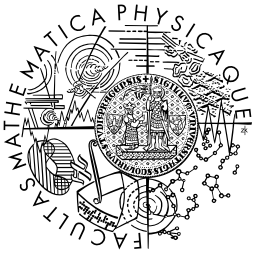


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Tomáš Arnold Tillmann

Optimalizace plánu směn pohotovostních služeb

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Adam Šmelko

Studijní program: Informatika

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Rád bych vyjádřil své upřímné poděkování všem, kteří mi pomohli a podporovali mě při psaní této práce. V první řadě děkuji svému vedoucímu práce, Mgr. Adamovi Šmelkovi, za cenné rady, trpělivost a odborné vedení, které mi během celého procesu tvorby této práce poskytl. Jeho znalosti a zkušenosti byly pro mě neocenitelné. Dále bych rád poděkoval své rodině za jejich podporu. Velké poděkování patří také mým přátelům za jejich pomoc, podporu a inspiraci.

Název práce: Optimalizace plánu směn pohotovostních služeb

Autor: Tomáš Arnold Tillmann

Department: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Adam Šmelko, Katedra distribuovaných a spolehlivých systémů

Abstrakt: Nalezení optimálního naplánování týmů záchranářů a záchranných vozidel je zásadním problémem každé pohotovostní služby. Pohotovostní služba by při optimálním naplánování měla být schopna úspěšně odbavit co největší počet incidentů za co nejmenší vynaložené náklady v rámci daného časového úseku, například jednoho dne. Tato práce se zabývá metodami, které úspěšně naleznou taková optimální naplánování. Nalezené metody jsou aplikovány a zkoumány na konkrétní případové studii, na pražské pohotovostní službě. Následně jsou mezi sebou jednotlivé metody porovnány a na závěr je vyhodnoceno, jaké metody umí nalézt nejlepší plán pohotovostní služby, a jsou tak nejvhodnějšími pro praktické využití.

Klíčová slova: Optimalizace, Pohotovostní Služba, Záchranná Služba, Optimální Plán, Simulace

Title: Emergency services shift plan optimization

Author: Tomáš Arnold Tillmann

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Adam Šmelko, Department of Distributed and Dependable Systems

Abstract: Finding the optimal scheduling of rescue service teams and vehicles is a fundamental problem for any emergency service. An emergency service, with optimal scheduling, should be able to successfully handle as many incidents as possible for the least cost within a given time period, for example, one day. This thesis discusses methods to successfully find such optimal scheduling. The found methods are applied and investigated on a specific case study, the Prague Emergency Service. The methods are then compared with each other and finally, it is evaluated which methods can find the best emergency service plan and are thus the most suitable for practical use.

Keywords: Optimization, Emergency Service, Optimal Plan, Simulation

Obsah

Úvod

Zásadním úkolem, se kterým se potýkájí všechny pohotovostní služby, je plánování konkrétních směn. Hlavním cílem tohoto plánování je rozdělení a rozvržení týmů záchranářů a dostupných záchranných vozidel na výjezdové stanice tak, aby byly jednotlivé incidenty efektivně odbaveny. Pod tím si můžeme představit, že během daného časového úseku, například jednoho dne, chceme úspěšně odbavit co největší počet incidentů, a to při nejnižších možných vynaložených nákladech. Nalézt takový optimální a efektivní plán je poměrně komplexní úkol. Zejména pak pro pohotovostní služby vlastnící vyšší stovky záchranářů a vozidel, které jsou velmi časté například ve Spojených státech amerických. Nalézt takový plán je ale velmi žádoucí, protože efektivní využití zdrojů může vést až k několikanásobně úspěšnějšímu počtu odbavených incidentů za nižší potřebné vynaložené náklady.

Cílem této práce je představit, aplikovat a porovnat metody, které nám pomohou taková optimální naplánování směn nalézt. Práce je rozdělena do tří částí. V první části je důkladně popsán a zformalizován řešený problém jako optimalizační úloha a jakým způsobem namodelujeme pohotovostní službu a sadu incidentů. Dále si popíšeme, jak konkrétně vyhodnotíme, kolik incidentů plán úspěšně odbaví. Budeme zkoumat chování plánu pohotovostní služby na dané sadě incidentů, pomocí deterministické diskrétní simulace. Také si zaanalyzujeme řešený problém, a o nalezené poznatky se budeme opírat při nalézání řešících metod, které jsou diskutovány v druhé části práce.

V druhé části se zabýváme několika možnými řešeními optimalizační úlohy. První se pokusíme nalézt rychlá řešení pomocí dynamického programování. Ukáže se, že jsme schopni nalézt rekurzivní vztahy mezi většími a menšími podproblémy, avšak v nejhorším případě se stále nebude jednat o řešení v polynomiálním čase. Na další metody, které se podíváme, budou metaheuristické přístupy, které se pokusí nalézt alespoň dostatečně dobrá řešení za rozumnou dobu. Podíváme se na nejčastěji používané metaheuristiky, které se běžně používají pro řešení náročných kombinatorických úloh.

V třetí části jednotlivé metody aplikujeme na reálnou pohotovostní službu, a to konkrétně na pohotovostní službu hlavního města Prahy. Budeme zkoumat nalezené optimální plány jednotlivými metodami a následně je mezi sebou porovnáme. Na závěr na základě dat vyhodnotíme, které metody jsou nejlepšími pro nalezení optimálních plánů, a které jsou doporučené pro reálné využití. Tím bude naplněn cíl práce, kterým bylo nalézt nejvhodnější metodu, řešící problém nalezení optimálního pohotovostního plánu.

1 Převedení na optimalizační úlohu

1.1 Formalizace problému

V první řadě je potřeba si problém plánování směn matematicky vymezit a popsat, abychom byli následně schopni problém systematicky řešit. Právě proto je tato kapitola věnována formalizaci problému a představení klíčových pojmů.

Pohotovostní služba má k dispozici *týmy záchranářů* $Z = \{z_1, z_2, \ldots, z_{Z_n}\}$, *záchranná vozidla* $A = \{a_1, a_2, \ldots\ a_{A_n}\}$ a *výjezdové stanice* $V = \{v_1, v_2, \ldots, v_{V_n}\}$. Na území působnosti pohotovostní služby se nachází nemocnice $H = \{h_1, h_2, \ldots h_{H_n}\}$. Pohotovostní služba definuje *pracovní směny* týmů záchranářů $D = D_s \times D_l$, kde $D_s \in \mathbb{N}_0$ množina *začátků pracovních směn* a $D_l \in \mathbb{N}_0$ množina *délek pracovních směn. Pracovní směna* $d \in D$ je tak dvojice $d = (d_s, d_l)$, kde $d_s \in D_s$ značí začátek a $d_l \in D_l$ délku trvání směny. Nechť $D_n = |D|$.

Plánem pohotovostní služby rozumíme přiřazení týmů záchranářů $z \in Z$ a záchranných vozidel $a \in A$ na konkrétní výjezdové stanice $v \in V$ a přiřazení pracovní směny $d \in D$ každému týmu záchranářů $z \in Z$ v rámci jednoho dne. Tato přiřazení popíšeme *přiřazovacími funkcemi*:

Definice 1 (Přiřazovací funkce)**.**

$$
\begin{aligned}
&p_Z \colon Z \to V \cup \{v_\emptyset\} && \textit{přiřazení týmů na stanice},\\
&p_A \colon A \to V \cup \{v_\emptyset\} && \textit{přiřazení vozidel na stanice},\\
&p_D \colon Z \to D && \textit{přiřazení směn týmům.}
\end{aligned}
$$

Tým $z \in Z$ nemá přiřazenou žádnou výjezdovou stanici právě tehdy, když $p_Z(z) = v_\emptyset$ a $p_D(z) = (d_s, d_l) \colon d_l = 0$. Záchranné vozidlo $a \in A$ nemá přiřazenou žádnou výjezdovou stanici, právě tehdy, když $p_A(a) = a_\emptyset$.

Pro pohodlnost zaveďme:

$$
\begin{aligned}
p_D(z) = (d_s, d_l) &\Rightarrow p_{D_s}(z) = d_s,\\
p_D(z) = (d_s, d_l) &\Rightarrow p_{D_l}(z) = d_l.
\end{aligned}
$$

Plánem pohotovostní služby budeme označovat čtveřici

$$p = (p_Z, p_A, p_{D_s}, p_{D_l}) \in P,$$

kde P je množina všech plánů pohotovostních služeb. Množina P obsahuje všechny plány. Ale ne všechny plány jsou pro nás zajímavé. Zajímat nás budou pouze plány, které splňují nějaké *omezující podmínky.* Pohotovostní služba definuje omezující podmínky, a to konkrétně *maximální počty týmů záchranářů a záchranných vozidel povolených na jednotlivých výjezdových stanicích.*

Definice 2 (Omezující podmínky pohotovostního plánu)**.** *Nechť* $\mathbf{c^z}, \mathbf{c^a} \in \mathbb{N}^{V_n}$ *vektory, kde i-tou položku definujeme:*

$\mathbf{c^z}_i$ *maximální počet záchranných týmů na výjezdové stanici* $v_i \in V$,

$\mathbf{c^a}_i$ *maximální počet záchranných vozidel na výjezdové stanici* $v_i \in V$,

pro $1 \leq i \leq V_n$*. Nechť omezující podmínky* $C = \{C_Z, C_A\}$*, kde* $C_Z \colon P \to \{0, 1\}$*,* $C_A \colon P \to \{0, 1\}$*, definované:*

$$C_Z(p) = 1 \iff |\{z \in Z \mid p_Z(z) = v_i\}| \leq \mathbf{c^z}_i, \qquad \textit{jinak 0},$$
$$C_A(p) = 1 \iff |\{a \in A \mid p_A(a) = v_i\}| \leq \mathbf{c^a}_i, \qquad \textit{jinak 0},$$

$\forall i \colon 1 \leq i \leq V_n$*,* $p \in P$*.*

Plán pohotovostní služby $p \in P$ splňující omezující podmínky C je plán, který splňuje:

$$C_Z(p) = 1 \wedge C_A(p) = 1.$$

V opačném případě p nesplňuje omezující podmínky C. Označme P_C jako *množinu plánů splňujících omezující podmínky* C.

Pohotovostní plán pak bude v průběhu dne odbavovat *incidenty*. *Incident* je trojice:

1. místo nastání,
2. čas nastání,
3. požadovaná maximální doba příjezdu záchranné jednotky.

Budeme vždy uvažovat pouze incidenty, které se odehrávájí v rámci jednoho dne. Tuto množinu označme I.

Zároveň neexistují dva incidenty, které se odehrají v přesně stejný čas. Tento předpoklad je bez újmy na obecnosti, protože kdyby bylo žádoucí, aby se dva incidenty odehráli v přesně stejný čas, tak jednomu z nich stačí čas nastání posunout o vteřinu, což je pro naše účely zanedbatelná změna.

Problém, který v této práci řešíme, je nalézt *optimální plán* $p_C \in P_C \subseteq P$, tedy plán, který bude maximalizovat počet *úspěšně odbavených incidentů* s_I a minimalizovat *celkovou cenu plánu* $u(p)$.

1.2 Simulace plánu pohotovostní služby

Naším cílem je nalézt způsob, který nalezne optimální plán pohotovostní služby. Ten určujeme podle počtu úspěšně odbavených incidentů s_I a ceny plánu $u(p)$. Z toho důvodu jsme si formálně nadefinovali prostředky, které má pohotovostní služba k dispozici a co je plán pohotovostní služby.

V kapitole 1.3 definujeme cenu plánu $u(p)$ (viz definice 10). V této kapitole navrhneme způsob, jakým zjistíme s_I pohotovostního plánu p na množine incidentů I. Pro zjištění s_I spustíme simulaci s pohotovostního plánu p na dané množině I. Následně formálně popíšeme simulaci a určíme, jaká pravidla chodu simulace se rozhodneme použít. Důvod použití přístupu simulace je popsán v nadcházející kapitole.

1.2.1 Přístup použití simulace

Je velmi důležité, aby počet úspěšně odbavených incidentů, které budeme uvažovat při optimalizaci, co nejvíce odpovídal počtu úspěšně odbavených incidentů, kdyby byl plán p použit v reálném světě, a v průběhu dne by se přesně děly incidenty I. Přesně k takovému účelu se používají simulace. Simulace bude napodobovat chování plánu v průběhu dne tak, jak by se plán skutečně choval v reálném světě. Čím realističtěji bude simulace navžena, tím lépe bude počet úspěšně odbavených incidentů simulace odpovídat počtu úspěšně odbavených incidentů v reálném světě.

Přístup použití simulace má ještě jednu podstatnou výhodu. Různé pohotovostní služby mohou používat různé způsoby a pravidla, například pro výběr záchranného týmu či vozidla pro odbavení incidentu, který právě nastal, nebo do jaké nemocnice je vhodné incident odbavit. Tato pravidla mohou být příliš složitá na to, aby bylo možné je výstižně zachytit jinými způsoby, jako například pouze matematickými rovnostmi a nerovnostmi, jak je zvykem pro lineární programování [1].

Na druhou stranu je podstatnou nevýhodou simulace její výpočetní náročnost a značné omezení použitelných technik využívaných pro řešení optimalizačních problémů.

1.2.2 Popis deterministické diskrétní simulace

Simulace je proces navrhnutí modelu reálného systému, a provádění tak na něm experimenty za účelem buď porozumění chování systému nebo vyhodnocení různých strategií chování systému. *Systém simulace* je chápan jako dobře definovaná kolekce objektů a interakcí mezi nimi. Simulace si udržuje *stav systému*. Ten definuje jak se má simulace chovat. Systém simulace se může měnit průběhem simulace nebo při nastání *události*.

Simulace obecně dělíme na *spojité* a *diskrétní*. Ve *spojité simulaci* se změny systému dějí kontinuálně v průběhu běhu simulace, nejčastěji podle soustavy diferenciálních rovnic. V *diskrétní simulaci* se změny sytému dějí v diskrétních časových úsecích, nejčastěji v čase nastání nějaké *události*.

Typický způsob, jakým diskrétní simulace probíhá je následovný. Simulace odbavuje údalosti v pořadí dle času nastání. Při inicializaci si naplánuje nějaké údalosti. Při odbavování události aktualizuje stav systému podle jeho předchozího stavu a aktuálně odbavované události. Zároveň si simulace náplanuje další události. Simulace skončí jakmile nejsou žádné další události k odbavení.

Dále dělíme simulace na *determinstcké* a *stochastické*. V *deterministické simulaci* jsme schopni z aktuálního stavu systému a údalosti deterministicky určit nadcházející stav systému. V *stochastické simulaci* nejsme schopni z aktuálního stavu systému a údalosti deterministicky určit nadcházející stav systému. Většinou proto, že při výběru nadhcázejícího stavu figuruje element náhody [2].

1.2.3 Popis deterministické diskrétní simulace plánu pohotovostní služby

Abychom mohli simulovat fungování pohotovostního plánu, potřebujeme si v první řadě definovat, jaké podmínky musí záchranný tým splňovat, aby mohl

úspěšně odbavit incident.

Definice 3. *Tým záchranářů $z \in Z$ je schopen úspěšně odbavit incident $i \in I$ právě tehdy, když:*

1. *Je alokován a má přiřazenou směnu. Pokud tým záchranáře není alokován, tj. $p_Z(z) = v_\emptyset$, tak samozřejmě není schopen obsloužit i.*

2. *Tým je schopen dorazit na místo incidentu do* požadované doby. *Ať už přímo z výjezdové stanice, nebo při vrácení se po vyřízení incidentu zpět na výjezdovou stanici. V prvním případě tým potřebuje mít na výjezdové stanici k dispozi volné záchranné vozidlo.*

3. *Týmu nekončí směna dříve, než je očekávaný konec celkové doby vyřízení incidentu.*

Nechť $Z_i \subseteq Z$ množina záchranných týmu, které jsou schopny úspěšně odbavit incident $i \in I$. Pro odbavení i musíme vybrat nějaký konkrétní $z_i \in Z_i$.

Je mnoho různých způsobu, jak z_i vybrat. V naší simulaci jsou použitá pravidla vybraná na základě konzultace se společností, která se danou problematikou zabývá přes 25 let a sama podobná pravidla používá pro plánovaní u několika jejích klientů ve Spojených státech amerických [3].

Simulace je navržena dostatečně genericky tak, aby bylo možné naimplementovat i libovolná jiná pravidla, například jednodušší, komplikovanější nebo klidně i stochastická. Jak bylo zmíněno v kapitole 1.2.1, možnost volby libovolných pravidel je jedna z několika výhod přístupu použití simulace.

Definice 4. *Nejvhodnější tým záchranářů $z_i \in Z_i$, kde $Z_i \subseteq Z$ jsou všechny týmy záchranářů schopny úspěšně odbavit incident $i \in I$ je tým, který je nejlepší podle následujících kritérií v daném pořadí:*

1. *Upřednostni tým, který je na výjezdové stanici před týmem, který ještě ukončuje vyřízení jednoho z předchozích incidentů.*

2. *Upřednostni tým, který na místo incidentu dorazí dříve.*

3. *Upřednostni tým, který obsloužil méně incidentů a je tedy méně vyčerpaný.*

Pravidla výběru *nejvhodnějšího týmu záchranářů* jsou navrhnuta tak, aby práce obsluhování incidentů byla rozmístěna rovnoměrně přes všechny týmy, ale zároveň aby byly incidenty obslouženy nejrychlejším možným způsobem.

Pravidla také počítají s možným zpožděním a raději upřednostní tým, který je aktuálně k dispozici, než tým, který ještě dokončuje jiný incident, ale mohl by být i na místě incidentu o něco dříve, pokud by neměl zpoždění. Příkladem je první pravidlo pro výběr nejvhodnějšího týmu.

Jakmile je z_i vybrán, naplánujeme mu incident i a z_i *odbavuje incident* i.

Definice 5 (Záchranný tým odbavuje incident)**.** *Řekneme, že záchranný tým odbavuje incident, pokud vykonává následující činnosti:*

1. *Tým přijíží na místo odehrání incidentu.*
2. *Tým odbavuje incident na místě odehrání incidentu.*
3. *Tým pacienty z incidentu převáží do nejbližší nemocnice.*
4. *Tým pacienty odbavuje v nemocnici.*
5. *Tým přijíždí zpět na výjezdovou stanici.*

Definujme si funkce, které budou v simulaci používány a pomocí výše uvedených pravidel naleznou nejvhodnější tým záchranářů z_i a pro něj nejvhodnější záchranné vozidlo.

