

Medium Graph

On a medium sized graph with 26,000 nodes and 55,000 arcs:

	without reassignment	with reassignment
Greedy	5s	54s
K-Medoid	1m 6s	1m 43s
Voronoi	47s	1m 28s

Larger Graph

On a larger graph with 250,000 nodes and 500,000 arcs:

	without reassignment	with reassignment
Greedy	50s	7m0 s
K-Medoid	2m 45s	11m 45s
Voronoi	3m 30s	12m 0s

As described in section 5.2, the significantly longer running times on the larger graph are due to the unoptimized version of Dijkstra's algorithm. The reassignment system adds a lot of additional multi-source-Dijkstra calls and thus increases the running time.

It can be observed that Voronoi is faster thank K-Medoid on the smaller graph, but slower on the larger graph. This may be due to the fact that the number of hospitals and ambulance bases is larger, on average, than the number of medoids or that they are less favourably placed, which has a negative effect on the number of Dijkstra calls during the assignment of free ambulances. So in this case the running time does not only depend on the size of the graph.

6.2 Setup

In each simulation we compare different agents to one another in the exact same scenario. Since our requests are randomly generated, we create several distinct request streams for each scenario, run each agent with all of them and take the average of the results.

Apart from the random requests the scenarios are subject to the following parameters:

- **Street graph:** The nature of the underlying street graph has great influence on the scenario.
- **Ambulance Bases and Hospitals:** The number and distribution of ambulance bases and hospitals within the graph can be adjusted to demonstrate for example, the influence the hospital distribution has on a specific strategy.
- **Ambulances:** The number and placement of the ambulances.
- **Requests:** The randomly generated request stream. As described in section 3.3 we can adjust the expected time between requests and thus simulate more and less busy environments.
- **Time-to-live (TTL):** A patients time-to-live is the time after the request, at which they die and cannot be saved any more. Since we do not differentiate between different medical conditions, we use the same TTL for every patient during one simulation. If not otherwise stated we use twice the time an ambulance needs from the most remote node to the nearest hospital. That way we avoid impossible requests and get a number of more flexible requests from nodes that are close to a hospital and some more time critical ones from nodes farther away.

6.2.1 Centrality

In chapter 4 we made the claim, that K-Medoid is less sensitive to the locations of hospitals on the graph. To support this we need to measure the influence that the hospital location has on the different strategies. Before we can do this, we need to have a measure by which to quantify this position. The defining factor for the quality of a hospital position is its closeness to all other nodes. If the average distance to any other node is low, the hospital is in a good position, since it can be reached reasonably fast by all nodes. If it is high, the hospital is not optimally placed.

We introduce the two measures *centrality* and *decentrality*. Centrality is a concept stemming from sociology and social network graphs and is used for example to measure how central the role of a certain person within a social group is. There are a multitude of different definitions for centrality all applicable to different fields

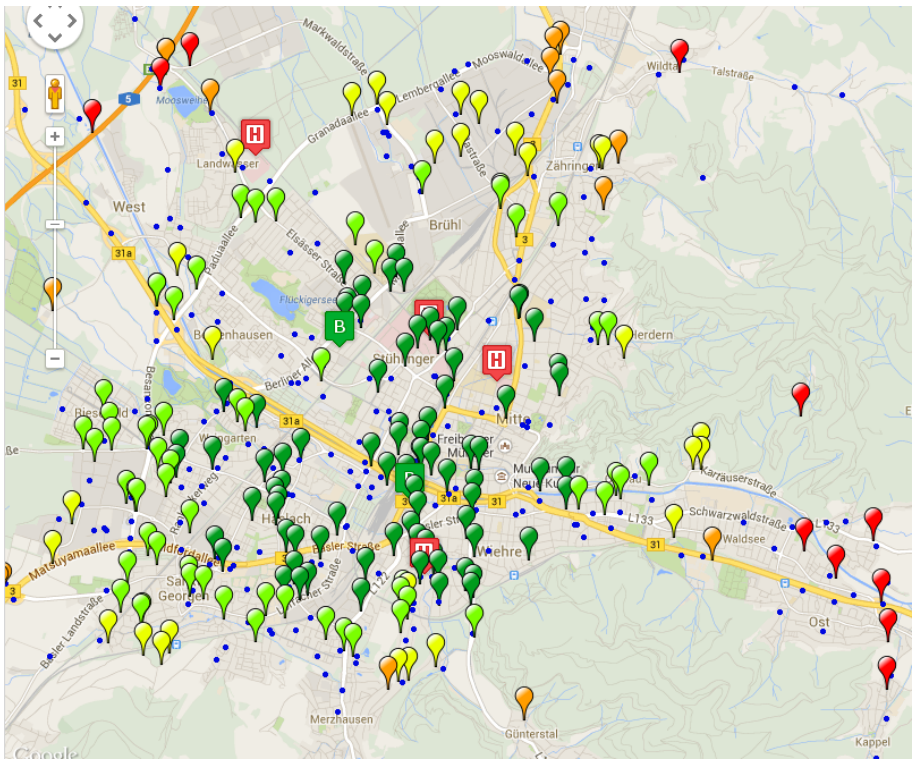

Figure 6.1: The color of the markers indicates the decentrality value of the marked node. Dark green: $1 - 1.25$, light green: $1.25 - 1.5$, yellow: $1.5 - 1.75$, orange: $1.75 - 2$, red > 2

of research. The one that comes closest to ours is *closeness centrality* described by Sabidussi in [Sab66].

With $d(v, u)$ being the length of the shortest path from v to u, we define the centrality of a node v as:

$$C(v) = \frac{\sum_{u \in V} d(u,v)}{|V|}$$

This gives us an absolute value that allows us to compare the centrality of two nodes. However, what we really want to measure is how bad a hospital is placed, i.e., how its position differs from an optimally placed hospital. To be able to do this, we define the *decentrality* $dc(v)$ of a node v as:

$$dc(v) = \frac{c(v)}{c(v_{opt})}$$

where v_{opt} is the optimal placement for a hospital on the graph, i.e. the node with the lowest centrality.

Calculation of the centrality of a node is rather expensive. It requires knowledge of the shortest path to every other node, i.e., a complete Dijkstra run. For this reason we estimate v_{opt} using the medoid of a K-Medoids run with $k = 1$.