Definice 6 (GetBestTeam)**.** *Funkce GetBestTeam vrací tým záchranářu z_i z týmů záchranářů Z_i, kteří jsou schopni úspěšně odbavit aktuální incident $i \in I$ (viz definice 3) , kde z_i je nejvhodnější tým pro obsloužení incidentu i ze Z_i (viz definice 4).*

Definice 7 (GetBestAmbulance)**.** *Funkce GetBestAmbulance vybere záchranné vozidlo, které je na stejné výjezdové stanici jako z_i a z hlediska času je nejdříve k dispozici.*

Nyní jsme připraveni definovat *simulaci pohotovostního plánu* $p \in P_C$ na množině incidentů I, a to jako deterministickou diskrétní simulaci s:

$$s_I = s(p, I).$$

Simulace s vrátí počet úspěšně odbavených incidentů s_I na množině I plánem p.

Událost je nastání incidentu

$$i \in I \text{ v čase } T_I(i),$$

kde $T_I \colon I \to \mathbb{N}_0$, určuje čas nastání incidentu. *Stav systému* simulace s je množina

$$S = \{S_A, S_Z\},$$

kde S_A je stav záchranných vozidel A a S_Z je stav týmů záchranářů Z. Stav S_A je rozmístění záchranných vozidel v prostoru. Stav S_Z je informace, kdy bude záchranný tým k dispozici a pokud tým odbavuje nějaký incident, tak v jaké fázi odbavování zrovna je.

Algoritmus 1 Simulace plánu pohotovostní služby p na množině incidentů I

```
1: function SIMULATION(p, I setřízené podle času nastání T_I)
2:     s_I ← 0
3:     T ←0
4:     S_Z ← Inicializuje podle p
5:     S_A ← Inicializuje podle p
6:     for i_k ∈ I, k ∈ {1, 2, ..., |I|} do
7:         T ← Čas nastání i_k, T_I(i_k)
8:         z_k ← GetBestTeam(i_k, S_Z, S_A, T)
9:         if z_k ≠ ∅ then
10:            s_I ← s_I + 1
11:            a_k ← GetBestAmbulance(z_k, T)
12:            Plan(S_Z, z_k, i_k)
13:            UpdateWhenFree(S_A, z_k, i_k, a_k)
14:        end if
15:    end for
16:    return s_I
17: end function
```

Průběh simulace je následovný. V krocích 2 až 5 simulace (viz algoritmus 1) položí s_I a T rovno nule a incicializuje si stavy S_A a S_Z podle plánu p.

V krocích 6 až 15 odbavuje události, takže se pohybuje po krocích v časech nastání incidentů od nejdřívějšího po nejpozdější. V každém kroku simulace $k \in \{1, 2, \ldots, |I|\}$ se simulace prvně pokusí deterministicky nalézt nejvhodnější $z_k \in Z$, který obslouží i_k, pomocí funkce *GetBestTeam* (viz definice 6), v kroku 8. Pokud takový z_k neexistuje, tak pokračuje v odbavování dalších incidentů.

Pokud existuje, v krocích 7 až 12 zvýší počet odbavených incidentů s_I o jedna, deterministicky nalezne pro z_k nejvhodnější a_k, pomocí funkce *GetBestAmbulance* (viz definice 7) a aktuální incident i_k naplánuje na z_k spolu s a_k funkcí *Plan*. Záchranný tým z_k tak *odbavuje* incident i_k (viz definice 5), a čas, kdy bude opět tým z_k k dispozici je nastaven na čas, kdy incident úspěšně odbaví. Tím se aktualizuje stav záchranných týmů S_Z.

V kroku 13 je funkcí *UpdateWhenFree* aktualizován stav záchranných vozidel S_A. Konkrétně vozidlu a_k se přiřadí čas, kdy bude opět k dispozici, tedy kdy má z_k přijet zpět na výjezdovou stanici.

Simulace doběhne jakmile nejsou žádné další události k odbavení, respektive jakmile projde všechny incidenty, to je po $|I|$ krocích. Simulace vrátí s_I, počet úspěšně odbavených incidentů.

1.3 Převedení problému na optimalizační úlohu s více účelovými funkcemi

V této kapitole si ukážeme, jak konkrétně budeme modelovat optimalizační úlohu nalezení optimálního pohotovostního plánu na množině incidentů I, pomocí počtu úspěšně odbavených incidentů s_I a ceny plánu $u(p)$. Dále si nadefinujeme potřebné pojmy, jako *optimalizační úlohu s více nebo s jednou účelovou funkcí*.

Ukážeme si, že problém nalezení optimálního plánu je vhodné modelovat jako optimalizační úlohu s více účelovými funkcemi, protože chceme maximalizovat počet úspěšně odbavených incidentů a zároveň minimalizovat cenu plánu.

Definice 8 (Optimalizační úloha s jednou účelovou funkcí [4])**.** *Optimalizační úloha s jednou účelovou funkcí je definována jako úloha nalezení* x^**,*

$$x^* = \max_{x \in \mathcal{X}} \{q(x)\}, \tag{1.1}$$

kde $\mathcal{X}$ *je množina všech možných konfigurací a* $q \colon \mathcal{X} \to \mathbb{R}$ *je účelová funkce.*

Návrh účelové funkce zásadně ovlivní, jaké x je řešením. Bez újmy na obecnosti můžeme účelovou funkci maximalizovat, protože

$$\min_{x \in \mathcal{X}} \{q(x)\} \equiv \max_{x \in \mathcal{X}} -\{q(x)\}.$$

Za *optimální řešení optimalizační úlohy s jednou účelovou funkcí* se považuje optimální konfigurace $x^* \in \mathcal{X}$, kde $q(x^*)$ je globální maximum q. Optimální řešení nemusí být jenom jedno, může jich být více. Zároveň alespoň jedno optimální řešení vždy existuje.

Definice 9 (Optimalizační úloha s více účelovými funkcemi)**.** *Optimalizační úloha s více účelovými funkcemi je definována jako úloha nalezení* x^**,*

$$x^* = \max_{x \in \mathcal{X}} \{\mathbf{q}(x)\}, \qquad \mathbf{q}(x) = [q_1(x), q_2(x), \dots q_m(x)], q_i \in \mathcal{Q},$$

pro $1 \leq i \leq m, m = |\mathcal{Q}|$*, kde* $\mathcal{X}$ *je množina všech možných konfigurací a* $\mathcal{Q}$ *je množina všech účelových funkcí, které chceme maximalizovat. Víceúčelovou funkci* $\mathbf{q}(x)$ *maximalizujeme po složkách.*

Z definice 9 je *optimálním řešením optimalizační úlohy s více účelovými funkcemi* $x^* \in \mathcal{X}$ takzvaný *utopia point* y^{utopia} [4]. To je konfigurace, která nabývá optima pro každou jednotlivou účelovou funkci. Zřejmě y^{utopia} nemusí existovat, protože často maximalizování $q_i \in \mathcal{Q}$ může minimalizovat $q_j \in \mathcal{Q}, i \neq j$. V nadcházející kapitole si ukážeme jak se optimalizace takového typu řeší.

V předchozí kapitole jsme si ukázali způsob, jakým nalezneme počet úspěšně odbavených incidentů s. Abychom mohli problém nalezení optimálního plánu modelovat jako optimalizační úlohu s více účelovými funkcemi, potřebujeme si ještě dodefinovat cenu plánu.

Definice 10 (Cena plánu u)**.** *Cena plánu* $u \colon P \to \mathbb{N}$*,*

$$u(p) = \sum_{z \in Z} p_{D_l}(z) + |\{a \in A \mid p_A(a) \neq v_\emptyset\}|.$$

Cena plánu je součet všech dob trvání směn přiřazených záchranným týmům a počtu naalokovaných záchranných vozidel.

Všimněme si, že $p_{D_l}(z) = 0$, pro nenaalokované týmy (viz kapitola 1.1), čili nijak nepřispějí do výsledku sumy.

Nyní již máme všechny prostředky potřebné k tomu, abychom modelovali problém nalezení optimálního plánu jako optimalizační úlohu s více účelovými funkcemi.

Definice 11 (Nalezení optimálního plánu pohotovostní služby jako optimalizační úloha s více účelovými funkcemi)**.** *Nalezení optimálního plánu pohotovostní služby jako optimalizační úloha s více účelovými funkcemi je úloha nalezení* p^**,*

$$p^* = \max_{p \in P_C} \{\mathbf{q}(p)\}, \qquad \mathbf{q}(p) = [s(p, I), -u(p)],$$

kde P_C *je množina všech povolených plánu pohotovostní služby splňující omezení* C *a* I *je daná množina incidentů. Účelové funkce* s *a* u *jsou simulace plánu* p *na* I *a cena plánu* p*.*

Všimněmě si, že maximalizování s bude velmi pravděpodobně vést k maximalizování u, tedy k minimalizování $-u$. Plány úspěšně odbavující incidenty zřejmě budou používát více záchranných týmu s delšími směnami a více záchranných vozidel, takže budou dražší. Z toho důvodu řešení y^{utopia} nemusí existovat. Tímto problémem se zabýváme v následující kapitole.

1.4 Metody pro řešení optimalizačního problému s více účelovými funkcemi

V předchozí kapitole jsme problém nalezení optimálního plánu pohotovostní služby namodelovali jako optimalizační úlohu s více účelovými funkcemi (viz definice 11), a to počtem úspěšně odbavených incidentů $s(p, I)$ a ceny plánu $u(p)$. Zároveň jsme pozorovali, že maximalizováním s minimalizujeme $-u$ a naopak.

Optimalizační úlohy s více účelovými funkcemi, mezi kterými je potřeba nalézt kompromis pro nalezení optima, jsou velmi časté, a proto existuje několik metod, které optimalizační úlohy s více účelovými funkcemi řeší. Jednou z účinných metod je převod víceúčelové funkce na jednu účelovou funkci, aby zároveň maximalizování víceúčelové funkce bylo stejné jako maximalizování jedné účelové funkce.

Řešení optimalizační úlohy s jednou účelovou funkcí je dobře definované, narozdíl od řešení optimalizační úlohy s více účelovými funkcemi. V této kapitole si ukážeme vybrané metody.

1.4.1 Lexikografické porovnání

Jedná se o způsob převedení víceúčelové funkce pouze na jednu účelovou funkci, kde konfigurace porovnáváme lexikograficky podle permutace jednotlivých účelových funkcí, která určuje jejich prioritu nad ostatními.

Definice 12 (lexikografické porovnání)**.** *Účelová funkce* q *porovnávající* $x_1, x_2 \in \mathcal{X}$ *lexikograficky podle permutace* $\pi \in S_m$ *je definovaná:*

$$q(x_1) < q(x_2) \Leftrightarrow \bigvee_{k=1}^{m} \left(\left(\bigwedge_{i=1}^{k-1} q_{\pi(i)}(x_1) = q_{\pi(i)}(x_2) \right) \wedge q_{\pi(k)}(x_1) < q_{\pi(k)}(x_2) \right)$$

a

$$q(x_1) = q(x_2) \Leftrightarrow q_{\pi(i)}(x_1) = q_{\pi(i)}(x_2), \forall i \in \{1, \ldots, m\}.$$

Jedná se o optimalizaci po jednotlivých účelových funkcích podle určené priority. Obecně není vhodná, protože neumí nalézt kompromis mezi jednotlivými účelovými funkcemi a často nemusí být jasné, jaké účelové funkce chceme preferovat.

1.4.2 Vážená suma účelových funkcí

Jedná se o způsob převedení víceúčelové funkce pouze na jednu účelovou funkci pomocí sumy účelových funkcí pronásobené jejich přidělenými váhami.

Definice 13 (Vážená suma účelových funkcí [4])**.** *Vážená suma účelových funkcí q' je definovaná jako*

$$q'(x) = w^T \mathbf{q}(x),$$

kde $w \in \mathbb{R}^m$ je vektor vah.

Tato metoda už umí nalézt kompromis mezi jednotlivými účelovými funkcemi, a to podle vah w. Stejně ale jako u lexikografického porovnání nemusí být obecně jasné, jaké účelové funkce preferovat.

1.4.3 Goal programming

Jedná se o způsob převedení víceúčelové funkce pouze na jednu účelovou funkci minimalizováním vzdálenosti od nějaké ideální hodnoty – cíle.

Definice 14 (Účelová funkce měřící vzdálenost od cíle [4])**.**

$$\min_{x \in \mathcal{X}} \{\|\mathbf{q}(x) - y^{goal}\|\}, \qquad y^{goal} \in \mathbb{R}.$$

Standardně $y^{goal} = \mathbf{q}(y^{utopia})$. Zvolením vhodné normy můžeme preferovat námi vybrané účelové funkce, ale obdodně jako u předchozích metod, obecně nemusí být jasné, jaké účelové funkce preferovat. Zvolením normy jako nějaké p-normy (například standardní euklidovské normy), budou účelové funkce implicitně preferovány podle toho, v jakých jednotkách se měří. Tudiž při maximalizaci budou preferovány účelové funkce s oborem hodnot ve větším rozsahu.

1.4.4 Vážená exponenciální suma účelových funkcí

Jedná se o kombinaci metody goal programming a metody vážené sumy účelových funkcí. Váhami w jsme schopni určit, kterou účelovou funkci je pro nás důležitější optimalizovat a parametrem φ jakou φ-normu chceme použít.

Definice 15 (Vážená exponenciální suma účelových funkcí [4])**.**

$$q(x) = \sum_{i=1}^{m} w_i (q_i(x) - y_i^{goal})^{\varphi},$$

kde $w \in \mathbb{R}^m, \varphi \in \mathbb{R}$.

Může být příjemnější na použití než goal programming, protože místo škálování prostoru normou škálujeme pomocí jednotlivých hodnot účelové funkce.

1.5 Převedení problému na optimalizační úlohu s jednou účelovou funkcí

Nadefinovali jsme účelové funkce s a u a problém nalezení optimálního plánu jako optimalizační úlohu s více účelovými funkcemi s a u. Pro optimalizační úlohy s více účelovými funkcemi obecně nemusí existovat optimum. Proto jsme si v předchozí kapitole ukázali několik přístupů, které vhodně převedou optimalizační úlohu s více účelovými funkcemi na optimalizační úlohu s jednou účelovou funkcí.

V této kapitole si ukážeme, jak optimalizační úlohu nalezení optimálního plánu pohotovostní služby s více účelovými funkcemi (viz definice 11) převést na optimalizační úlohu nalezení optimálního plánu pohotovostní služby s jednou účelovou funkcí, pomocí metod zmíněných v předchozí kapitole.

K tomu si je ještě potřeba prvně dodefinovat přeškálované účelové funkce do intervalu $\langle 0, 1\rangle$, aby bylo možné výsledné účelové funkce definovat přehledněji.

Definice 16 (Přeškálování s do intervalu $\langle 0, 1\rangle$)**.** *Definujme s' jako přeškálování s do intervalu $\langle 0, 1\rangle$,*

$$s'(p, I) = s(p, I)/|I|.$$

Definice 17 (Přeškálování u do intervalu $\langle 0, 1\rangle$)**.** *Definujme u' jako přeškálování u do intervalu $\langle 0, 1\rangle$,*

$$u'(p) = u(p)/K,$$

kde K je maximální možná cena plánu,

$$K = Z_n \cdot \max_{d_l}\{d_s\} + A_n,$$

naalokování nejdelší směny na všechny týmy záchranářů a naalokování všech záchranných vozidel.

Nyní už si můžeme ukázat jak vypadají jednotlivé účelové funkce optimalizace pohotovostního plánu. Začneme aplikováním lexikografického porovnání (viz definice 18) na náš problém, a to následovně:

Definice 18 (Lexikografické porovnávání optimalizace pohotovostního plánu)**.** *Lexikografické porovnávání optimalizace pohotovostního plánu q^{Lex} definujeme jako*

$$\begin{aligned} &q^{Lex}(p_1) < q^{Lex}(p_2) \Leftrightarrow \\ &(s(p_1, I) < s(p_2, I)) \vee (s(p_1, I) = s(p_2, I) \wedge u(p_1) < u(p_2)) \end{aligned}$$

a

$$q^{Lex}(p_1) = q^{Lex}(p_2) \Leftrightarrow s(p_1, I) = s(p_2, I) \wedge u(p_1) = u(p_2),$$

pro danou množinu incidentů I.

Pokud za účelovou funkci zvolíme q^{Lex}, tak optimální plán bude takový, že odbaví co nejvíce incidentů je možné, a z nich budou optimální ty nejlevnější plány:

$$\min_{p^* \in P}\{u(p^*)\}, \quad P = \{p \mid \max_{p \in P_C}\{s(p, I)\}\},$$

pro danou množinu incidentů I.

Definovat q^{Lex} obráceně nedává smysl, protože nejlevnější plán je prázdný plán a ten na množině incidentů I nikdy neodbaví žádný incident.

Váženou sumu účelových funkcí (viz definice 13) aplikujeme na náš problém podle následující definice.

Definice 19 (Vážená suma účelových funkcí optimalizace pohotovostního plánu)**.** *Váženou sumu účelových funkcí optimalizace pohotovostního plánu q_α definujeme jako*

$$q_\alpha(p) = \alpha \cdot s'(p, I) - (1 - \alpha) \cdot u'(p), \qquad \alpha \in [0, 1], p \in P_C,$$

pro danou množinu incidentů I, kde $w_1 = \alpha$ a $w_2 = 1 - \alpha$.

Parametr α bychom preferovali blíže jedné, pro upřednostnění s' a pro upřednostnění u' blíže nule.

Metodu goal programming (viz definice 14) aplikujeme následovně.

Definice 20 (Goal programming optimalizace pohotovostního plánu)**.** *Víceúčelovou funkci převedeme na jednu podle metody goal programming na účelovou funkci q^{Goal} definovanou jako*

$$q^{Goal}(p) = \|[1 - s'(p, I), u'(p)]\|,$$

pro danou množinu incidentů I a libovolnou normu.