6.2.2 Data Source

We obtain the used geographical information from OpenStreetMap (OSM) [HW08]. Their servers allow for custom areas to be selected and downloaded as an XML-file with extensive information about the region. Besides the road network there are also rivers, buildings and even trees present. We strip the downloaded data from all information except for nodes and ways (the OSM term for all different kinds of roads) and use it to construct our street graph. The length of arcs connecting two nodes is calculated using the geographic coordinates of the nodes and obtaining the straight line heuristic for the two nodes. Since the nodes are typically very close to each other this suffices for the purposes of our experiments and we do not need to take the earth's great circle into account. Since OpenStreetMap data contains closed off roads such as private parks we reduce the graph to its largest connected component to make sure each node in the network can be reached from any other node. To save time when loading the graph file, we save it, represented as JSON, and make use of the JSON parser from the Python standard library.

6.3 Evaluation of Upper Bounds

In section 2.3 we introduced two different techniques to calculate an upper bound for a given set of requests, i.e., an approximation of the number of patients an optimal offline algorithm could save. Both of these bounds are based on the concept of duty zones and the number of ambulances. The duty zone of a request is the (possibly empty) time period in which an ambulance has to be busy with this request in order to save the patient. Requests with a lot of flexibility and thus no duty zone do not influence the upper bounds. For the evaluation of the upper bounds we used half the TTL as described in section 6.2 to get less requests with no duty zones.

Figure 6.2 compares both upper bound techniques and shows that they are mainly influenced by the number of ambulances available. For smaller ambulance fleets we get a better bound using the method based on independent sets of duty zones and for bigger fleets the duty zone conflict based approach yields better results. For the remainder of this section "upper bound" will refer to the combination of both, i.e., the lower value for each set of requests.

To see how good our strategies perform compared to the upper bound, we evaluated K-Medoid with reassignment, the strategy that generally performed the best, against the upper bound. We used the street graph of Freiburg and simulated request intervals ranging from 60 to 260 seconds. We repeated this benchmark with two, four and six ambulances (see Figure 6.3). The evaluation confirms that the upper bound calculation depends more on the number of ambulances than on the request density: Although the bound does increase with growing request intervals, it does so slowly. However, it increases significantly when more ambulances are added.

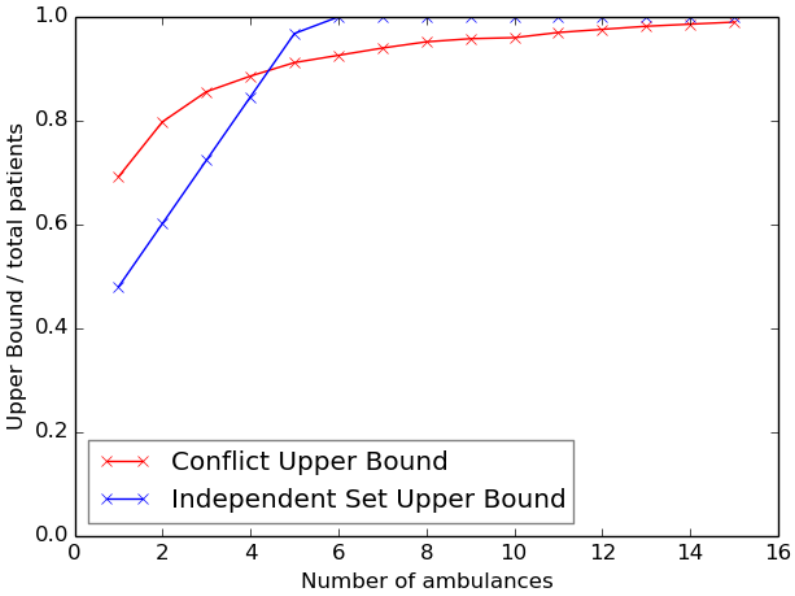

Figure 6.2: A comparison of the two upper bound approaches: For this evaluation, we used a synthetic dataset of 1000 requests with an expected time between requests of 5 seconds. For lower numbers of ambulances the independent set approach gives a better upper bound and for higher numbers of ambulances while the conflict based approach gives a better upper bound.

In conclusion, the K-Medoid with reassignment comes close to the upper bound for higher request intervals and higher numbers of ambulances, but falls short when the request stream is very dense and fewer ambulances are available. For the remainder of the evaluation we focused on comparing the different approaches against one another, for which we use TTLs as described in section 6.2 and bigger ambulance fleets. This gives us upper bounds equal to the total number of request, for which reason we do not list them explicitly.

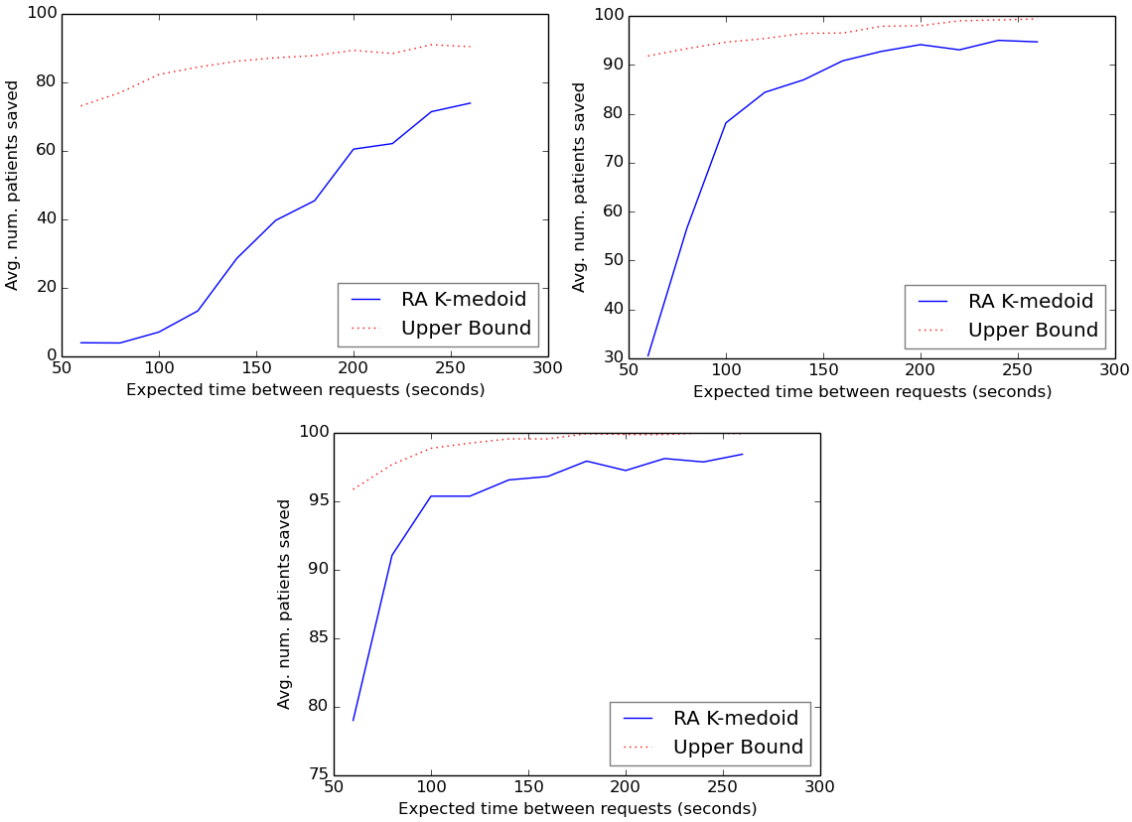

Figure 6.3: This figure shows a comparison between the upper bound and the actual performance of K-Medoid with reassignment. This strategy was chosen, since it generally performed best. The evaluation was repeated for two ambulances (top left), four ambulances (top right) and six ambulances (bottom).