Pro pohotovostní plány nechť $p^{goal} = q(p^{utopia}) = [1, 0]$, úspěšné odbavení všech incidentů za nulovou cenu. Samozřejmě $p^{utopia} \notin P_C$, ledaže $|I| = 0$.

Nulovou cenu má pouze plán, kde není naalokován žádný tým záchranářů ani záchranné vozidlo a takový plán nemůže úspěšně obsloužit žádný incident. Můžeme však měřit, jak blízko $q(p)$ k $q(p^{utopia})$ je.

Jako poslední aplikujeme na náš problém metodu vážené exponenciální sumy účelových funkcí (viz definice 15).

Definice 21 (Vážená exponenciální suma účelových funkcí optimalizace pohotovostního plánů)**.** *Váženou exponenciální sumou účelových funkcí optimalizace pohotovostního plánů q_α^φ definujeme jako*

$$q_\alpha^\varphi(p) = \alpha(1 - s'(p, I))^\varphi + (1 - \alpha)(u'(p))^\varphi, \qquad p \in P_C,$$

pro danou množinu incidentů I, kde $w_1 = \alpha$ a $w_2 = 1 - \alpha$.

Stejně jako u vážené sumy účelových funkcí jsme díky parametru α schopni upřednostit maximalizování s' nad $-u'$.

Nyní máme prostředky pro definování problému nalezení optimálního pohotovostního plánu jako optimalizační úlohy s jednou účelovou funkcí.

Označme si $Q_I = \{q^{\text{Lex}}, q^{\text{Goal}}, q_\alpha, q_\alpha^\varphi\}$ jako množinu účelových funkcí na dané množině incidentů I.

Definice 22 (Problém nalezení optimálního plánu jako optimalizační úloha s jednou účelovou funkcí)**.** *Optimalizační úloha nalezení optimálního plánu pohotovostní služby s jednou účelovou funkcí je definována jako úloha nalezení p^*,*

$$p^* = \max_{p \in P_C} \{q(p)\}, \tag{1.2}$$

kde $q \in Q_I$.

1.6 Analýza optimalizační úlohy

V předchozí kapitole se nám podařilo problém nalezení optimálního plánu namodelovat jako optimalizační úlohu s jednou účelovou funkcí, pomocí metod diskutovaných v kapitole 1.4. V této kapitole klasifikujeme, o jakou optimalizační úlohu se jedná, a to zkoumáním účelové funkce a množiny plánů. Zjištěním přesně o jaký typ optimalizační úlohy se jedná, budeme schopni použít metody, které jsou vhodné pro řešení úloh takového typu. Podrobněji jsou tyto metody diskutovány v kapitole 2.

1.6.1 Analýza množiny plánů pohotovostních služeb

Znát velikost množiny plánů pohotovostních služeb P a P_C je klíčové pro navrhování metod hledající optima, a proto je v této kapitole spočítáme. To platí zejména pro P_C, jelikož námi hledané optimum právě P_C náleží. Jaké implikace má velikost P_C na zvolené metody řešení je popsáno v kapitole 1.6.4.

Věta 1 (Velikost množiny plánů pohotovostních služeb)**.** *Velikost plánu pohotovostní služby P je rovna:*

$$\sum_{i=0}^{Z_n} D_n^{\binom{Z_n-i+V_n-1}{V_n-1}} \cdot \sum_{i=0}^{A_n} \binom{A_n - i + V_n - 1}{V_n - 1}.$$

Důkaz. Všimněme si, že počet uspořádaných V_n-tic nezáporných celých čísel posčítajících se na k, přičemž záleží na pořadí sčítanců, přesně odpovídá počtu naalokování záchranných týmů na výjezdové stanice, kde chceme naalokovat přesně k týmů. Například pro $k = 10$ a $V_n = 4$ by

$$3 + 0 + 5 + 2 = 10$$

odpovídalo naalokování 3 týmů na první stanici, 0 na druhou, 5 na třetí a dva na čtvrtou. Podle lemma o počtu uspořádaných r-tic nezáporných celých čísel, které se posčítají na m, přičemž záleží na pořadí (Jiří Matoušek [5], (2.3), str. 61), víme, že takových uspořádaných V_n-tic je

$$\binom{k + V_n - 1}{V_n - 1}. \tag{1.3}$$

Přesčítáním přes všechny $k \in \{0, \ldots, Z_n\}$ tak spočítáme sumu všech naalokování přes k týmů záchranářů, které jsou navzájem disjunktní, takže žádnou alokaci týmů nesčítáme vícekrát. Každému týmu záchranářů v rámci alokace je ještě přiřazena pracovní směna, těch je D_n, proto D_n umocňujeme.

Pro alokování záchranných týmů postupujeme analogicky, akorát nepřiřazujeme pracovní směny. □

Věta 2 (Velikost množiny plánů pohotovostních služeb splňující omezení C)**.** *Velikost plánu pohotovostní služby P_C splňující omezení C je rovna:*

$$\sum_{i=0}^{Z_n^c} D_n^{\binom{Z_n^c-i+V_n-1}{V_n-1}} \cdot \sum_{i=0}^{A_n^c} \binom{A_n^c - i + V_n - 1}{V_n - 1},$$

kde

$$Z_n^c = \sum_{i=1}^{V_n} c_{z_i}, A_n^c = \sum_{i=1}^{V_n} a_{z_i}.$$

Důkaz. Stejný jako v předchozí větě, avšak namísto toho, abychom vybírali ze všech záchranných týmu a vozidel, vybíráme ze součtu přes korespondující dostupné kapacity na výjezdových stanicích Z_n^c a A_n^c. □

Věta 3 (Asymptotický odhad velikosti množiny plánů pohotovostních služeb)**.** *Nechť P množina plánu pohotovostních služeb. Pak*

$$|P| \in \mathcal{O}\left({D_n}^{2^{Z_n+V_n}} \cdot 2^{A_n+V_n}\right).$$

Důkaz. Odhadněme s pomocí identity kombinačního čísla (Jiří Matoušek [5], (2.4), str. 62) a binomické věty (Jiří Matoušek [5], (2.6), str. 63),

$$\sum_{i=0}^{n} \binom{n-i+V_n-1}{V_n-1} = \sum_{i=0}^{n} \binom{n-i+V_n-1}{n-i} \leq \sum_{i=0}^{n} \binom{n+V_n-1}{n-i} = 2^{n+V_n-1}$$

a dosaďmě do věty 1 pro $n = Z_n$, A_n,

$$|P| \leq {D_n}^{2^{Z_n+V_n-1}} \cdot 2^{A_n+V_n-1},$$

takže,

$$|P| \in \Theta({D_n}^{2^{Z_n+V_n}} \cdot 2^{A_n+V_n}).$$

Odhad má chybu nanejvýš n^2. □

Věta 4 (Asymptotický odhad velikosti množiny plánů pohotovostních služeb splňující omezující podmínky)**.** *Nechť P_C množina plánu pohotovostních služeb splňující omezující podmínky C. Pak*

$$|P_C| \in \Theta({D_n}^{2^{Z_n^c+V_n}} \cdot 2^{A_n^c+V_n}).$$

Důkaz. Analogicky jako u důkazu věty 3, dosadíme $n = Z_n^c, A_n^c$. □

1.6.2 Analýza účelové funkce

V této kapitole se zaměříme na vlastnosti účelové funkce. Nejdříve si zanalyzujeme vlastnosti simulaci s, z čehož nám vyplynou vlastnosti o účelové funkci, jelikož účelová funkce je jejím složením s cenou. Jaké implikace mají vlastnosti účelové funkce na zvolené metody řešení je diskutováno v kapitole 2.

Věta 5 (Vlastnosti simulace s)**.** *Simulace s není spojitá a derivovatelná.*

Důkaz. Definiční obor přiřazujících funkcí p_Z, p_A, p_{D_1}, p_{D_2}, jsou množiny Z, A, D_s, D_l, obsahující konečný počet objektů. Plán p je jednoznačně určen přiřazujícími funkcemi, takže i P je konečná množina. Definiční obor s je množina plánů $p \in P_C$, čili definiční obor funkce s je konečná množina, a tak s nemůže být spojitá funkce. Jelikož s není spojitá, nemůže být ani derivovatelná.

Argumentů pro nederivovatelnost je více, například s nemá žádný matematický předpis, nebo také že P_C je diskrétní množina. □

Věta 6 (Vlastnosti účelových funkcí Q_I)**.** *Účelové funkce $q \in Q_I$ nejsou spojité a derivovatelné.*

Důkaz. Víceúčelové funkce $q \in Q_I$ které budeme maximalizovat jsou jen jednoduchou kombinací původních účelových funkcí s a u, podle definice 22. Z věty 5 víme, že s není spojitá ani derivovatelná. Tím pádem i $q \in Q_I$ není spojitá ani není derivovatelná. □

1.6.3 Naivní řešení

V této kapitole se podíváme na přímočarý způsob, jak nalézt optimální plán pohotovostní služby. Jednoduše zkusíme účelovou funkci $q \in Q_I$ vyhodnotit ve všech bodech, tedy všech plánech $p \in P_C$. Ukáže se, že takové přímočaré naivní řešení je velmi neefektivní. Sofistikovanější způsoby řešení našeho problému jsou diskutovány v kapitole 2.

Definice 23 (Naivní řešení)**.** *Naivní řešení problému nalezení optimálního plánu pohotovostní služby znamená vyhodnocení $q \in Q_I$ ve všech bodech $p \in P_C$, kde si v průběhu vyhodnocování držíme $p^* \in P_C$ doposud s maximální hodnotou $q(p^*)$.*

Po vyhodnocení ve všech bodech p, tak p^* bude optimem, protože $q(p^*) \geq p, \forall p \in P_C$.

Věta 7 (Složitost naivního řešení)**.** *Složitost naivního řešení je*

$$\Theta((A_n + Z_n|I|) \cdot {D_n}^{2^{Z_n^c + V_n}} \cdot 2^{A_n^c + V_n}),$$

kde $\Theta(A_n + Z_n|I|)$ je složitost vyhodnocení $q \in Q_I$.

Důkaz. Z definice naivního řešení 23 potřebujeme vyhodnotit q ve všech bodech $p \in P_C$. Účelová funkce q je konstantní kombinací funkcí simulace s a ceny plánu u. Složitost vyhodnocení u je

$$\Theta(Z_n + A_n).$$

To plyne přímo z definice u 10. Složitost vyhodnocení s je

$$\Theta(Z_n \cdot |I|).$$

Z definice s 1.2.3 nahlédneme, že q potřebujeme vyhodnotit pro každý incident, těch je $|I|$, nalézt nejvhodnější tým záchranářů. Nejvhodnější tým záchranářů

nalezneme pro $i \in I$ v čase Z_n, protože potřebujeme všechny týmy proiterovat a práci na jeden tým už považujeme za konstantní.

Vyhodnocení účelové funkce q má pak složitost

$$\Theta(Z_n + A_n + Z_n|I|) = \Theta(A_n + Z_n|I|).$$

Potřebujeme vyhodnotit q pro každý plán $p \in P_C$. Asymptotický odhad velikosti množiny P_C je

$$\Theta({D_n}^{2^{Z_n^c+V_n}} \cdot 2^{A_n^c+V_n}),$$

z věty 3. Z toho již plyne složitost naivního řešení z věty. □

Naivní řešení běží v exponenciálním čase, protože prostor plánů P_C je exponenciálně velký, takže už i samotný průchod běží v exponenciálním čase. Chtěli bychom nalézt způsob, kterým nalezneme optimum ideálně v polynomiálním čase, nebo alespoň ne v exponenciálním čase.

1.6.4 Klasifikace optimalizační úlohy

V kapitole 1.6.1 jsme přesně spočítali velikost P a P_C a zároveň jsme je odhadli asymptoticky. V kapitole 1.6.2 jsme si ukázali základní vlastnosti účelové funkce $q \in Q_I$. V předchozí kapitole jsme si ukázali, že nemůžeme úlohu vyřešit jednoduše naivním řešením v polynomiálním čase. V této kapitole dáme tyto poznatky dohromady, a zklasifikujeme tak řešenou optimalizační úlohu, za účelem nalezení nejvhodnějších metod pro řešení.

Vybíráme optimální konfigurace z konečně mnoho diskrétních objektů, a to sice přiřazení p_Z, p_A, p_{D_s}, p_{D_l}. Jedná se tak o *diskrétní* nebo taky o *kombinatorickou* optimalizaci [6]. Jedná se o takový druh optimalizace, u kterého se snažíme nalézt optimální konfiguraci z diskrétně mnoho možností.

Spojitá optimalizace je typ optimalizace, kde prohledávané konfigurace jsou spojité [7].

Z věty 6 víme, že $q \in Q_I$ není spojitá ani derivovatelná, takže metodami pro řešení spojitých optimalizačních úloh se nebudeme zabývat.

Podívejme se na úspěšnou metodu nalezení optima v kombinatorické optimalizci, sice lineární programování [1]. Problém formulovaný jako lineární program lze vyřešit v polynomiálním čase [8].

Definice 24 (Formulace problému pro lineární programování ([4], str. 189))**.**

$$\max_{x \in \mathcal{X}} c^T x,$$

kde x splňuje

$$\begin{aligned} {w^1}_i^T x &\leq b_i, \qquad i \in \{1, 2, \dots\}, \\ {w^2}_j^T x &\geq b_j, \qquad j \in \{1, 2, \dots\}, \\ {w^3}_k^T x &= b_k, \qquad k \in \{1, 2, \dots\}, \end{aligned}$$

kde c je vektor, reprezentující lineární účelovou funkci.

Náš problém ale nelze vyjádřit v takovém tvaru, především z důvodu navrhnutí účelové funkce. Účelová funkce $q \in Q_I$ je složením se simulací s. Simulace s nemá žádný matematický předpis. Použití simulace tak nevylučuje jenom lineární programování, ale obecně všechny metody, ve kterých je potřeba optimalizační úlohu vyjádřit jako soustavu rovnic nebo nerovnic, které ani nemusí být lineární. Použití simulace je pro nás ale žádoucí, a to hned z několika důvodů, které jsou diskutovány v kapitole 1.2.1.

Dále bychom chtěli pro problém nalezení optimálního plánu zjistit, do jaké třídy složitosti náleží. Nejdříve si ukažme, do jaké třídy složitosti náleží optimalizační úloha, na jejíž účelovou funkci dáme daná omezení.

Definice 25 (Optimalizační úloha s černou skříňkou Q)**.** *Nechť $\mathcal{X}\colon |\mathcal{X}| \in \Theta(2^n)$ množina všech možných konfigurací a $q\colon X \to \{1, 2, \ldots, n\}, n \in \mathbb{N}$. Jako optimalizační úlohu s černou skříňkou rozumíme optimalizační úlohu*

$$\max_x q(x), \qquad x \in \mathcal{X},$$

kde $\forall x_1, x_2 \in \mathcal{X}$ jsme schopni zjistit zda platí $q(x_1) \leq q(x_2)$ jedině vyhodnocením q v x_1, q v x_2 a následným porovnáním vyhodnocených hodnot, $q(x_1) \leq q(x_2)$. Vyhodnocení q může být nejhůř polynomiální vůči oboru hodnot q. Takové funkci q budeme říkat černá skříňka.

Označme problém optimalizační úlohy s černou skříňkou Q. Odpověď $Q(y)$ je rovna optimu x^*, pro libovolný vstup problému y.

Definice 26 (Rozhodovací problém černé skříňky R)**.** *Nechť $\mathcal{X}\colon |\mathcal{X}| \in \Theta(2^n)$ množina všech možných konfigurací a $q\colon X \to \{1, 2, \ldots, n\}, n \in \mathbb{N}$, kde q je černá skříňka. Jako rozhodovací problém černé skříňky rozumíme otázku, zda*

$$\exists x \in \mathcal{X}\colon q(x) = k, \qquad x \in \mathcal{X},$$

kde $k \in \{1, 2, \ldots, n\}$.

Označme rozhodovací problém černé skříňky R. Pokud existuje, $R(y) = 1$, jinak $R(y) = 0$, pro libovolný vstup problému y.

Věta 8 (Rozhodovací problém černé skříňky je NP-úplný)**.** *Problém R je NP-úplný.*

Důkaz. Zaprvé dokážeme, že problém R náleží třídě NP. Řekněme, že nějaký algoritmus A tvrdí, že $x \in \mathcal{X}$ je řešením problému R. Vyhodnocením q v x ověříme, jestli $q(x) = k$, tedy $A(x) = R(x)$. Vyhodnocení q trvá nejhůře polynomiálně dlouho vůči n, takže R náleží NP.

Zadruhé ukážeme převod problému SAT na R: $SAT \to R$. Problém SAT je NP-úplný. Převodem tak dokážeme, že problém R je alespoň tak těžký, jako problém SAT, takže je alespoň NP-úplný.

Nechť klauzule v konjuktivním normálním tvaru ϕ problému SAT o n literálech. Nechť $\mathcal{X}$ je množina všech možných ohodnocení ϕ.

Funkci q definujme $q(x) = \phi(x)$. Hledané k tak bude 1, splění formule. Takto definovaná funkce q je černá skříňka, protože pro libovolné ohodnocení x_1,$x_2 \in \mathcal{X}$ musíme ϕ vyhodnotit, abychom zjistili ohodnocení ϕ. Nejsme schopni na základě $q(x_1)$ určit ohodnocení $q(x_2)$, pro libovolné ohodnocení $x_1, x_2 \in \mathcal{X}$, tedy ani $q(x_1) \leq q(x_2)$.

Takže $x \in \mathcal{X}$ takové, že $q(x) = \phi(x) = 1$, je hledané splnitelné ohodnocení formule ϕ a sestrojili jsme tak převod mezi problémem *SAT* a problémem R.

Ukázali jsme, že problém R náleží třídě NP a zároveň jsme sestrojili převod NP-úplného problému na problém R. Jelikož jsou všechny NP-úplné problémy navzájem převoditelné, tak i problém R je NP-úplný. □

Věta 9 (Optimalizační úloha s černou skříňkou je NP-těžký problém)**.** *Problém Q je NP-těžký problém.*

Důkaz. Víme, že problém R je NP-úplný (viz věta 8). Nyní dokážeme, že problém Q je NP-těžký. Prvně sestrojíme převod mezi problémy R a Q: $R \to Q$, a tím ukážeme, že Q je alespoň tak těžký jako R. Zadruhé ukážeme, že problém Q není NP-úplný. Z obojího plyne, že problém Q je NP-těžký, podle definice NP-těžkosti.