6.4 Strategy Benchmark Results

In this section we present the actual results of the benchmark runs we did. We used two different modes: One with a fixed set of hospitals and variable expected time between requests, which we use to observe the change in performances as the busyness of the system changes. The other mode has a fixed expected time between requests and a variable decentrality. In this mode we add one hospital with a certain decentrality for each run and track the influence of hospital centrality on the strategies.

6.4.1 Fixed set of hospitals and variable time between requests

In this variant of our benchmarking method, we have a fixed set of hospitals, ambulance bases and ambulances for each run and change the *expected time between requests* variable.

Scenario 1: Medium sized city

First we benchmarked our strategies in a real world example. We used the street graph of Freiburg, a city with about 200,000 inhabitants. The hospitals, ambulance bases and ambulances used in the simulation correspond to those actually present in Freiburg (see Figure 6.4).

Nodes	25447
Arcs	54944
Hospitals	4
Ambulance Bases	2
Ambulances	5
Requests per run	100
Time between requests range	60s - 180s

Table 6.1: Parameters of scenario 1

We simulated a time between requests (e_{tbr}) range from 60s to 180s with 100 patients per run.

The results (Figure 6.5) of this benchmark show the following:

- **Total patients saved:** With low e_{tbr} values all strategies perform equally bad and are only able to save 50% or less of the patients. With increasing intervals K-Medoid has a slight advantage over the other two other approaches, saving about two to three patients more on average.

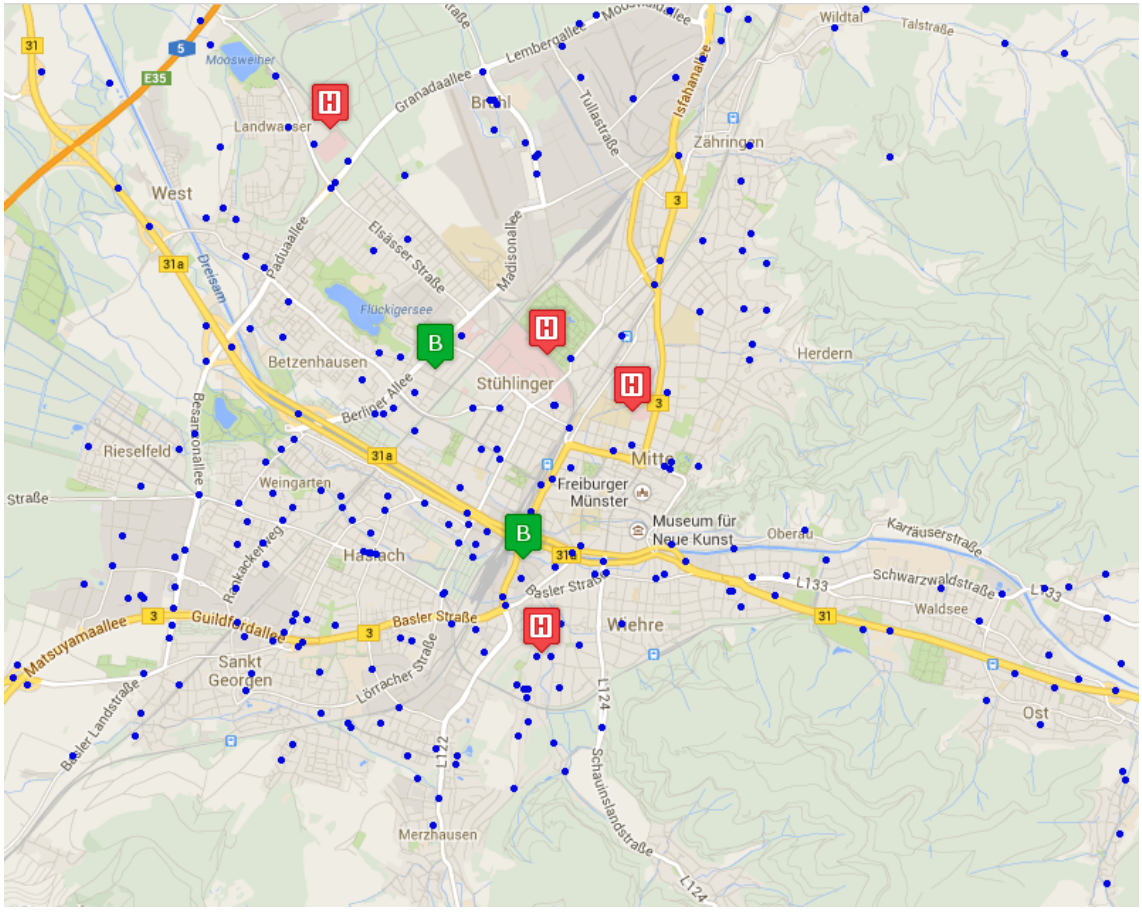

Figure 6.4: This figure shows the graph we use in scenario one: Every 100th node is represented as a blue dot to give an idea about the graphs area. The red markers represent the 4 hospitals actually present in Freiburg and the green markers the two ambulance bases.

- **Response time and time to hospital:** K-Medoid is also slightly better in those categories: With increasing request intervals it responds about 30 to 40 seconds faster to requests than the other two approaches.
- **Total distance driven:** This measure shows the costs involved in the K-Medoid strategy. The constant redeployment of ambulances results is almost double the total distance of Greedy with higher request intervals. With lower e_{tbr} values the difference is less significant, because here ambulances have to abort redistribution moves more often and go to the next patient. Voronoi is only slightly worse than Greedy, since often only a few ambulances have to move during a redeployment.