Nechť rozhodovací problém černé skříňky R, kde nás zajímá, zda existuje x pro které $q(x) = k, k \in \{1, \ldots, n\}$. Nechť optimalizační problém s černou skříňkou Q, kde z definice hledáme optimum x^* černé skříňky q'. Definujme $q'(x) =$

$$\begin{cases} q(x), & \text{je-li } q(x) \leq k, \\ k-1, & \text{jinak.} \end{cases}$$

Pokud $q'(x^*) = k$, pak existuje x takové, že $q(x) = k$, a to například právě optimum x^*. Pokud $q'(x^*) = l, l \neq k$, pak neexistuje x takové, že $q(x) = k$, protože $l < k$ a zároveň x^* je optimální. Takže pro nalezené optimum x^* platí $q'(x^*) = k$ právě tehdy, když existuje $x \in \mathcal{X}$ takové, že $q(x) = k$. Tím jsme sestrojili převod $R \to Q$.

Zadruhé ukažme, že problém Q není NP-úplný. Nechť algoritmus A řešící problém Q. Nechť x^* optimální podle A. Jak ověříme, že x^* je skutečně optimální, tedy $Q(x) = x^*$? Z definice nesmí existovat $x' \colon q(x') > q(x^*)$. Abychom ověřili, že takové x' skutečně neexistuje, musíme projít všechna $x \in \mathcal{X}$ a ověřit, že $q(x) \leq q(x^*)$. Jiným způsobem, než průchodem $\mathcal{X}$ to zjistit nemůžeme, protože q je černá skříňka. Jenže $|\mathcal{X}| = 2^n$, průchod trvá 2^n, čili hůře než polynomiálně vůči n.

Na optimalizační problém s černou skříňkou Q jsme převedli NP-úplný rozhodovací problém černé skříňky R a zároveň problém Q není NP-úplný, takže problém Q je NP-těžký. □

Pojďme si propojit optimalizační problém s černou skříňkou (viz definice 25) s našim problémem (viz definice 22).

Optimalizační úloha nalezení optimálního plánu pohotovostní služby používá účelovou funkci $q \in Q_I$. Účelová funkce q je složením se simulací s. Její předpis je dán průběhem simulace a vyhodnocení $s(p, I)$ závisí na průběhu simulace pohotovostního plánu na množině incidentů. Simulace je velmi komplikovaná, a přestože v podstatě známe její předpis (viz algoritmus 1), tak nalézt nějaké vztahy, díky kterým budeme schopni ze znalosti $s(p_1)$ zjistit $s(p_1) \leq s(p_2)$ pro $\forall p_1, p_2 \in P_C \colon p_1 \neq p_2$ není vůbec snadné. Dává tak smysl na simulaci nahlížet jako na černou skříňku.

Věta 10 (Simulace jako černá skříňka)**.** *Nechť optimalizační úloha plánu pohotovostní služby O používající účelovou funkci q, kde q je složením se simulací s. Pokud předpokládáme, že s je černá skříňka, pak O je NP-těžká úloha.*

Důkaz. Na úlohu O se můžeme dívat jako na konkrétní instanci problému Q. Účelovou funkci Q zvolíme jako $q \in Q_I$, využívající simulaci, množinu $\mathcal{X}$ zvolíme jako P_C. Z věty 4 je velikost P_C exponenciálně závislá na oboru hodnot q a zároveň z předpokladu je černou skříňkou. Podle věty 9 je problém Q NP-těžký, úlohu O jsme zformuvali jako problém Q, takže i O je NP-těžké. □

Důsledek věty je, že pokud bychom chtěli úlohu O vyřešit v polynomiálním čase, tak na simulaci nesmíme nahlížet jako na černou skříňku. Budeme minimálně potřebovat umět nalézt nějaké vztahy mezi $p_1, p_2 \in P_C$, díky kterým budeme schopni zjistit $q(p_1) \leq q(p_2)$ a najít pomocí těchto vztahů optimální plán, aniž bychom museli q vyhodnotit přes celý její definiční obor. Jedná se ale pouze o nutnou podmínku, nikoliv postačující. Čili pokud nalezneme nějaké vztahy mezi p_1, p_2 a nebudeme na simulaci nahlížet jako na černou skříňku, neznamená to, že je nutně úloha optimálních plánů pohotovostních služeb řešitelná v polynomiálním čase. O nalezení takových vztahů se pokusíme v kapitole 2.1. Metody, které na účelovou funkci pohlíží jako na černou skříňku zkoumáme v kapitole 2.2.

2 Řešení optimalizační úlohy

V této kapitole vyzkoušíme různé metody, které můžeme pro nalezení optimálního plánu pohotovostní služby použít.

2.1 Dynamické programování

Dynamické programování je technika řešení problému, která si průběžně ukládá řešení menších podúloh a pomocí rekurzivního vztahu menších podúloh s většími, definovaného *rekurzivním vztahem*, řeší větší podúlohy efektivněji, až po vyřešení původní úlohy. Průběžnému ukládání výsledků podúloh se říká *memoizace*. Díky této memoizaci dynamické programování neprohledává prostor řešení duplicitně, a tak je často velmi efektivní metodou pro řešení optimalizačních úloh [9].

Triviálním příkladem využití dynamického programování je výpočet n-tého *Fibbonaciho čísla* [10]. Pro nás zajímavějším příkladem využití dynamického programování je řešení úlohy kombinatorické optimalizace *Problému batohu*.

Definice 27 (Problém batohu)**.** *Nechť n počet předmětů, které chceme vložit do batohu s kapacitou c. Každý předmět i má výdělek p_i a váhu w_i. Problém batohu pak je,*

$$\begin{aligned} &\text{maximalizování} && z = \sum_{i=1}^{n} p_i x_i, \\ &\text{splňující} && \sum_{i=1}^{n} w_i x_i \leq c, \end{aligned}$$

kde $x_i = 1$, pokud předmět i je v batohu, jinak 0.

Existuje 2^n možností, kterými předměty vložíme do batohu. Naivní řešení prohledání všech možností tak běží v čase $\mathcal{O}(2^n)$. Avšak pomocí dynamického programování lze vyřešit problém batohu v *pseudopolynomiálním* čase $\mathcal{O}(nc)$, což pro velká n a konstatní c je až exponenciálním zlepšením.

Rekurzivní vztahy vypadají následovně,

Definice 28 (Rekurzivní vztah pro problém batohu)**.**

$$m_{i,c'} = m_{i-1,c'} \text{ pokud } w_i > c', \tag{2.1}$$
$$m_{i,c'} = \max(m_{i-1,c'}, m_{i-1,c'-w_i} + p_i) \text{ pokud } w_i \leq c', \tag{2.2}$$

pro $0 \leq c' \leq c$ aktuální uvažovaná kapacita batohu.

Rekurzivní vztah (viz rovnice 28) nám pouze říká, že ze znalostí optimálního výběru předmětů podpbroblému $m_{i-1,c'-w_i}$ a podproblému $m_{i-1,c'}$, umíme v konstantím čase zjistit optimální výběr předmětů pro aktuální problém c', tedy $m_{i,c'}$. Buď předmět i použijeme při výběru, a tak aktuální optimální výběr je roven

Algoritmus 2 Problém batohu

```
1: function KnapsackProblem(n, c, p_i, w_i, 1 ≤ i ≤ n)
2:     m_{i,j} ←0, 0 ≤ i ≤ n, 0 ≤ j ≤ c
3:     for 1 ≤ i ≤ n do
4:         for 1 ≤ j ≤ c do
5:             if w_i > j then
6:                 m_{i,j} ←m_{i−1,j}
7:             else
8:                 m_{i,j} ←max (m_{i−1,j}, m_{i−1,j−w_i} + p_i)
9:             end if
10:        end for
11:    end for
12:    return m_{n,c}
13: end function
```

$m_{i-1,c'-w_i} + p_i$, nebo předmět při výběru nepoužijeme, takže aktuální optimální výběr je optimální výběr podproblému, $m_{i-1,c'}$.

Algoritmus 2 vrací sumu hodnot optimálních předmětů přidaných do batohu. Jaké konkrétní předměty přispěly do sumy lze snado zjistit zpětným následováním rekurzivního vztahu.

Rádi bychom našli nějaký podobný rekurzivní vztah v problému hledání optimálního plánu. Nalezením rekurzivních vztahů by pak simulace s nebyla černou skříňkou a byla by splněna nutná podmínka (viz věta 10) pro šanci na polynomiální řešení.

2.1.1 Prohledávání prostoru plánů tahy

V předchozí kapitole jsme si ukázali, jak lze pomocí dynamického programování vyřešit náročnou kombinatorickou úlohu.

V této kapitole budeme zkoumat, jestli existují nějaké rekurzivní vztahy „optimálních plánu“ na méně incidentech s „optimálními plány“ na více incidentech, abychom efektivně vyřešili optimalizační problém nalezení optimálního plánu pohotovostní služby.

Nalezneme takový rekurzivní vztah (viz věta 12) a to pomocí prohledávání plánů pomocí *tahů* (viz definice 31), konkrétně *optimálních tahů* (viz definice 38). Optimálními tahy budeme postupně budovat *plány optimální v ceně* (viz definice 35), podle vhodně zvolených podmnožin množiny incidentů I, *seřazených incidentů* (viz definice 29). Věta o dosažitelnosti optimálními tahy 13 dokáže, že takový postup skutečně musí nalézt všechny optimální plány, ale pouze při účelové funkci q^{Lex}. Také navrhneme algoritmus (viz algoritmus 3), který korektně nalezne všechny takové optimální plány pro množinu I, a bude fungovat rychleji než naivní řešení. Detailní rozebrání složitosti algoritmu popisuje věta o složitosti algoritmu 17.

Definice 29 (Seřazené incidenty)**.** *Nechť množina incidentů*

$$I = \{i_1, \ldots, i_n\},\ \textit{kde}\ \forall_{j \neq k} j, k \in \{1, \ldots n\} \colon T_I(i_j) < T_I(i_k),$$

čili incidenty jsou seřazené podle času nastání. Definujme množiny incidentů

$$\emptyset = I_0, I_1, \ldots, I_n, \text{ kde } \forall k \in \{1, \ldots, n\} \colon I_k = \{i_1, \ldots, i_k\}.$$

Incidenty I_k jsou tak incidenty I_{k-1} po odebrání incidentu i_k, který se odehrál jako poslední.

Pomocí seřazených incidentů budeme postupně konstruovat plány optimální v ceně. Z plánu optimálního v ceně pro I_{k-1} získáme plány optimální v ceně pro I_k pomocí *tahů*, konkrétně nějakých *optimálních tahů*.

Definice 30 (Tah)**.** *Nadefinujme tah jako funkci $T \colon P_C \to P_C$. Pokud $p' = T(p)$, řekneme, že jsme plán p' získali z p tahem T.*

Definice 31 (Inverzní tah)**.** *Nechť tah T. Inverzní tah T^{-1} k tahu T je tah, pro který platí*

$$T(p) = p' \Leftrightarrow T^{-1}(p') = p.$$

Můžeme pak nějakou posloupností tahů převést jeden plán na druhý.

Definice 32 (Posloupnost tahů)**.** *Posloupností tahů velikosti n rozumíme posloupnost tahů $T_1, T_2, \ldots, T_n$. Řekneme, že z plánu p získáme plán p' posloupností tahů $T_1, T_2, \ldots, T_n$, pokud $(T_1 \circ T_2 \circ \cdots \circ T_n)(p) = p'$, kde symbol $\circ$ značí binární operaci skládání funkcí, kde se první vyhodnotí levá funkce a pak pravá.*

Následující *kanonické tahy* jsou zajímavé z toho důvodu, že vhodnými posloupnostmi vhodných kanonických tahů získáme *optimální tahy*, právě kterými budeme postupně budovat pro množiny incidentů I_{k-1} až I_k plány optimální v ceně.

Definice 33 (Kanonické tahy)**.** *Kanonickými tahy rozumíme tahy*

1. *Alokace záchranného týmu na výjezdovou stanici.*
2. *Alokace záchranného vozidla na výjezdovou stanici.*
3. *Prodloužení doby trvání směny již naalokovaného záchranného týmu.*
4. *Identita.*

Formálně,

1. *kanonický tah alokace týmu z' na výjezdovou stanici je tah T, kde*

$$p'_Z(z) = \begin{cases} v \neq v_\emptyset & \text{pro tým } z' \in Z \colon p_Z(z') = v_\emptyset, v \in V, \\ p_Z(z) & \text{pro } \forall z \in Z \setminus \{z'\}, \end{cases}$$
$$p'_D(z) = \begin{cases} (d_s, d_l) \in D \colon d_l - d_s > 0 & \text{pro } z', \\ p_D(z) & \text{pro } \forall z \in Z \setminus \{z'\}, \end{cases}$$
$$p'_A(a) = p_A(a), \forall a \in A,$$

pro $p_Z, p_D, p_A \in p$ a $p'_Z, p'_D, p'_A \in p'$, kde $p, p' \in P_C$ a $T(p) = p'$.

2. *Kanonický tah alokace záchranného vozidla a' na výjezdovou stanici je tah T, kde*

$$p'_Z(z) = p_Z(z), \forall z \in Z,$$
$$p'_D(z) = p_D(z), \forall z \in Z,$$
$$p'_A(a) = \begin{cases} v \neq v_\emptyset & \textit{pro záchranné vozidlo } a' \in A\colon p_A(a') = v_\emptyset, v \in V, \\ p_A(a) & \textit{pro } \forall a \in A \setminus \{a'\}, \end{cases}$$

pro $p_Z, p_D, p_A \in p$ a $p'_Z, p'_D, p'_A \in p'$, kde $p, p' \in P_C$ a $T(p) = p'$.

3. *Kanonický tah prodloužení směny týmu z' je tah T, kde*

$$p'_Z(z) = p_Z(z), \forall z \in Z$$
$$p'_D(z) = \begin{cases} (d'_s, d'_l) \in D\colon d'_l > d_l \wedge d'_s = d_s & \textit{pro } z' \\ p_D(z) & \textit{pro } \forall z \in Z \setminus \{z'\}, \end{cases}$$
$$p'_A(a) = p_A(a), \forall a \in A,$$

pro $p_Z, p_D, p_A \in p$ a $p'_Z, p'_D, p'_A \in p'$, kde $p, p' \in P_C$ a $T(p) = p'$.

4. *Kanonický tah posun začátku směny týmu z' je tah T, kde*

$$p'_Z(z) = p_Z(z), \forall z \in Z$$
$$p'_D(z) = \begin{cases} (d'_s, d'_l) \in D & \textit{pro } z' \\ p_D(z) & \textit{pro } \forall z \in Z \setminus \{z'\}, \end{cases}$$
$$p'_A(a) = p_A(a), \forall a \in A,$$

pro

$$(d'_s, d'_l)\colon d'_s \in D_S \wedge d'_l = p_{D_L}(z') \wedge d'_s < d'_l$$

a $p_Z, p_D, p_A \in p$ a $p'_Z, p'_D, p'_A \in p'$, kde $p, p' \in P_C$ a $T(p) = p'$.

5. *Kanonický tah identita je tah $T : p \to p$.*

Dále budeme tahy definovat pouze slovně, protože se bude vždy jednat o nějaké posloupnosti kanonických tahů nebo o jejich inverze.

Mohou existovat plány $p \in P_C$, na kterých nějaký tah nelze provést, čili $T(p)$ je nedefinované. Například, pokud bychom plánu chtěli alokovat tým, ale už nejsou žádné k dispozici, nebo už není volná žádná výjezdová stanice podle omezení C. Řekněme, že tah T *lze provést na plánu* $p \in P_C$ právě tehdy, když $T(p)$ je definováno.

Pomocí tahů můžeme P_C *prohledávat*. Plán p_2 *je dosažitelný tahy* T *z plánu* p_1, pokud existuje posloupnost tahů $T_1, \ldots, T_n$ taková, že $(T_1 \circ \cdots \circ T_n)(p_1) = p_2$. Pokud optimální plán $p^* \in P_C$ je nějakými tahy T_i dosažitelný z $p \in P_C$, pak ho můžeme prohledáváním, které začíná v p nalézt. Pomocí tahů můžeme P_C prohledávat sofistikovaněji, než je jenom všechny procházet, jak jsme dělali v naivním řešení 23.

Některé plány totiž ani nemá smysl prohledávat, jako například plány, které naalokují záchranáře natolik nešikovně, že neodbavují úspěšně žádný incident.

Taková naalokování jenom přispívají do ceny plánu, ale nijak do počtu úspěšně odbavených incidentů.

Dále budeme zkoumat, jaké plány má smysl prohledávat a jakými tahy jsme takové plány schopni nalézt.

Definice 34 (Tahy snižující cenu)**.** *Za tahy snižující cenu rozumíme tahy:*

1. *dealokování týmu,*
2. *dealokování vozidla,*
3. *zkrácení doby trvání směny,*
4. *posun začátku směny na později a zkrácení doby trvání směny.*

Tahy 1–3 jsou inverzní kanonické tahy pro alokování týmu, alokování vozidla a prodloužení směny. Tah 4 lze chápat jako zkrácení směny „zleva".

Definice 35 (Plán optimální v ceně)**.** *Nechť množina incidentů I a plán $p \in P_C$: $s(p, I) = r$. Řekneme, že p je optimální v ceně právě tehdy, když pro plán p' vzniklý z plánu p provedením libovolného tahu snižujícího cenu platí:*

$$s(p', I) < r.$$

Plán optimální v ceně je takový plán, že jakýkoliv pokus o snížení ceny nějakým tahem snižujícím cenu způsobí, že výsledný plán bude odbavovat méně incidentů. Následující věta objasní, k čemu jsou plány optimální v ceně užitečné.

Věta 11 (Optimální plán při účelové funkci q^{Lex} je optimální v ceně)**.** *Optimální plán p^* při účelové funkci q^{Lex} na množině incidentů I je optimální v ceně.*

Důkaz. Pro spor předpokládejme, že p^* není optimální v ceně. Pak existuje tah snižující cenu T takový, že

$$s(T(p^*), I) = s(p^*, I) \wedge u(T(p^*)) < u(p^*).$$

Plán $T(p^*)$ odbavuje stejně incidentů a zároveň je levnější. To je spor s optimalitou p^* pří účelové funkci q^{Lex}. □

Z věty 11 plyne, že pokud bychom nalezli způsob, kterým budeme schopni prohledávat plány optimální v ceně v množině P_C, tak nutně nalezneme i všechny optimální plány při účelové funkci q^{Lex}. Následujícím tahům budeme říkat *optimální tahy*, a těmito tahy budeme schopni z prázdného plánu budovat plány optimální v ceně. Následuje definování jednotlivých optimálních tahů.

Definice 36 (Tah minimální prodloužení směny týmu)**.** *Tahem minimální prodloužení směny týmu na plánu $p \in P_C$ pro množinu incidentů I_{k-1}, rozumíme tah T, prodloužení směny týmu a případně alokace vozidla tak, že*

1. $s(T(p), I_k) = s(p, I_{k-1}) + 1$,
2. *$T(p)$ je optimální v ceně.*

Definice 37 (Tah minimální alokace týmu)**.** *Tahem minimální alokace týmu na $p \in P_C$ pro množinu incidentů I_{k-1}, rozumíme tah T, alokace týmu a případně alokace vozidla tak, že*

1. $s(T(p), I_k) = s(p, I_{k-1}) + 1$,
2. *$T(p)$ je optimální v ceně,*
3. *T alokuje směnu s nejpozdějším možným začátkem směny.*

Definice 38 (Optimální tahy)**.** *Optimálními tahy T^* plánu $p \in P_C$ pro množinu incidentů I rozumíme množinu tahů:*

1. *tah minimální prodloužení směny týmu,*
2. *tah minimální alokace týmu,*
3. *tah identita,*

které lze na plánu p pro množinu I provést.

Následující věta vše spojí dohromady a poskytne základ pro postup, jak šikovně prohledávat plány pomocí optimálních tahů.

Věta 12 (O optimálních tazích)**.** *Nechť množiny incidentů $I_0, \ldots, I_n$ podle definice 29. Nechť plán optimální v ceně $p_k \in P_C \colon s(p_k, I_k) = r$. Pak existuje plán optimální v ceně*

$$p_{k-1} \in P_C \colon s(p_{k-1}, I_{k-1}) = r \vee s(p_{k-1}, I_{k-1}) = r - 1$$

a optimální tah T takový, že

$$T(p_{k-1}) = p_k,$$

pro všechna $k \in \{1, \ldots, n\}$.

Důkaz. Nechť $\forall k \in \{1, \ldots, n\}$, $i_1, \ldots, i_k$ incidenty I_k a $i'_1, \ldots, i'_r$ incidenty, které odbaví p_k. Rozlišme dvě situace, které mohou nastat:

1. $i'_r \neq i_k$, čili poslední odbavený incident i'_r není poslední incident, který v I_k nastal,
2. $i'_r = i_k$, čili poslední odbavený incident i'_r je poslední incident, který v I_k nastal.

V prvním případě je hledaný p_{k-1} právě p_k a tah T identita. Platí $s(p_k, I_{k-1}) = r$, protože p_k neodbaví poslední incident $i_k \notin I_{k-1}$ a simulace incidenty prochází postupně (viz algoritmus simulace 1), takže výpočet simulace na plánu p_k a na množinách I_k i I_{k-1} je do i_{k-1} incidentu totožný. Zároveň p_k je optimální v ceně na I_k a jelikož p_k odbavuje stejné incidenty na I_{k-1}, tak je optimální v ceně i na I_{k-1}.

V druhém případě, je poslední incident $i_k = i'_r$ plánem p_k odbaven. Z toho důvodu p_k na množině inidentů I_{k-1} odbavuje incidenty $i'_1, \ldots, i'_{r-1}$, o jeden méně – opět, simulace incidenty prochází postupně. Platí tak,

$$s(p_k, I_{k-1}) = r - 1.$$

Plán p_k však nemusí být optimální v ceně, takže T nemusí být tah identita jak pro případ $i'_k = i'_r$.

Nyní rozborem případů ukážeme, že v každé situaci, která může nastat, existuje optimální tah T splňující znění věty.

Uvažme tým záchranářů z, který odbavoval incident i_k. Pro tým z pak pro spuštění simulace na plánu p_k na množině incidentů I_{k-1} platí:

1. Tým z odbavuje jenom i_k, pak je v plánu p_k naalokován zbytečně. O odbavení všech ostatních $r-1$ incidentů se postarájí ostatní týmy.

2. Tým z odbavuje i jiné incidenty I_z, ale jelikož každý $z_i \in I_z$ nastal dříve, tak z odbavuje i_k až po odbavení všech incidentů I_z. Zde využíváme předpokladu, že incidenty se odehrávájí v různých časech (viz definice I v kapitole 1.1). Aby z úspěšně odbavil i_k , může mít delší směnu, než by potřeboval pro odbavení všech incidentů I_z.

V obou případech ještě může být zbytečně naalokováno vozidlo, které z používal pro odbavení i_k. Žádná jiná možnost nemůže nastat.

Z pozorování plyne, že vždy existuje dvojice tahů T_1^{-1} a T_2^{-1}, kde pro jejich složení $T^{-1} = T_1^{-1} \circ T_2^{-1}$, platí, že $T^{-1}(p_k)$ je optimální v ceně a $s(T^{-1}(p_k), I_{k-1}) = r-1$. Těmito tahy jsou:

1. dealokace týmu a dealokace vozidla,

2. dealokace týmu a identita,

3. maximální zkrácení směny týmu a dealokace vozidla,

4. maximální zkrácení směny týmu a identita,

5. identita a identita.

Maximálním zkrácením směny týmu z myslíme tah zkrácení směny T' tak, že

$$s(T'(p_k), I_{k-1}) = r - 1 \tag{2.3}$$

a směna už týmu z nelze zkrátit více, aby platila rovnost 2.3.

Případ 1 a případ 2 pokrývají situaci, kdy z odbavuje pouze i_k. Případ 3 a případ 4 pokrývají situaci, kdy z odbavuje i nějaké jiné incidenty a z jde zkrátit směnu, aby stále odbavoval všechny incidenty I_z. Poslední případ 4 pokrývá situaci, kdy z odbavuje i jiné incidenty, ale směnu mu nelze nijak zkrátit.

Ukázali jsme tak existenci T^{-1} pro všechny situace. Nechť

$$T = T_1 \circ T_2,$$

kde T_1 a T_2 jsou inverzními tahy k T_1^{-1} a T_2^{-1}. Alokace týmu jsou minimální, protože p_k byl plán optimální v ceně. Dohromady dostáváme, že tah T je optimální tah, a to v každém případě. □

Předchozí věta je velmi důležitá pro navrhnutí sofistikovanějšího algoritmu prohledávání plánů. V následující větě je její tvrzení využito pro dokázání dosažitelnosti libovolného optimálního plánu, ale pouze při účelové funkci q^{Lex}.

Věta 13 (O dosažitelnosti optimálního plánu optimálními tahy)**.** *Optimální plán $p^* \in P_C$ na množině incidentů I při účelové funkci q^{Lex} je z $p_0 = p_\emptyset$ dosažitelný nějakou posloupností optimálních tahů $T_0 \circ \ldots T_{|I|-1}$.*

Důkaz. Nechť $n = |I|$ a nechť plán p^* odbavuje $r = s(p^*, I)$ incidentů. Z věty 11 je p^* optimální v ceně. Existují pro něj podle věty 12 optimální tah T_{n-1} a plán p_{n-1} takové, že

$$p_n = T_{n-1}(p_{n-1})$$

a p_{n-1} odbavuje stejně nebo o jeden méně incidentů na I_{n-1} než r. Opět z věty 12 pro něj existují $p_{n-2}, T_{n-2} \colon p_{n-1} = T_{n-2}(p_{n-2})$, atd.

Takovou konstrukcí sestrojíme $p_k, T_k, \forall k \in \{0, \ldots, n-1\}$. Plán p_0 je z věty 12 plán na prázdné množině odbavující buď jeden nebo nula incidentů, optimální v ceně. Pro prázdnou množinu incidentů jakýkoliv jiný plán než prázdný plán $p_\emptyset$ není optimální v ceně. Tudiž plán p_0 je prázdný plán. Tahy T_k jsou optimální tahy, tudiž se jedná o hledanou posloupnost takovou, že

$$p^* = (T_0 \circ T_1 \circ \cdots \circ T_{n-1})(p_\emptyset).$$

□

2.1.2 Rekurzivní prohledávání stromu optimálních tahů

Věta 13 nám dává návod, jak plány P_C z $p_\emptyset$ prohledávat optimálními tahy $T_1 \in T_1^*, \ldots, T_{|I|-1} \in T_{|I|-1}^*$.

Algoritmus 3 Rekurzivní prohledávání prostoru plánů optimálními tahy

```
1: function OptimalMovesSearch(p, I_k, h, P*)
2:     if k = h then
3:         p* ←∀p ∈ P*
4:         if q^Lex(p*, I_k) = q^Lex(p, I_k) then
5:             P* ←P* ∪ p
6:         else if q^Lex(p*, I_k) < q^Lex(p) then
7:             P* ←{p}
8:         end if
9:         return P*
10:    end if
11:    T_k^* ←optimální tahy (viz definice 38)
12:    for T* ∈ T_k^* do
13:        if not Visited[k, T*(p)] then
14:            P* ←OptimalMovesSearch(T*(p), I_{k+1}, h, P*)
15:            Visited[k, T*(p)] ←true
16:        end if
17:    end for
18:    return P*
19: end function
```

Pro zjištění všech optimálních plánu P^* na množině incidentů I algoritmus (viz algoritmus 3) zavoláme: P^* = OptimalMovesSearch($p_\emptyset$, I_1, $|I|$, $\{p_\emptyset\}$).

Pokud je v kroku 2 dosaženo hloubky h, algoritmus 3 zkontroluje, zda je aktuální plán p lepší než dosud nejlepší plán $p^* \in P^*$ podle účelové funkce q^{Lex}:

1. Pokud je aktuální plán p lepší, v kroku 7 se P^* nahrádí množinou obsahující pouze aktuální plán p.
2. Pokud je stejně kvalitní, plán p se v kroku 5 přidá do množiny P^*.
3. Pokud je horší, plán p není relevantní a ignoruje se.

Plán p^* může být vybrán libovolně, protože všechny $p \in P^*$ mají stejné ohodnocení při účelové funkci q^{Lex}. V kroku 9 se vrátí aktualizovaná množina doposud nejlepších nalezených plánů P^*.

Na každé hladině rekurze se v kroku 11 vygenerují optimální tahy pro aktuální množinu incidentů I_k, a pokud plán $T^*(p)$ vzniklý optimálním tahem T^* nebyl navštíven, rekurzivně se na něj algoritmus zavolá a aktualizuje se *cache* navštívených stavů *Visited*.

Tímto způsobem se systematicky prohledává celý prostor možných plánů a ukládají se všechny doposud nejlepší nalezené plány P^*. Zároveň díky *Visited* se zbytečně neprohledává podstrom ve stejné hloubce se stejným plánem vícekrát.

Věta 14 (Korektnost a úplnost algoritmu)**.** *Algoritmus rekurzivního prohledávání prostoru plánů optimálními tahy (viz algoritmus 3) nalezne všechny optimální plány $P^* \subseteq P_C$ při účelové funkci q^{Lex}.*

Důkaz. Z věty 13 víme, že $p^* \in P^*$ je dosažitelný nějakou posloupností optimálních tahů z prázdného plánu $p_\emptyset$. V algoritmu v kroku 11 množina T_k^* obsahuje všechny optimální tahy, které lze provést na plánu p na množině I_k, počínaje prázdným plánem a prvním incidentem. V kroku 14 se rekurzivně zavoláme na každý $T^*(p), T^* \in T_k^*$, pokud již nebyl navštíven. Není důvod vícekrát prohledávát stejný podstrom, protože vrácená P^* by byla stejná. Algoritmus tudiž prohledá každou posloupnost optimálních tahů, takže určitě i všechny takové, kterými se získá libovolný optimální plán $p \in P^*$ (plyne z věty 13). Algoritmus je úplný.

V krocích 3-8 vybíráme pouze plány optimální vůči q^{Lex}. Vybíráme tak ze všech plánů, které jsou dosažitelné nějakou posloupností optimálních tahů, ty optimální při q^{Lex}. Zároveň i *cache* je korektně používaná, tudiž algoritmus je i korektní. □

Omezili jsme se na použití účelové funkce q^{Lex}, protože potřebujeme, aby optimální plán podle účelové funkce q byl zároveň optimální v ceně. Tento předpoklad optimální plán při q^{Lex} splňuje (viz věta 11). Aby byl algoritmus korektní, mohli bychom použít jakoukoliv jinou účelovou funkci q, pro kterou platí, že optimální plán při q je zároveň optimálním v ceně. Podle věty 13 bude dosažitelný optimálními tahy, takže jej algoritmus navštíví.

Víme, že algoritmus je korektní. Ještě by nás zajímala jeho časová složitost. Pro tento účel si prvně odhadneme maximální počet optimálních tahů na hladině.

Věta 15 (Odhad počtu optimálních tahů)**.** *Nechť plán optimální v ceně $p \in P_C$ a nechť optimální tahy T, které lze provést na plánu p na množině I_k. Pak*

$$1 \leq |T| \leq Z_n + 1.$$

Důkaz. Optimální tahy jsou minimální alokace týmu, minimální prodloužení směny týmu a identita. Zaprvé spočítejme, kolik je tahů minimální alokace týmu. Nechť Z_1 počet nenaalokovaných záchranných týmů. To jsou jediné týmy, které lze alokovat. Minimální alokace přiřazuje naalokovanému týmu z nejkratší směnu se začátkem posunutým co nejpozději a případně alokuje záchranné vozidlo, aby byly odbavovány všechny incidenty I_{k+1}. Taková alokace ale existuje pro každý nenaalokovaný tým právě jedna.

Zadruhé spočítejme, kolik je tahů minimální prodloužení týmu. Nechť Z_2 počet naalokovaných záchranných týmů. To jsou jediné týmy, kterým lze prodlužovat směna. Minimální prodloužení směny týmu z směnu prodlouží o nejkratší možnou dobu a případně alokuje záchranné vozidlo, aby byly odbavovány všechny incidenty I_{k+1}. Takové prodloužení opět, tentokrát pro naalokované týmy, existuje právě jedno.

Dohromady dostáváme: $Z_1 + Z_2 = Z_n$ a tah identita lze provést vždy a je pouze jeden, čili:

$$1 \leq |T| \leq Z_n + 1.$$

□

Díky předchozí větě známe maximální počet optimálních tahů na každé hladině rekurze. V průběhu algoritmu se tvoří *strom rekurze* s větvícím faktorem úměrným počtu optimálních tahů a hloubkou stromu rovnou velikosti množiny incidentů I. V následující větě spočítáme jeho velikost, což je klíčové pro zjištění složitosti algoritmu.

Věta 16 (Velikost stromu rekurze)**.** *Počet vrcholů stromu rekurze algoritmu je nanejvýš*

$$\frac{(Z_n + 1)^{|I|+1}}{Z_n}$$

Důkaz. Větvení stromu je dáno počtem optimálních tahů. Z věty 15 jich je nanejvýš $Z_n + 1$. Hloubka stromu je $|I|$, proto počet vrcholů stromu činí nanejvýš

$$\sum_{k=0}^{|I|} (Z_n + 1)^k = \frac{(Z_n + 1)^{|I|+1}}{Z_n}.$$

□

Nyní máme všechny prostředky pro spočítání složitosti algoritmu.

Věta 17 (Složitost algoritmu rekurzivního prohledávání optimálních tahů)**.** *Algoritmus rekurzivního prohledávání optimálních plánů běží v čase*

$$\mathcal{O}(Z_n^{|I|}).$$

Důkaz. Z věty 16 je počet vrcholů stromu optimálních tahů nanejvýš

$$\frac{(Z_n + 1)^{|I|+1}}{Z_n}.$$

V každém vrcholu procházíme všechny optimální tahy plánu. Z věty 15 jich je nanejvýš $Z_n + 1$. □

Nalezli jsme korektní algoritmus při účelové funkci q^{Lex}, který běží v exponenciálním čase podle počtu incidentů. Algoritmus sice neběží v polynomiálním čase, ale stále se jedná o dramatické zlepšení oproti naivnímu řešením, jehož složitost je rovna velikosti množiny plánů (viz věta 4).

Hlavní výhodou algoritmu je schopnost plány prohledávat strukturovaně. Díky tomu můžeme uvažovat i různé jiné varianty algoritmu, kdy například nemusíme strom rekurze prohledávat deterministicky, ale stochasticky tak, že v každé hladině rekurze budeme optimální tahy náhodně permutovat. Nemusíme ani na každé hladině zkoušet všechny optimální tahy, ale například pouze jeden náhodný, nebo nějakou část. Oproti postupnému prohledávání všech tahů na každé úrovni má tento přístup výhodu ve vyzkoušení více odlišných konfigurací a nalezené plány budou s menší pravděpodobností sdílet podobně naalokované týmy, vozidla a směny, což povede k diversifikovanějšímu prohledání množiny povolených plánů. Při aplikování metod budeme uvažovat tuto lehce pozměněnou verzi algoritmu (viz kapitola 3.3).

2.2 Metaheuristické metody

V kapitole 1.6.4 jsme diskutovali, do jaké třídy složitosti patří optimalizační úloha nalezení optimálního pohotovostního plánu. Konkrétně jsme si ve větě 10 ukázali, že pokud na účelovou funkci budeme nahlížet jako na černou skříňku, tak je optimalizační úlohu NP-těžká. V předchozí kapitole jsme nalezli rekurzivní vztah, který nám umožnil se na účelovou funkci nedívat jako na černou skříňku a nalezli jsme dramaticky lepší algoritmus 3, než naivní řešení (viz kapitola 1.6.3).

V této kapitole jsou diskutovány metody, které na účelovou funkci pohlížejí jako na černou skříňku. Připouštíme tedy NP-těžkou složitost problému, ale ukáže se, že i tak budeme schopni docílit velmi kvalitních pohotovostních plánů.