In conclusion, scenario one shows that although K-Medoid is slightly better, Greedy performs reasonably well, provided a more or less favourable distribution of hospitals and ambulances. Voronoi does not provide any improvement over Greedy in this

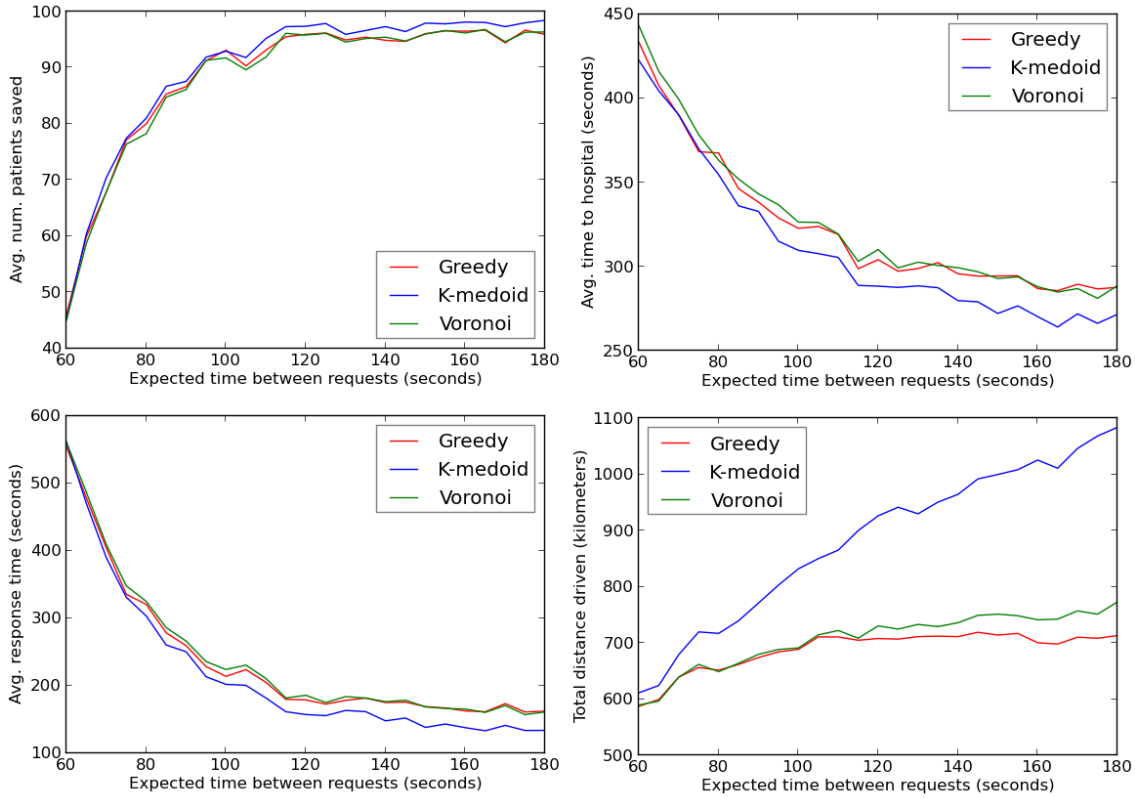

Figure 6.5: In a medium sized city with many and well distributed hospitals and ambulances all algorithms perform similarly. K-Medoid has a slight advantage in response/to-hospital times and patients saved, while performing poorer in total distance travelled.

scenario.

Scenario 2: A metropolitan area with unbalanced ambulance distribution

For the second scenario we use the street graph of the Rhine-Neckar metropolitan region. A dense city cluster with approximately 2.5 million inhabitants. The region consist of the three cities Ludwigshafen, Mannheim and Heidelberg and surrounding municipalities.

We again distributed hospitals relatively evenly on the map, but restricted the ambulance bases to one half of the map. This is to simulate a scenario where half of the ambulance fleet is not operational due to fire or flood in a base or not reachable due to communications outage (see Figure 6.6).

Nodes	133,416
Arcs	292,338
Hospitals	5
Ambulance Bases	5
Ambulances	10
Requests per run	100
Time between requests range	100s - 220s

Table 6.2: Parameters of scenario 2

The results (Figure 6.7) of this benchmark show the following:

- **Total patients saved:** This measure shows a clear advantage of the two redeployment strategies over Greedy. For all tested e_{tbr} values K-Medoid is able to save about five to ten patients more than Greedy. Voronoi ranks in the middle with two to five patients more than Greedy.
- **Response time and time to hospital:** The ranking from the first measure is confirmed in response time and to hospital time. Here K-Medoid performes best, followed by Voronoi and with a greater gap Greedy. With large request intervals K-Medoid responds about 2.5 minutes and Voronoi about two minutes faster than Greedy.
- **Total distance driven:** As in scenario one, ambulances managed by K-Medoid travel by far the greatest overall distance. However while the total distance driven by K-Medoid was almost double the total distance of Greedy in scenario one, it is only 1.4 times Greedy's distance in scenario 2. Since Greedy always sends the ambulances back to the western part of the map when they are idle, they have to travel very long distances and thus Greedy is even outperformed by Voronoi.

Scenario two shows the advantages of strategies using redeployment over Greedy in less optimal environments. While Greedy suffers from the bad distribution of ambulances the other two approaches are not influenced as much. Especially the value of Voronoi is demonstrated: It almost reaches the performance of K-Medoid and is very economical even compared to Greedy.