2.2.1 Popis

Metaheuristické prohledávání prostoru konfigurací patří mezi nejefektivnější způsoby řešení těžkých optimalizačních problémů, jak diskrétních, tak spojitých, které nelze jednoduše vyřešit v polynomiálním čase. Tyto metody prohledávání prostoru konfigurací koordinují interakce mezi lokálním a globálním optimalizováním, aby byly schopny prohledávat lokálně optimální konfigurace, ale zároveň aby z nich mohly uniknout a prostor prohledávat robustně [11].

Často metaheuristiky při prohledávání využívají konceptu *sousedství*. Míra, do jaké jsou sousedství využívána, se liší podle konkrétního typu metody. Některé metaheuristiky, jako například *simulované žíhání* nebo *tabu prohledávání*, využívají *přípustné tahy*, pomocí kterých přecházejí z jednoho řešení na jiné v lokálním pohledu. Zároveň dochází v lokálním pohledu k výběru horších konfigurací, aby se zajistila robustnost prohledávání a metoda neuvázla pouze v lokálním optimu [11].

V případě metod, založených na *populaci*, jako jsou například *genetické algoritmy* [12], jsou sousedství implicitně definována nahrazováním komponent jednoho řešení komponentami jiného řešení, na základě různě zvolených pravidel výměny.

Dalším zajímavým příkladem metaheurstické metody je *optimalizování kolonií mravenců*, spadající do kategorie metaheuristik *inteligence hejna* [13]. Sousedství jsou definována propojením cest a procházena pomocí simulování chodu mravenců,

reprezentující kolektivní inteligenci. Mravenci chodí po těchto cestičkách, a vybírají si tak optimální konfigurace s pravděpodobností, jenž je určena feromony.

Existuje spousta dalších různých metod. V poslední době jsou velmi populární *hybridní metody*, které propojují více metaheuristických metod dohromady, s myšlenkou použít každou metaheuristiku co nejvhodněji, případně i měnit různé metaheuristiky za běhu [14].

2.2.2 Lokální prohledávání

Asi nejjednodušší metoda na prohledávání prostoru konfigurací je *lokální prohledávání*. Lokální prohledávání pouze lokálně prohledává prostor konfigurací od dané startovní konfigurace $x \in \mathcal{X}$. Prostor prohledává prostřednictvním sousedů $N(x)$, kde vždy vybere souseda s nejlepší hodnotou účelové funkce. (Christian Blum [14], str. 3).

Sousedy $N(p)$ zkonstruujeme implicitně pomocí tahů, což je standardní způsob konstrukce sousedů nějaké konfigurace (Christian Blum [14], str. 3). Sousední plány $N(p)$ plánu $p \in P_C$ lze definovat různě.

My budeme od $p' \in N(p)$ vyžadovat alespoň následující:

1. $\forall p \in P_C \colon N(p) \subseteq P_C$.
2. $\forall p \in P_C, \exists (T_1, \ldots, T_n) \colon (T_1 \circ \cdots \circ T_n)(p) = p^*$, kde p^* je globálně optimální plán.
3. Pro všechny $p' \in N(p)$ pro dané p platí, že jsou si plány p a p' podobné, čili z větší části mají naalokovaná stejná vozidla a týmy, se stejnými směnami.

Podmínka 1 zakazuje navštěvování nedovolených plánů. Pro některé problémy je navštěvování nedovolených konfigurací žádoucí, protože omezující podmínky mohou být příliš svazující (Michel Gendreau [11], str. 50-51). To ale není náš případ, a tedy se můžeme omezit pouze na dovolené plány.

Podmínka 2 zajišťuje, že při prohledávání je vůbec možné globální optimum navštívit.

Podmínka 3 vyžaduje, aby sousední plány sdílely s původním plánem nějaké komponenty. Kdyby sousední plány nesplňovaly tuto podmínku, prohledávání pomocí sousedů by se o moc nelišilo od náhodného generování dovolených plánů, což není efektivní způsob prohledávání.

Příklad 2.2.1. Pro představu, jedna z možných definic sousedství by mohla vypadat následovně. Jelikož sousední plány definujeme pomocí tahů, stačí nadefinovat tahy:

1. alokování záchranného týmu a záchranného vozidla,
2. dealokování záchranného týmu,
3. dealokování záchranného vozidla,
4. nejkratší prodloužení směny záchranného týmu,
5. nejkratší zkrácení směny záchranného týmu,

6. posun začátku směny záchranného týmu o nejkratší možnou dobu.

Je snadné ověřit, že sousedství definována implicitně těmito tahy splňují všechny podmínky.

Podívejme se, jak vypadá algoritmus lokálního prohledávání plánů pomocí sousedů.

Algoritmus 4 Lokální prohledávání plánů pohotovostních služeb

```
1: function LocalSearch(p_start, I)
2:     p_current ←p_start
3:     while true do
4:         p* ←p_∅
5:         for p ∈ N(p_current) do
6:             if q(p, I) > q(p*, I) then
7:                 p* ←p
8:             end if
9:         end for
10:        if p* = p_∅ then
11:            return p_current
12:        end if
13:        p_current ←p*
14:    end while
15: end function
```

Algoritmus 4 lokálního prohledávání v krocích 3 až 14 prohledává sousedy dokud není splněna podmínka v kroku 10. Algoritmus v kroku 5 prohledává sousední plány splňující omezující podmínky C, aktuálně prohledávaného plánu $p_{current}$, který je na začátku algoritmu v kroku 2 nastaven na $p_{start} \in P_C$.

V krocích 6 a 7, pokud má soused lepší hodnocení pří účelové funkci $q \in Q_I$, tak je aktuálně nejlepší soused p^* změněn na souseda p. Podmínka v kroku 10 kontroluje, zda existuje nějaký soused $p_{current}$, který má lepší hodnocení při q. Pokud ne, algoritmus našel lokální maximum, tím je $p_{current}$, a proto je vrácen v kroku 11.

V kroku 13 je $p_{current}$ nastaveno na nejlepší sousední plán. Zřejmě lokální prohledávání nalezne nějaké lokální maximum, jaké konkrétně, závisí na startovním plánu p_{start} a definici $N(p)$.

Věta 18 (Časová složitost lokálního prohledávání)**.** *Nechť libovolný plán $p \in P_C$, pak lokální prohledávání běží v čase*

$$\mathcal{O}(k \cdot |N(p)| \cdot |I| \cdot |Z|),$$

kde k je počet iterací, než lokální prohledávání nalezne nějaké lokální maximum.

Důkaz. Prohledáváme v každé iteraci všechny sousedy. Celkem je tudiž prohledáno $k \cdot |N(p)|$ plánů. Každý plán vyhodnocujeme při účelové funkci $q \in Q_I$. Vyhodnocení $q(p, I)$ je konstantní až na spuštění simulace. Složitost simulace je $|I| \cdot |Z|$, protože pro každý incident je třeba nalézt nejvhodnější plán. Aktualizace stavů a spočítání doby odbdavování považujeme za konstantní. □

Pokud by k i $|N(p)|$ byly „malé“, tak jsme získali velmi rychlý způsob, jak upravit libovolný plán, abychom dostali jeho „nejlepší verzi“. Startovní plán definuje určitý základ, který je následně doplňován nebo redukován o záchranné týmy či vozidla, nebo jsou týmům postupně prodlužovány či zkracovány směny, dokud jakákoliv změna už pouze nezhorší výslednou kvalitu. Jak konkrétně už záleží na konkrétní definici $N(p)$.

Velkou nevýhodou lokálního prohledávání je neschopnost umět nalézt kompromis mezi lokálním optimem a globálním optimem, přičemž nalezené lokální optimum je v drtivé většině suboptimální (Michel Gendreau [11], str. 43).

2.2.3 Tabu prohledávání

Tabu prohledávání stejně jako lokální prohledávání prohledává plány lokálně pomocí sousedů. Tabu prohledávání se chová stejně jako lokální prohledávání, avšak narozdíl od lokálního prohledávání po nalezení lokálního optima v něm nezůstane, ale umožní i navštívení souseda s horším ohodnocením. Aby se následně prohledávání nevrátilo zpět do stejného optima, drží si seznam inverzních tahů posledních provedených tahů. Tomuto seznamu se říká *tabu* a jedná se o tahy, které jsou při prohledávání zakázany. Velikost seznamu tabu je takzvaná *tabu tenura*, a jelikož se vždy ze seznamu odstraní nejposledněji provedený tah, tak tabu tenura určuje, pro jaký počet kroků tabu prohledávání se tah uvažuje, že je zakázan. Ovšem vždy zakázat tahy z tabu může být příliš omezující a výsledná nalezená řešení by mohla být suboptimální. Z toho důvodu ještě tabu metoda využívá konceptu *aspiračního kritéria*, které pokud je pro nějaký tah splněno, tak je soused po provedení tohoto tahu uvažován i pokud je tah zakázan, čili i když se tah nachází v tabu [15].

Tabu prohledávání lze ukončit několika způsoby. Nejčastěji se prohledávání ukončuje, jakmile v iteraci už neexistují žádní zlepšující sousedé, které získáme tahem, jenž není v tabu. (Michel Gendreau [11], str. 41–49).

Algoritmus 5 Tabu prohledávání plánů pohotovostních služeb

```
1: function TabuSearch(p_start, I, t)
2:     p_current ←p_start
3:     p_best ←p_start
4:     tabu ←{}
5:     while true do
6:         p* ←p_∅
7:         for T ∈ neighbours_moves(p_current) do
8:             p ←T(p_current)
9:             if q(p, I) > q(p*, I) and (T ∉ tabu or q(p, I) > q(p_best, I)) then
10:                p* ←p
11:            end if
12:        end for
13:        if p* = p_∅ then
14:            return p_best
15:        end if
16:        p_current ←p*
17:        if q(p_current, I) > q(p_best, I) then
18:            p_best ←p_current
19:        end if
20:        AddFirst(tabu, T^-1)
21:        if |tabu| > t then
22:            RemoveLast(tabu)
23:        end if
24:    end while
25:    return p_best
26: end function
```

Algoritmus 5 tabu prohledávání si v krocích 1-5 inicializuje stav algoritmu. Tabu je ze začátku prázdné. Podobně jako v lokálním prohledávání se prohledává prostor v krocích 5 až 24, dokud není splněna podmínka na řádku 13. V krocích 7–12 se prohledávájí pomocí sousedních tahů sousedé aktuálně prohledávaného plánu $p_{current}$.

Nejzajímavější je krok 9. Zde se prvně kontroluje stejně jako v lokálním prohledávání, zda je spěna nutná podmínka, tedy jestli je vůbec třeba souseda p uvažovat. Druhá podmínka kontroluje, zda p buď není v tabu, nebo je splněné aspirační kritérium. Tah T, který v tabu uvažujeme, pouze pokud soused $T(p_{current})$ je doposavad nejoptimálnějším nalezeným plánem. Takové aspirační kritérium je nejčastěji používané a obecně nejefektivnější aspirační kritérium (Michel Gendreau [11], str. 47).

Podmínka v kroku 13 je totožná s podmínkou u lokálního prohledávání. Pokud již neexistuje žádný zlepšující soused, který není v tabu nebo nesplňuje aspirační kritérium, tak vrátíme nejlepší nalezený plán p_{best}. V kroku 17 aktualizujeme p_{best}, pokud jsme v aktuální iteraci nalezli kvalitnější plán.

V poslední řadě, v krocích 20–23 aktualizujeme tabu. Do taba přidáváme inverzní tah, protože se ze souseda nechceme vrátit do původního plánu. Pokud je tabu větší, než dovoluje hyperparametr tabu tenura t, odebereme nejdříve přidaný tah.

Tabu prohledávání již narozdíl od lokálního prohledávání nezůstane uvězněné v lokálním optimu a díky tabu se do něj nevrátí a nezacyklí se. Prohledá tak více plánů a bude vracet lepší řešení, než lokální prohledávání [15].

2.2.4 Simulované žíhání

Další variantou lokálního prohledávání je *simulované žíhání*. Podobně jako tabu prohledávání dovoluje navštěvovat lokálně neoptimální konfigurace, se snahou nalézt globální optimum. Je to jedna z nejpopulárnějších technik na řešení těžkých optimalizačních úloh, především kombinatorických úloh [16].

Simulované žíhání je inspirováno metalurgickým procesem žíhání, kdy je kov zahřán na vysokou teplotu a následně ochlazován, za účelem odstranění vnitřních defektů. Stejně jako ve fyzickém procesu žíhání, simulované žíhání si drží teplotu. Při vyšší teplotě je větší šance na navštěvování horších konfigurací a při nižší se pravděpodobnost snižuje, takže se navštěvují především lokálně optimální konfigurace.

Při postupném chlazení, tedy snižování teploty, se v průběhu běhu simulovaného žíhání postupně navštěvují horší konfigurace méně a optimální konfigurace více častěji. Tento proces má zajistit ze začátku dostatečnou exploraci prostoru konfigurací, ale zároveň postupnou konvergenci, ideálně k nějakému globálnímu řešení (Michel Gendreau [11], str. 1-2).

Algoritmus 6 Simulované žíhání prohledávání plánů pohotovostní služby

```
1:  function SimulatedAnnealing(p_start, t_start, M_k, t_cooling, t_final)
2:      p_current ← p_start
3:      p_best ← p_start
4:      t ← t_start
5:      while t > t_final do
6:          for m ← 1 to M_k do
7:              p ← zatím neprohledaný ∈ N(p_current)
8:              Δq ← q(p_current, I) − q(p, I)
9:              if Δq ≤ 0 then
10:                 p_current ← p
11:             else
12:                 p_current ← p s pravděpodobností exp(−Δq/t)
13:             end if
14:             if q(p, I) > q(p_best, I) then
15:                 p_best ← p
16:             end if
17:         end for
18:         t ← t_cooling(t)
19:     end while
20:     return p_best
21: end function
```

Algoritmus 6 simulovaného žíhání si v krocích 2 až 4 inicializuje aktuálně prohledávaný plán p_{current}, nejlepší nalezený plán p_{best} a akutální teplotu t. Hlavní smyčka algoritmu v krocích 5–19 běží, dokud t nepoklesne pod nejnižší dovolenou

teplotu t_{final}. V každé iteraci se provádí M_k iterací, přičemž každá iterace se snaží najít lepší plán ze sousedních plánů p. V kroku 8 se vypočítá rozdíl kvality Δq aktuálně nejlepšího plánu a sousedního plánu.

Podle Δq se rozhodne, zda přijmout sousední plán rovnou, nebo až v kroku 12. V kroku 12 je p přijato s pravděpodobností definovanou jako

$$\exp\left(-\frac{\Delta q}{t}\right). \tag{2.4}$$

Pravděpodobností funkce 2.4 je takzvané *metropolis akceptační kritérium* [17], které modeluje jak přechází termodynamický systém z aktuální konfigurace. Konkrétně určuje s jakou pravděpodobností v závislosti na aktuální teplotě t je suboptimální soused uvažován.

V krocích 14–16 se standardně aktualizuje p_{best}, pokud p má vyšší kvalitu než dosud nejlepší plán. Po provedení M_k iterací se v kroku 18 snižuje teplota t pomocí ochlazovacího rozvrhu $t_{\text{cooling}}(t)$. Jakmile aktuální teplota klesne pod t_{final}, algoritmus vrátí nejlepší nalezený plán p_{best}.

3 Aplikace metod

V této kapitole se zabýváme aplikováním a porovnáváním metod, kterým jsme se věnovali v kapitole 2. Namodelujeme pohotovostní službu a synteticky vytvoříme sadu incidentů, pro kterou budeme hledat optimální pohotovostní plán. V rámci této práce namodelujeme pražskou záchranou službu pro rok 2017. Budeme vycházet z veřejně dostupných dat poskytnutých přímo zdravotnickou záchrannou službou hlavního města Prahy[1]. Především nás bude zajímat kvalita nalezených optimálních plánů a rychlost jejich nalezení.

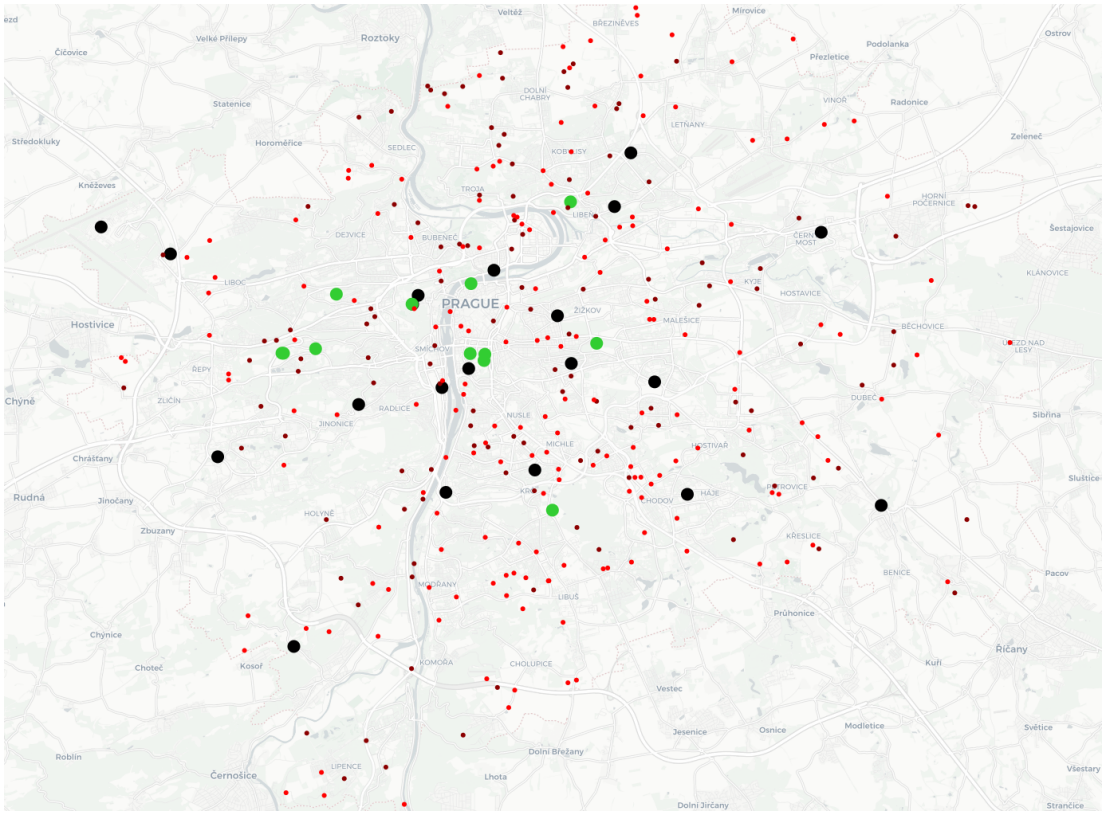
3.1 Generování dat

Nejprve vygenerujeme data reprezentující výjezdové stanice a nemocnice, čímž namodelujeme pohotovostní službu. Ty můžeme vygenerovat libovolně, ale z důvodu praktičtějšího využití si v této prácí vybereme jako pohotovostní službu pražskou záchrannou službu. Pražská záchranná služba disponuje 20 výjezdovými stanicemi a 140 vozidly. Údaje o počtu záchranných týmu nejsou veřejně dostupné. Jedinou dostupnou informací je celkový počet zaměstanců, který činí přes 500 lidí. Tam ale spadají i nezáchranáři. V Praze se nachází 12 nemocnic.

Nyní namodelujeme incidenty. Podle statistik zveřejněných pražskou záchrannou službou se v roce 2017 v Praze denně uskutečnilo průměrně na 330 výjezdů. Nejrušnější část všedního dne je mezi 9 a 12 hodinou, kdy se odehrává až dvojnásobně incidentů než je průměr. Incidenty se odehrávají mnohem častěji ve středu Prahy, než v jejím okolí.

Podle těchto informací namodelujeme sadu incidentů. V první řadě si území Prahy reprezentujeme jako mnohoúhleník a pomocí normálního rozdělení vygenerujeme souřadnice v něm obsažené. Normálnímu rozdělení nastavíme střední hodnotu na střed mnohoúhelníka a směrodatnou odchylku nastavíme tak, aby se incidenty na okrajích Prahy odehrávaly méně než v centru. Kde a v kolik hodin se incidenty odehrávají přesně záchranná služba Prahy nezveřejňuje, pravděpodobně proto, že by se mohlo jednat o zneužitelné nebo citlivé údaje. Rozloha Prahy činí 496 kilometrů čtverečních a je relativně symetrického tvaru. Pro naše účely tak bude stačit nastavit směrodatnou odchylku na 10 kilometrů. Incidenty jsou vygenerovány tak, aby se děly častěji mezi 9 a 12 hodinou, a odpovídaly tak skutečnosti.

[1] https://www.zzshmp.cz/wp-content/uploads/2017/12/Statistiky-160let-ZZSHMP.pdf



Obrázek 3.1 Záchranná pohotovostní služba a nemocnice spolu s incidenty na mapě Prahy.

Na obrázku 3.1 je znázorněna mapa Prahy s rozmístěnými 20 výjezdovými stanicemi (body černé barvy), 12 nemocnicemi (body zelené barvy) a 300 incidenty (body červené barvy). Tmavě červené body reprezentují incidenty, které se odehrají mezi 9 a 12 hodinou.

S výše popsanou reprezentací pohotovostní služby a sadou incidentů je třeba zajistit, aby simulace uměla věrohodně zjistit dobu trvání každého příjezdu. Připomeňme si, že simulaci používáme právě z toho důvodu, aby počet úspěšně odbavených incidentů plánem byl co nejvěrohodnější. Věrohodnost zajistíme použitím Google APIs, která nám pomohou zjistit doby všech příjezdů. Konkrétně použitím Distance Matrix API a Routes API.

V průběhu simulace je potřeba především znát následující doby příjezdů:

1. z výjezdové stanice na incident,
2. z incidentu do nemocnice,
3. z nemocnice zpět na výjezdovou stanici.

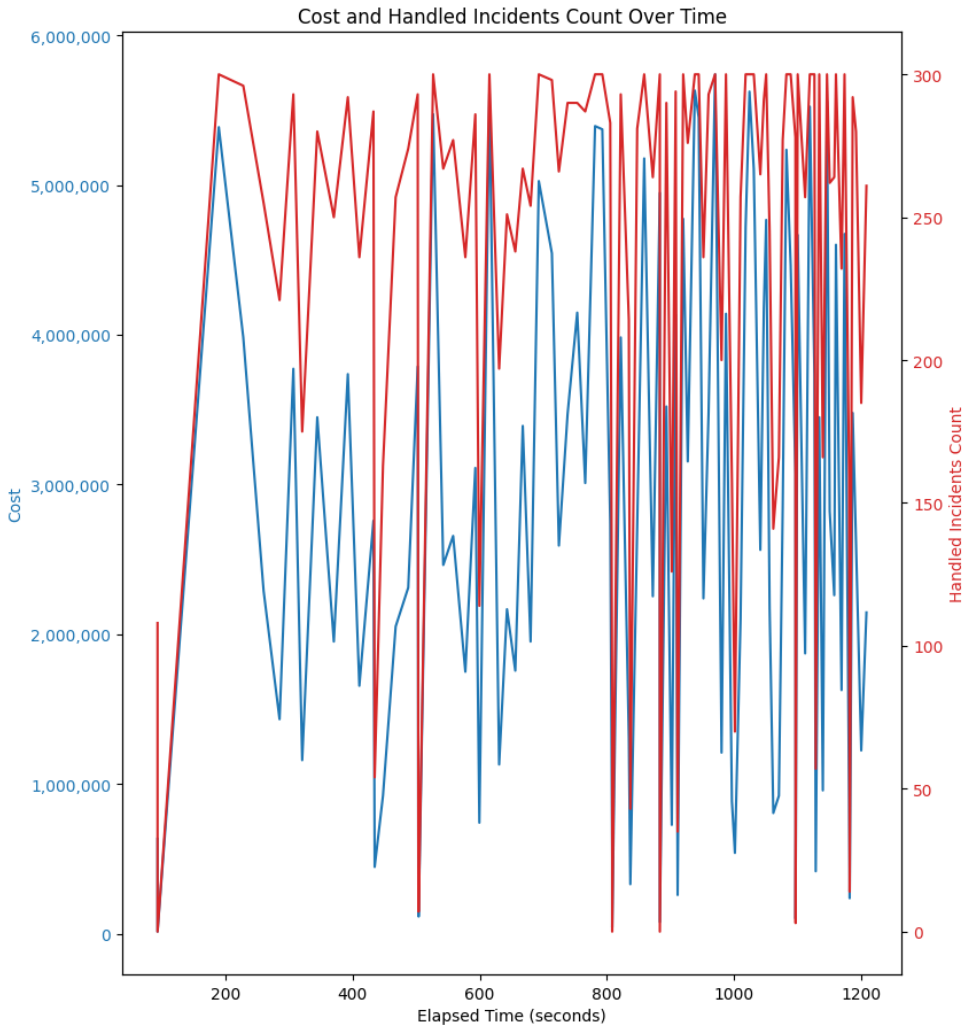
Simulace podporuje i tzv. *reroute*, tedy v momentě, kdy se záchranný tým vrací po odbavení incidentu zpět na výjezdovou stanici, tak je povoleno, aby mohl z aktuální lokace vyrazit na odbavování dalšího incidentu, aniž by se musel vracet zpět na výjezdovou stanici.

Doby příjezdů mezi výjezdovými stanicemi, incidenty a nemocnicemi si můžeme předpočítat, ale *reroute* předpočítat nelze. Můžeme si alespoň v průběhu simulování

všechny spočítané doby příjezdů mezi lokacemi udržovat s přesností na desítky metrů, aby se v budoucnu nemuselo volat Google API. To je samozřejmě pomalá operace, trvající desítky až stovky milisekund.

3.2 Aplikace naivního řešení

První metodu, kterou aplikujeme na namodelovanou záchrannou službu a incidenty dle předchozí kapitoly je naivní řešení.



Obrázek 3.2 Nalezené plány naivním řešením.

Na grafu 3.2 vidíme naivní řešení spuštěné po dobu 20 minut. Naivní řešení pouze náhodně generuje pohotovostní plány a vyhodnocuje je při účelové funkci. Můžeme vidět, že nalezené plány velmi kolísají jak v ceně tak v počtu odbavených incidentů. Naivní řešení není nijak více zajímavé a nebudeme jej dále zkoumat. Slouží spíše jako základ, se kterými můžeme porovnávat lepší metody, diskutované níže.

3.3 Aplikace prohledávání plánů optimálními tahy

V této kapitole aplikujeme algoritmus prohledávání plánů optimálními tahy 3. Konkrétně použijeme jeho mírně upravenou verzi, diskutovanou na závěru kapitoly 2.1.2, kde strom tahů budeme prohledávat vždy od prázdného plánu, a na každé hladině rekurze zvolíme pouze jeden náhodný optimální tah.

Čas běhu v minutách	Cena plánu	Odbavené incidenty
7:11	2455260	292
10:53	**2469661**	**296**
13:38	2433659	294
16:28	2347256	294
19:23	2376058	293
22:06	2340053	290
25:14	2433661	292
28:13	2368858	293
30:46	2404856	294
33:08	2448061	291

Tabulka 3.1 Spuštění prohledávání optimálními tahy na modelu Prahy.

Po spuštění metody na modelu Prahy po dobu kolem 30 minut metoda navštívila 10 plánů dosažitelných optimálními tahy. Celkově při budování těchto 10 plánů však navštívila něco přes $300 \cdot 10 = 3000$ plánů, protože algoritmus plány buduje postupně podle incidentů (viz algoritmus 3). Nalezený optimální plán odbavuje 296 incidentů a stojí 2469661 (viz tabulka 3.1). Dále má naalokovaných 100 týmů a 61 záchranných vozidel.

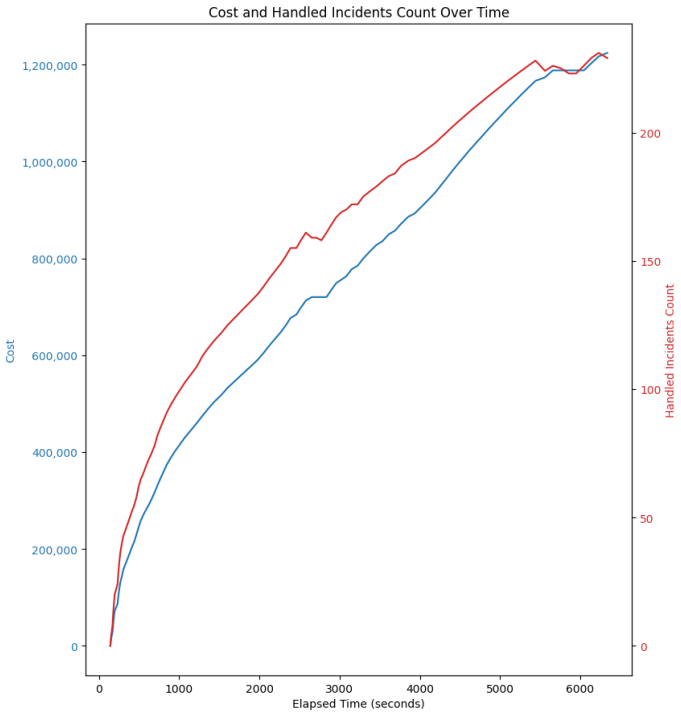
První plán trvá nalézt nejdéle, něco přes 7 minut. Důvodem je, že *cache* dob příjezdů obsahuje pouze předpočítané hodnoty a musí se vykonávat větší množství dotazů na Google API. Při prohledávání dalších plánů už se dotazuje na Google API mnohem méně často. Pro zajímavost, celkový počet dotazů na dobu trvání příjezdu je 133189050, kde pro 133179688 případu, tedy 99.99%, byla doba příjezdu předpočítaná a vrácena z *cache*. Podobně se *cache* chová i u ostatních metod.

3.4 Aplikace lokálního prohledávání

V této kapitole aplikujeme na model Prahy lokální prohledávání (viz kapitola 2.2.2). Metodu použijeme přesně jak je popsaná algoritmem 4. Prohledávání začneme z prázdného plánu a jako účelovou funkci použijeme váženou sumu ceny plánu a počtu odbavených incidentů, s parametrem $\alpha = 0.99$ (viz definice 13), abychom upřednostňovali plány s vyšším počtem odbavených incidentů. Jakou přesně účelovou funkci si zvolíme uvážíme podle toho, co pro nás konkrétně znamená optimální plán. My budeme chtít zkoumat především plány odbavující co nejvíce incidentů, a až poté minimalizovat cenu.

Potřebujeme ještě nadefinovat jak budou vypadat sousedi nějakého plánu. Ty standardně nadefinujeme implicitně pomocí tahů a vyžadujeme, aby splňovali alespoň nutné podmínky. Takové tahy můžeme vybrat různě, my si vybereme tahy uvedené jako příklad tahů, které nutné podmínky splňují (viz příklad 2.2.1).

Záměrně alokování týmu a vozidla zvolíme jako jeden tah a nikoliv jako dva samostatné tahy. Vyhneme se tak situaci, kdy by se lokální prohledávání zastavilo v lokálním optimu, kde už nelze odbavit žádný incident naalokování týmu a vozidla odděleně, ale pouze zároveň. Takové lokální optimum by pak bylo suboptimální, protože by odbavovalo méně incidentů, než by mohlo být možné. Takto definované sousedství budeme používat i u dalších metod.

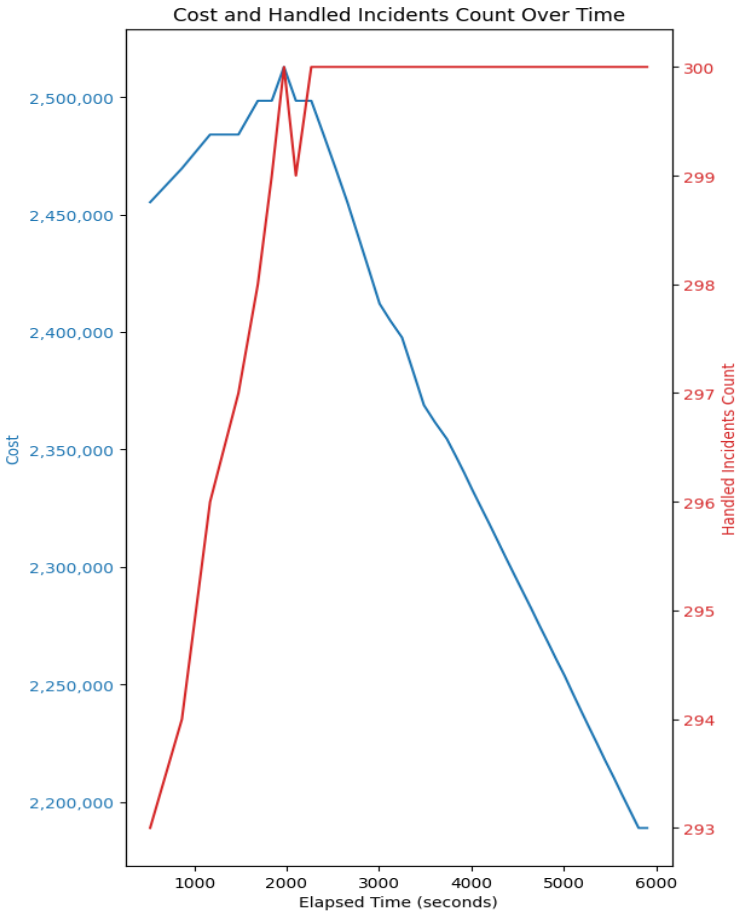


Obrázek 3.3 Nalezené plány metodou lokálního prohledávání plánů.

Na grafu 3.3 vidíme jak se aktuální plán mění v čase. Lokální prohledávání bylo spuštěné po dobu 50 minut, a pak bylo přeřušeno. Nejlepší nalezený plán po hodině a půl úspěšně odbaví 229 incidentů z 300 incidentů a stojí 1224080. Lokální prohledávání celkem navštívilo 29248 plánů. To je až 7 krát tolik, kolik plánů navštívila metoda prohledávání optimálními tahy.

I když se podle růstu křivky dá usoudit, že by metoda lokálního prohledávání byla schopná nalézt dost dobrý plán, trvalo by to příliš dlouho. Můžeme si polepšit, sice tak, že začneme prohledávát ne z prázdného plánu, ale z plánu zvoleného nějak chytře. Z plánu, který už je skoro optimální a lokální prohledávání už jej jenom doladí.

Jednou z možností jak takový startovní plán zvolit je použít první nalezený optimální plán předchozí metodou, a na něj spustit lokální prohledávání.

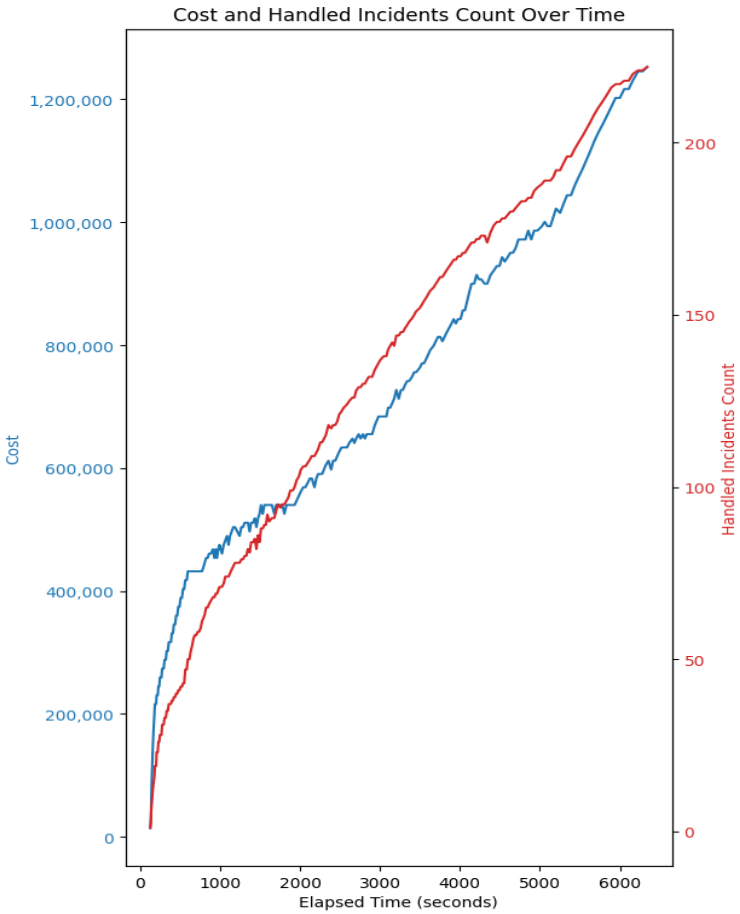


Obrázek 3.4 Nalezené plány metodou lokálního prohledávání plánů z plánu nalezeného optimálními tahy.