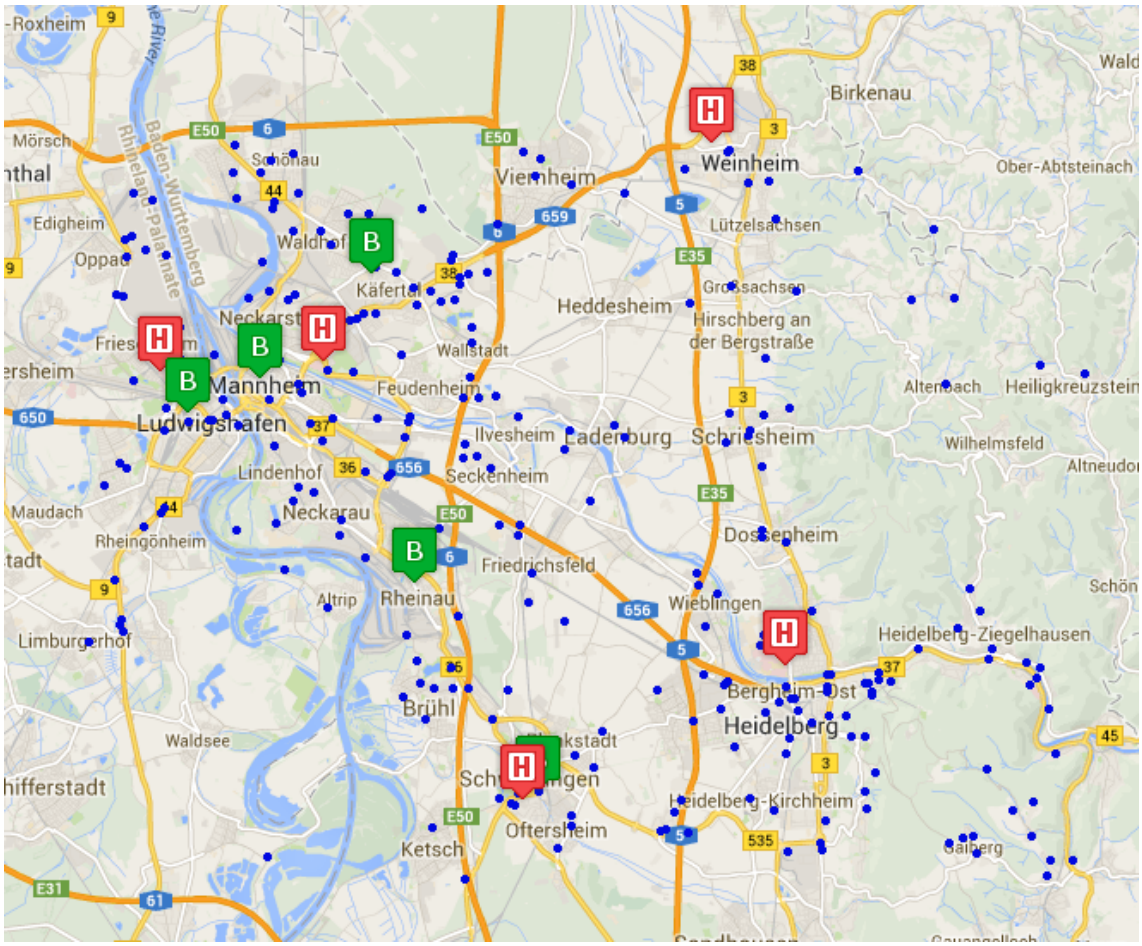

Figure 6.6: In this scenario all ambulance bases in the eastern part of the service area have become non-operational, which has to be compensated by the rest of the fleet. The blue dots is a subset of the nodes in the graph to demonstrate its dimensions. Each of the green ambulance bases holds two ambulances.

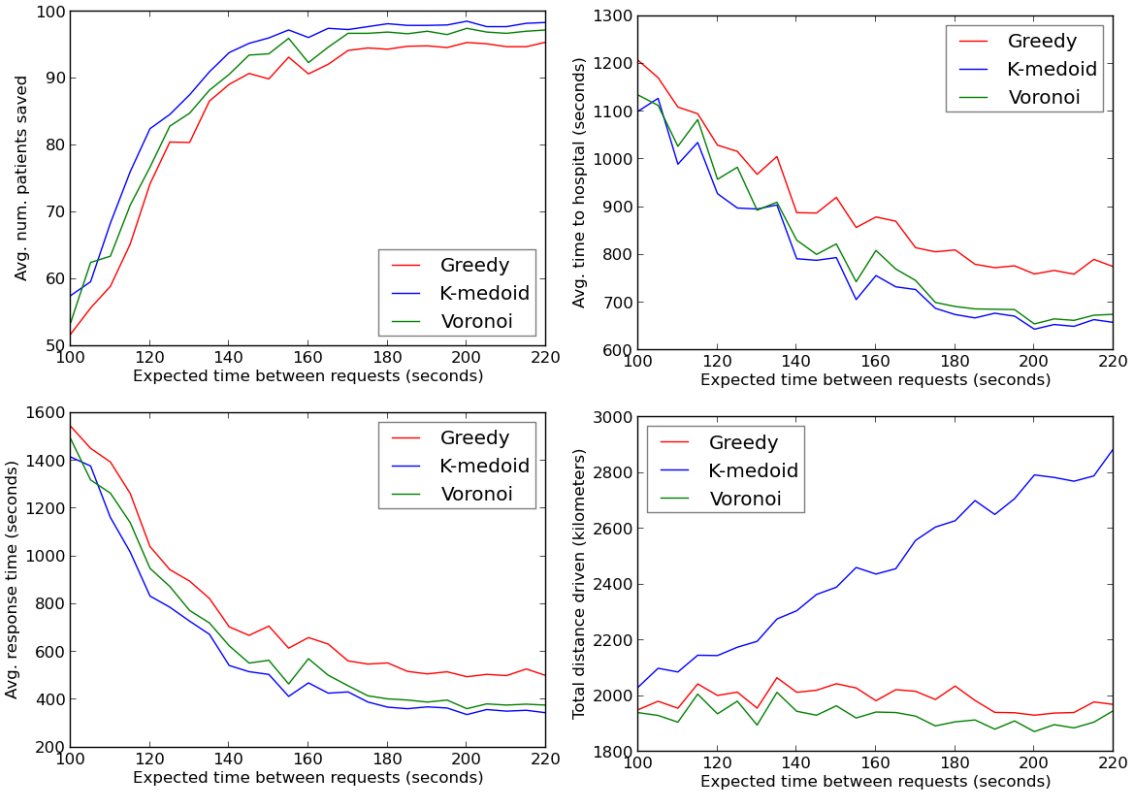

Figure 6.7: The results of scenario 2: The least infrastructure dependant strategy, K-Medoid, performs best except for the economic measure. This also shows how Voronoi, making use of the hospitals in the eastern area as ambulance positions is a good compromise between Greedy and K-Medoid.

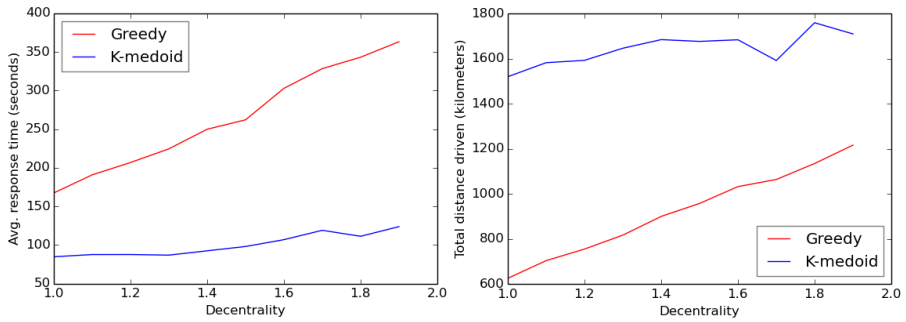

Figure 6.8: When provided with 12 ambulances K-Medoid is able to compensate the increasing decentrality very well. The averages response time does not change very much until the hospital is really far on the outside of the graph.

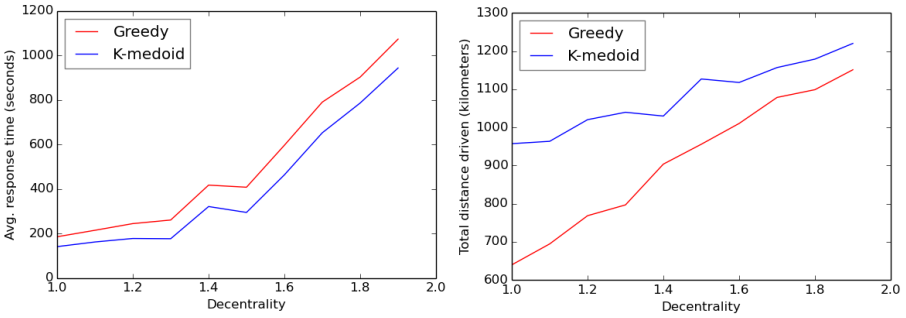

Figure 6.9: With fewer ambulances (in this case 6) the compensation K-Medoid provides is less significant.

6.4.2 Decentrality Based Evaluation

In this section we evaluate the performance of our strategies in relation to centrality of the objects on the graph. We argued before that the position of objects, i.e. hospitals and ambulance bases, influences EMS operations.

In this benchmark mode we chose a fix e_{tbr} value and for each run add only one hospital to the graph with a certain decentrality value and position all ambulances there. We increase the decentrality and observe how it affects the outcome of our strategies. We use the street graph of Freiburg and chose a request interval of $100s$, as this proved to provide a moderatly busy stream in the previous benchmarks.

As figure Figure 6.8 shows, when given many ambulances, K-Medoid can compensate non optimally placed hospitals rather well. Greedy however is sensitive to hospital placement and has increasing response times proportional to the hospitals decentrality. The reason for this is the redistribution K-Medoid does. We did not benchmark Voronoi, since for one hospital it is equivalent to Greedy.

When the number of Ambulances is decreased, K-Medoid ability to compensate for hospital decentrality becomes less effective, as Figure 6.9 shows. The compensation

can only be successful if idle ambulances are available to cover the medoids. With fewer ambulances the average number of idle ambulances at each point in time becomes so low that the requests queue grows. In that case ambulances will start to the next patient, when they just dropped another patient off at the hospital. This means the behaviour is not different from Greedy's, and thus the performance is similar.

6.5 Adding Reassignment

When we added the reassignment strategy described in section 4.5 to all three strategies, we could significantly increase their performance. Especially with high request frequencies all four measures improve for all three strategies. High request frequencies lead to a lower chance of having a close free ambulance available once a new request comes in. This forces bad assignments if reassignment is not allowed. These bad assignments can never be corrected, as without reassignment an ambulance has to respond to a request once it is assigned to one.

We repeated the simulation of both scenarios from subsection 6.4.1, this time with additional runs for the reassignment version of the three strategies.

Scenario 1

The results (Figure 6.10 for scenario one show a decrease of average response time for request intervals of $60s$ by about one minute and 40 seconds between the non reassingment version and the reassignment version of the same strategy. This leads to about 10 to 15 saved patients more for the same interval.

Scenario 2

For the second scenario, the one with the unbeneficial ambulance base positions, the benefits of reassignment are even higher (see Figure 6.11). Here the reassignment versions of the strategies are able to save about 30 patients more than the regular versions, while decreasing the average response time by over 50%.

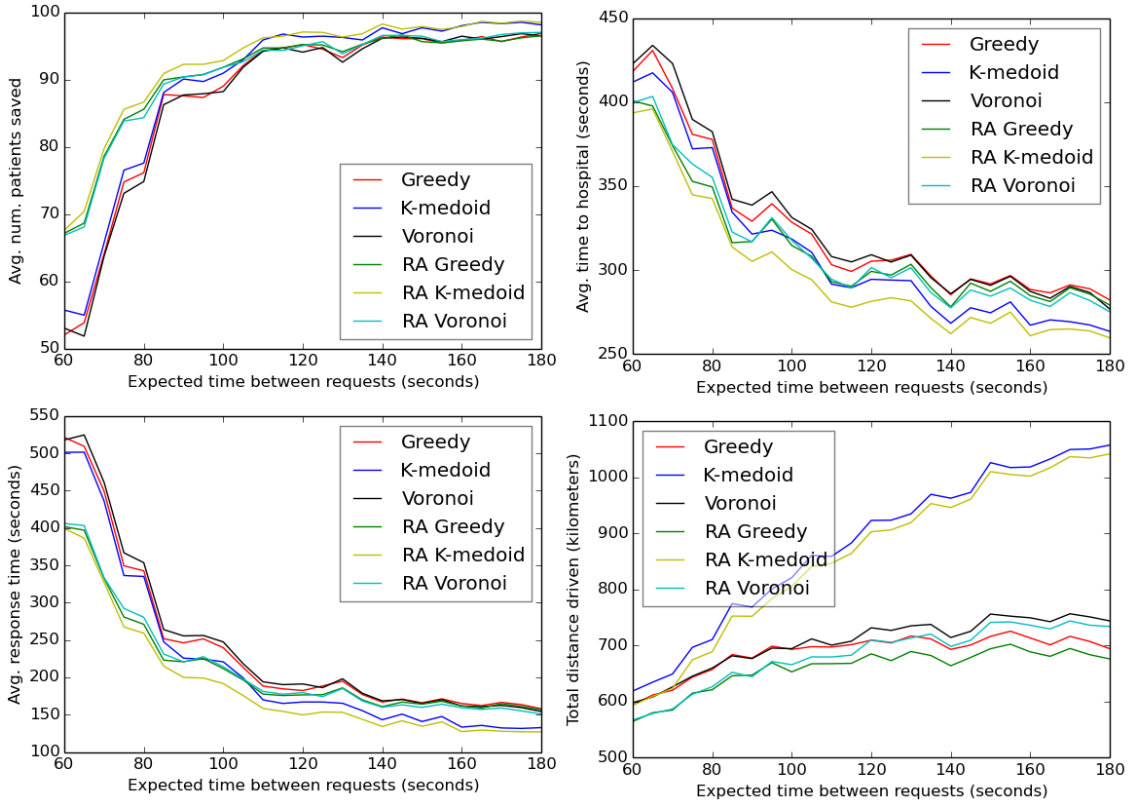

Figure 6.10: Scenario 1: The performance of all three strategies improves when we add reassignment. Especially with high requests frequencies reassignment increases the number of saved patients, lowers response and to hospital times and also decreases the total distance driven.

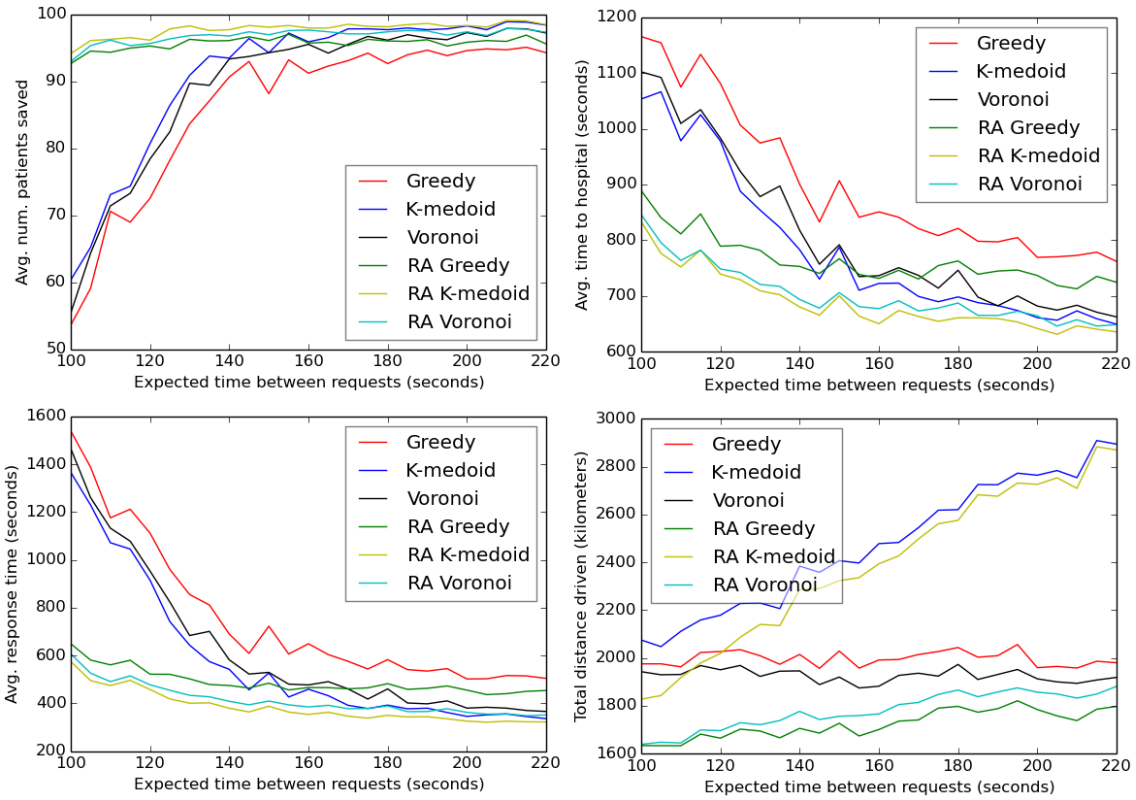

Figure 6.11: Scenario 2: The results for scenario two with reassignments also show a performance gain for all three strategies.

7 Conclusion

In this thesis we described and examined the Ambulance Fleet Management problem. We gave a definition of the problem and derived a general model as a basis for the development of solution strategies.
We then presented a formal analysis of the problem, introducing the concept of duty zones to find upper bounds and proved the *One-Hospital-One-Ambulance* problem to be NP-hard.

We designed three distinct strategies to approach the problem. The naive Greedy approach, the economically costly K-Medoid approach and the Voronoi approach as a compromise between economical and medical performance. To improve those strategies, we introduce the concept of reassignment, which allows already assigned ambulances to be reassigned to different patient, once new requests come in.

We implemented an event-based simulator, which we use to benchmark our approach in different scenarios. To visualize results and to observe the behaviour of our strategies we added a graphical user interface (GUI). The implementation of the simulator and GUI is done in Python, C++ and Javascript. This combination is used to make use of the flexibility and ease of development of Python and Javascript and the efficiency of C++.

Our evaluation shows that Greedy performs reasonably well in a well structured EMS setting, but is unable to efficiently adapt to less favourable environments. K-Medoid on the other hand is very resistant to bad distributions of ambulances and bases, but lets ambulances travel much longer distances than Greedy. Voronoi, which was developed as a compromise approach could be shown to perform equally as good as Greedy in environments with good infrastructure and better than Greedy in environments with worse infrastructure while also staying very economical. We also examined the running time of our implementations and although the simulation does not scale very well due to the underlying unoptimized shortest path algorithms, all decision making processes which the strategies use to react to events are fast enough to be deployed as a real time assistance system for EMS operators.

7.1 Future Work

The main scope of this thesis was to develop and study different approaches to the ambulance fleet management problem in a general way. Many of the processed sub-

tasks gave opportunities for further research. In this section we present some of the ideas we have for optimizations or future work on related topics.

7.1.1 Improving the Performance

The EMS coverage of a country is usually partitioned into small autonomous sections operating in a defined area, for example a city. Our implementation has been shown to be fit to simulate an EMS system of such a size. Some special special settings might need the ability to simulate bigger areas. For example, in Germany the network for intensive care inter hospital transfer vehicles is coordinated on federal state level and some medical helicopters even on a national level. To simulate such large systems, we suggest a reimplementation of the main simulator loop in C++ or a similarly fast compiled language. Furthermore the Dijkstra implementation could be replace with one of the available optimizations, such as contraction hierarchies or arc flags.

To make further improvements to the simulation performance it is possible to make more use of parallel computing on a machine with more cores or even a computer cluster.

7.1.2 Incorporation of Secondary Ambulances

Many EMS implementations around the world differentiate between primary or emergency patients and secondary patients. Requests by secondary patients are less urgent as they are for example to be transported from one hospital to another or waiting at a hospital after treatment to be brought back home. In developed countries, especially in urban areas, these secondary patients are the majority of patients and EMS providers have adapted to this situation by providing special secondary ambulances which carry less equipment and are thus cheaper to maintain. For this reason most EMS dispatch centers have to coordinate two different and almost disjunct services. An idea for further research would be to incorporate secondary ambulances and patients into the strategies we presented to make better use of the possible synergy effects of these two related services. In times of low emergency request frequency primary ambulances could be used to help out in inter hospital transfer and in buys times secondary ambulances could be used as first responders to emergency patients.

7.1.3 More Detailed Patient and Hospital Modelling

In our model we made the assumption, that any patient can be treated in any hospital. In reality this is not true. Many medical conditions like strokes, heart attacks or amputations require special medical equipment and/or specially trained

personnel to be treated successfully. Most hospitals do not provide the capacity to treat all medical emergencies. A more detailed model could provide multiple classes of patients and hospitals and give the opportunity to examine an EMS system in a more detailed way.

7.1.4 Examine more Special Cases

We aimed to model an EMS environment in a very general way such that it can be applied to many different geographical areas and EMS infrastructures. This leaves aside special cases of emergency care such as disaster relief or medical assistance in large events such as marathons or music festivals. While the general EMS setting does not allow many predictions about future requests, such special events may offer more basis for planning ahead. For example at a marathon it could be reasonable to assume a higher number of patients at the finish area once the main group of runners is expected to finish. In a disaster case, depending on the type, it is also reasonable to expect many patients at the site of the disaster.

Such special scenarios provide opportunity to develop a more specialized request generation and strategies which are capable of incorporating planning ahead based on scenario specific information.

Bibliography

[AGK03] Abrahams, David ; Grosse-Kunstleve, Ralf W.: Building hybrid systems with Boost. Python. In: *CC Plus Plus Users Journal* 21 (2003), Nr. 7, S. 29–36

[AGP12] Azi, Nabila ; Gendreau, Michel ; Potvin, Jean-Yves: A dynamic vehicle routing problem with multiple delivery routes. In: *Annals of Operations Research* 199 (2012), Nr. 1, S. 103–112

[Aur91] Aurenhammer, Franz: Voronoi diagrams—a survey of a fundamental geometric data structure. In: *ACM Computing Surveys (CSUR)* 23 (1991), Nr. 3, S. 345–405

[AV06] Arthur, David ; Vassilvitskii, Sergei: How Slow is the k-means Method? In: *Proceedings of the twenty-second annual symposium on Computational geometry* ACM, 2006, S. 144–153

[BL74] Berlin, Geoffrey N. ; Liebman, Jon C.: Mathematical analysis of emergency ambulance location. In: *Socio-Economic Planning Sciences* 8 (1974), Nr. 6, S. 323–328

[BLS03] Brotcorne, Luce ; Laporte, Gilbert ; Semet, Frederic: Ambulance location and relocation models. In: *European journal of operational research* 147 (2003), Nr. 3, S. 451–463

[CHS+13] Carnes, Timothy A. ; Henderson, Shane G. ; Shmoys, David B. ; Ahghari, Mahvareh ; MacDonald, Russell D.: Mathematical programming guides air-ambulance routing at Ornge. In: *Interfaces* 43 (2013), Nr. 3, S. 232–239

[CL03] Cordeau, Jean-François ; Laporte, Gilbert: The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. In: *Quarterly Journal of the Belgian, French and Italian Operations Research Societies* 1 (2003), Nr. 2, S. 89–101

[Dij59] Dijkstra, Edsger W.: A note on two problems in connexion with graphs. In: *Numerische mathematik* 1 (1959), Nr. 1, S. 269–271

[FBG13] Ferrucci, Francesco ; Bock, Stefan ; Gendreau, Michel: A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. In: *European Journal of Operational Research* 225 (2013), Nr. 1, S. 130–141

[GB07] Gong, Qiang ; Batta, Rajan: Allocation and reallocation of ambulances to casualty clusters in a disaster relief operation. In: *IIE Transactions* 39 (2007), Nr. 1, S. 27–39

[GJ79] Garey, Michael R. ; Johnson, David S.: Computer and intractability. In: *A Guide to the NP-Completeness. New York, NY: WH Freeman and Company* (1979)

[GSSD08] Geisberger, Robert ; Sanders, Peter ; Schultes, Dominik ; Delling, Daniel: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: *Experimental Algorithms*. Springer, 2008, S. 319–333

[Hal72] Hall, William K.: The application of multifunction stochastic service systems in allocating ambulances to an urban area. In: *Operations Research* 20 (1972), Nr. 3, S. 558–570

[HKP06] Han, Jiawei ; Kamber, Micheline ; Pei, Jian: *Data mining: concepts and techniques*. Morgan kaufmann, 2006

[HW08] Haklay, Mordechai ; Weber, Patrick: Openstreetmap: User-generated street maps. In: *Pervasive Computing, IEEE* 7 (2008), Nr. 4, S. 12–18

[KHS12] Knight, VA ; Harper, PR ; Smith, L: Ambulance allocation for maximal survival with heterogeneous outcome measures. In: *Omega* 40 (2012), Nr. 6, S. 918–926

[KR90] Kaufman, Leonard ; Rousseeuw, Peter J.: Partitioning around medoids (program pam). In: *Finding groups in data: an introduction to cluster analysis* (1990), S. 68–125

[Kuh55] Kuhn, Harold W.: The Hungarian method for the assignment problem. In: *Naval research logistics quarterly* 2 (1955), Nr. 1-2, S. 83–97

[Lau04] Lauther, Ulrich: An Extremely Fast, Exact Algorithm for Finding Shor test Paths in Static Networks with Geographical Background. (2004)

[Llo82] Lloyd, Stuart: Least squares quantization in PCM. In: *Information Theory, IEEE Transactions on* 28 (1982), Nr. 2, S. 129–137

[Mat08] Matloff, Norm: Introduction to discrete-event simulation and the simpy language. In: *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2 (2008), S. 2009

[Mun57] Munkres, James: Algorithms for the assignment and transportation problems. In: *Journal of the Society for Industrial & Applied Mathematics* 5 (1957), Nr. 1, S. 32–38

[Par09] Parragh, Sophie: *Ambulance routing problems with rich constraints and multiple objectives*, uniwien, Diss., 2009

[PD09] Panahi, S ; Delavar, MR: Dynamic Shortest Path in Ambulance Routing Based on GIS. In: *International Journal of Geoinformatics* 5 (2009), Nr. 1

[RHT09] Restrepo, Mateo ; Henderson, Shane G. ; Topaloglu, Huseyin: Erlang loss models for the static deployment of ambulances. In: *Health care management science* 12 (2009), Nr. 1, S. 67–79

[Sab66] Sabidussi, Gert: The centrality index of a graph. In: *Psychometrika* 31 (1966), Nr. 4, S. 581–603

[SH05] Snoeyink, Jack ; Hill, UNC C.: Maximum Independent Set for Intervals by Divide-Prune-and-Conquer. In: *CCCG*, 2005, S. 264–265

[YMK12] Yue, Yisong ; Marla, Lavanya ; Krishnan, Ramayya: An Efficient Simulation-Based Approach to Ambulance Fleet Allocation and Dynamic Redeployment. In: *AAAI*, 2012

[ZM93] Zhu, Zhiwei ; McKnew, Mark A.: A goal programming workload balancing optimization model for ambulance allocation: An application to Shanghai, P.R.China. In: *Socio-Economic Planning Sciences* 27 (1993), Nr. 2, S. 137 – 148. – ISSN 0038–0121