Na grafu 3.4 vidíme lokální prohledávání začínající ne z prázdného plánu, ale z plánu nalezeného optimálními tahy. Můžeme vidět, že lokální prohledávání upravuje plán tak, že postupně odbavuje více incidentů, až nakonec všech 300. Poté už jen postupně snižuje cenu plánu. Po zhruba 30 minutách nalezne plán odbavující všechny incidenty a do hodiny a půl lokální prohledávání stagnuje a nalezne lokální optimum. Optimální nalezený plán odbavuje všech 300 incidentů, stojí 2188863 a má naalokovaných pouze 88 týmů a 63 vozidel.

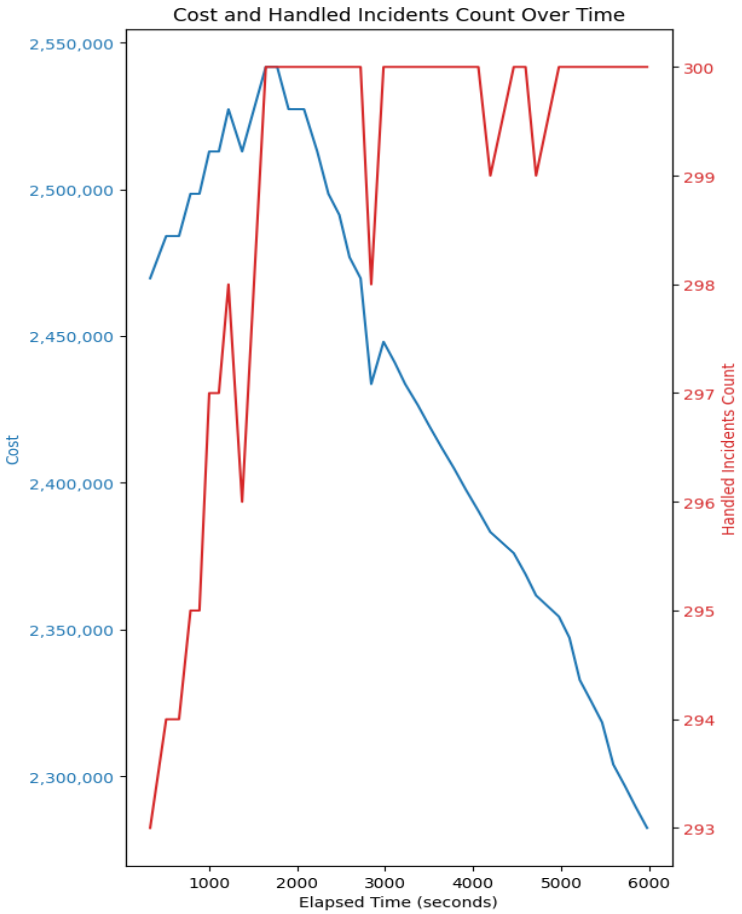
3.5 Aplikace tabu prohledávání

V této kapitole aplikujeme na model Prahy tabu prohledávání (viz kapitola 2.2.3). Tabu prohledávání je v podstatě lokální prohledávání s pamětí, podle které umí sofistikovaněji vybrat sousední plán, pokud se ocitne v lokálním optimu.



Obrázek 3.5 Nalezené plány tabu metodou z prázdného plánu.

Z toho důvodu bude podobně jako u lokálního prohledávání (viz graf 3.5) trvat příliš dlouho, než nalezne nějaký dost dobrý plán.



Obrázek 3.6 Nalezené plány tabu metodou z plánu nalezeného optimálními tahy.

Na grafu 3.6 vidíme doposud nejlepší plány nalezené tabu metodou v čase z plánu nalezeného optimálními tahy. Nalézt plán odbavující všech 300 incidentů trvá kolem 40 minut, což je o 10 minut déle, než trvalo lokálnímu prohledávání. To je pochopitelné, protože při navštívení každého souseda, kterých bude obdobně jako u lokálního prohledávání, se musí kontrolovat, zda není obsažen v tabu. Proto vyhodnocení souseda bude trvat o něco déle, což se při tak velkém množství navštívených plánů poměrně rychle nasčítá.

Tabu prohledávání bylo puštěno něco kolem hodiny a půl, a obdobně jako u lokálního prohledávání, s roustoucí délkou běhu programu se i snižuje cena plánu. Výhoda tabu prohledávání je hlavně v schopnosti nezůstat v lokálním optimu a umět robustněji prohledávat prostor konfigurací. Tato výhoda v našem případě ale není využita, protože tabu prohledávání ani za hodinu běhu na lokální optimum nenarazilo. Spíše naopak, prohledávání tabu je nevýhodou, protože zbytečně kontroluje tabu, a lokální prohledávání je v tomto případě lepší volbou.

3.6 Aplikace simulovaného žíhání

V této kapitole aplikujeme na model Prahy simulované žíhání (viz kapitola 2.2.3). Simulovanému žíhání je potřeba nastavit následující hyperparametry:

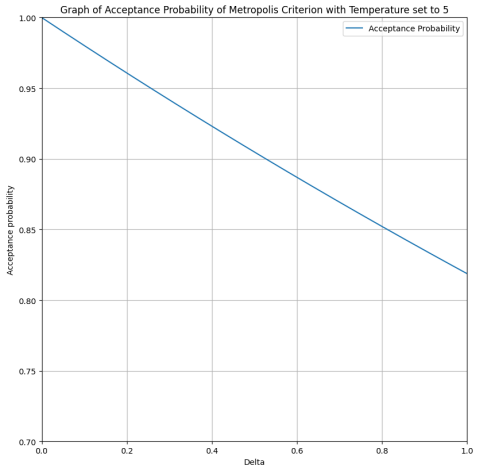
1. počáteční teplota,
2. koncová teplota,
3. chladící rozvrh,
4. počet iterací v rámci stejné teploty.

Počáteční teplotu je vhodné nastavit tak, aby pravděpodobnost přijetí horší konfigurace byla ze začátku kolem 0.8–0.9 procent [18]. Vzhledem k tomu, že používáme váženou sumu účelových funkcí, tak se výsledná hodnota účelové funkce pohybuje mezi 0 a 1. Tím pádem rozdíl se pohybuje mezi -1 a 1. Pokud je Δ záporná, tak je sousední plán vždy navštíven, protože je jeho hodnota při účelové funkci vyšší. Zajímá nás tedy pouze případ, kdy je Δ nekladná.

Připomeňme si jak vypadá akceptační kritérium:

$$\exp\left(-\frac{\Delta q}{t}\right).$$

Počáteční teplota splňující takový požadavek činí například 5 stupňů, jak můžeme vidět na obrázku 3.7, kde na ose x je hodnota delta, a na ose y pravděpodobnost přijetí sousedního plánu metropolisním kritériem, který se od aktuálního liší o danou deltu.



Obrázek 3.7 Metropolis kritérium pro teplotu 5 stupňů.

Finální teplota se často volí velmi blízko nule, aby bylo možné dostatečně důkladně prohledat konfigurace lokálně, podobně jak dělá například lokální prohledávání [18]. Zatím vyberme finální teplotu jako 10^{-6}.

Vybrat ochlazovací rozvrh lze různě, nejčastěji používané jsou [18]:

1. Lineární ochlazovací rozvrh, kde se teplota aktualizuje následovně:

   $$t_{k+1} = t_k - k \cdot \Delta t_k,$$

   kde t_k je aktuální teplota, t_{k+1} je nová teplota a Δt_k je hyperpametr, o kolik teplotu snižovat. Je velmi jednoduchý na implementaci, ale není tak dobrý jako ostatní ochlazovací rozvrhy [19].

2. Logaritmický ochlazovací rozvrh:

$$t_{k+1} = t_k / \log(1 + k).$$

   Logaritmický ochlazovací rozvrh má dobré teoretické vlastnosti, ovšem pro praktické použití je příliš pomalý [19].

3. Exponenciální ochlazovací rozvrh:

$$t_{k+1} = t_k \cdot \alpha,$$

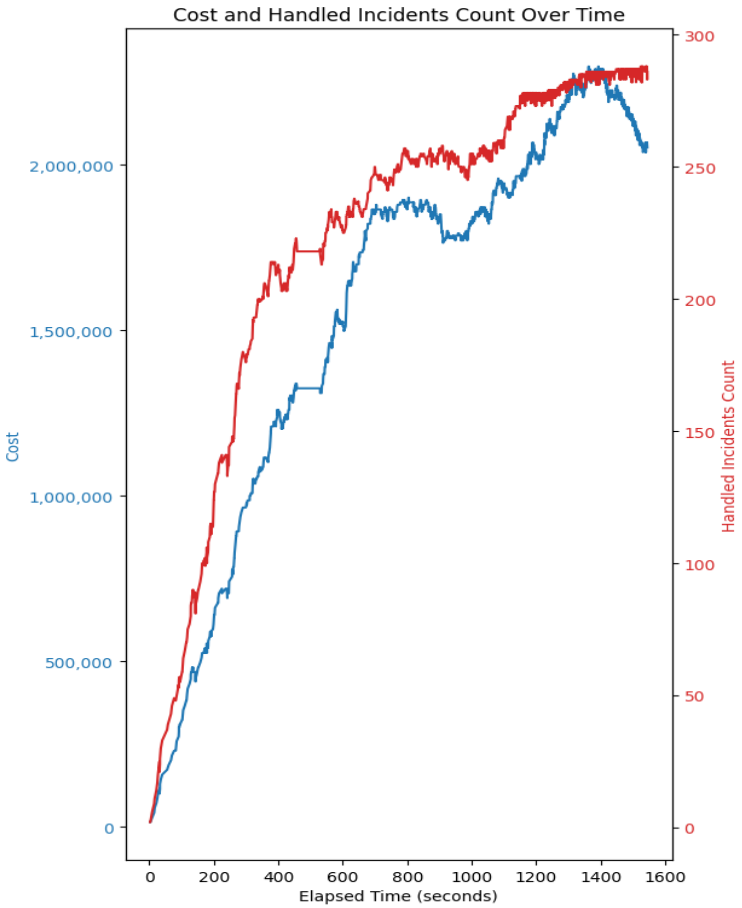
   kde α se často volí mezi 0.8 a 0.99 [18]. Podobně jako předchozí ochlazovací rozvrhy je velmi jednoduchý na implementaci, a často bývá dobrou první volbou, protože umí najít dobrý balanc mezi explorační fází, kdy je teplota vyšší a mezi ladící fází, kdy je teplota nižší [18].

4. Adaptující se ochlazovací rozvrh. Algoritmus si sám v průběhu nastavuje teplotu podle doposud nalezených konfigurací. Náročnější na implementaci než předchozí ochlazující rozvrhy. Hodí se uvažovat o jeho použití především, pokud jednodušší ochlazující rozvrhy nalézají pouze suboptimální řešení.

My si jako ochlazovací rozvrh zvolíme exponenciální ochlazovací rozvrh, především pro jeho jedhoduchou implementaci a pěkné vlastnosti. Parametr α určující rychlost snižování teploty zatím zvolíme jako 0.99.

Nelze obecně říct, kolik iterací M_k provést v rámci stejné teploty. Menší počet iterací může vést k předčasné kovergenci, ale na druhou stranu je výpočet méně náročný. [18]. Zvolme zatím pouze jednu iteraci, čili $M_k = 1$. Samozřejmě, že ideální je použít α rovno 0.99 a M_k velmi vysoké, například v rámci stovek až tisíců, ovšem simulované žíhání s takto zvolenými parametry je velmi výpočetně náročné a běh by trval příliš dlouho. Z toho důvodu je vhodné balancovat M_k s α, pro zajištění rozumné výpočetní náročnosti.

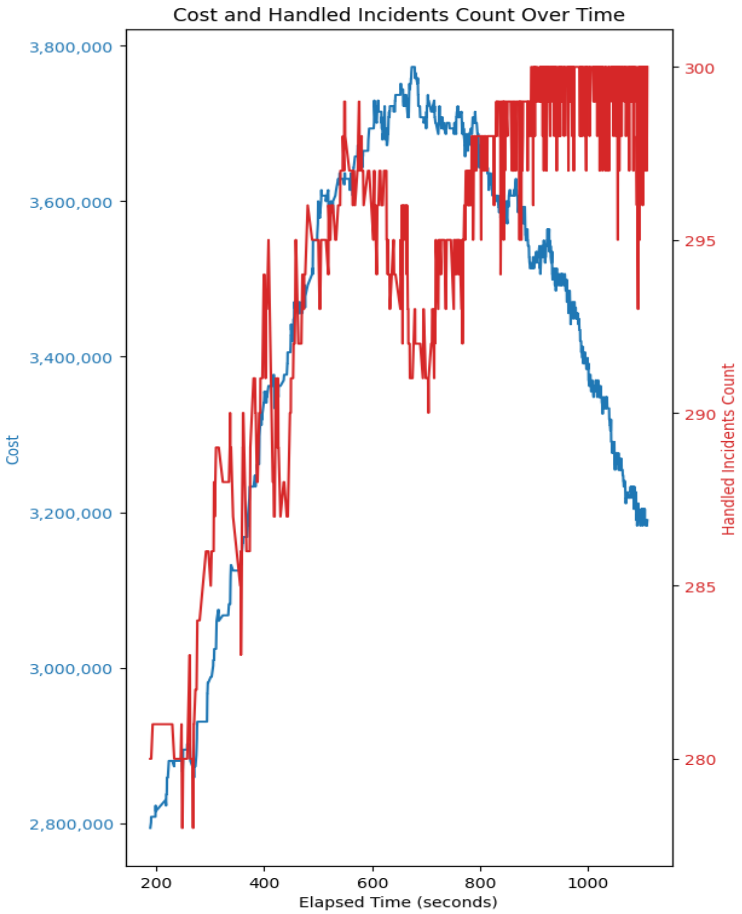
Jako první spustíme simulované žíhání z prázdného plánu.



Obrázek 3.8 Simulované žíhání z prázdného plánu.

Na grafu 3.8 vidíme, jaké plány simulované žíhání s počáteční teplotou 5 stupňů, finální teplotou 10^{-6}, exponenciálním ochlazovacím rozvrhem s $\alpha = 0.99$ a s $M_k = 1$, v průběhu z prázdného plánu navštěvovalo. Celkově běželo simulované žíhání 25 minut a navštívilo celkem 1779 plánů. Nejlepší nalezený plán odbavuje 288 incidentů, stojí 2059332 a má naalokovaných 93 týmů a 132 záchranných vozidel. V porovnání s lokálním nebo tabu prohledáváním z prázdného plánu se jedná o dramaticky lepší výsledek.

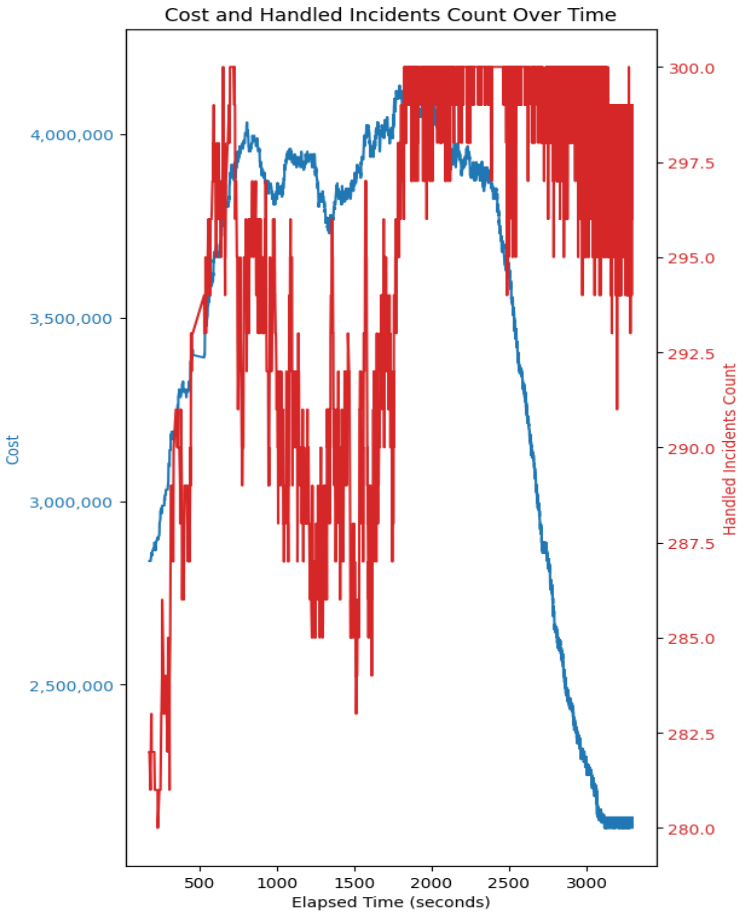
Podobně jako u předchozích metod můžeme zkusit spustit simulované žíhání z jiného než z prázdného plánu. Zkusme prvně spustit z uniformě náhodně vygenerovaného plánu, který má naalokovaných zhruba 50% týmů a vozidel.



Obrázek 3.9 Simulované žíhání z náhodného plánu.

Na grafu 3.10 vidíme chování simulovaného žíhání s počáteční teplotou 5 stupňů, finální teplotou 10^{-6}, exponenciálním ochlazovacím rozvrhem s $\alpha = 0.85$ a s $M_k = 10$. Přibližně do 12 minuty (800 sekund) lze vidět explorační fázi, kdy následně už je plán pouze lokálně vylepšován. Spustit simulované žíhání z náhodného plánu přináší lepší výsledky než z plánu prázdného. Nejlepší nalezený plán odbavuje všech 300 incidentů, stojí 3182531 a má naalokovaných 139 týmů a 126 vozidel.

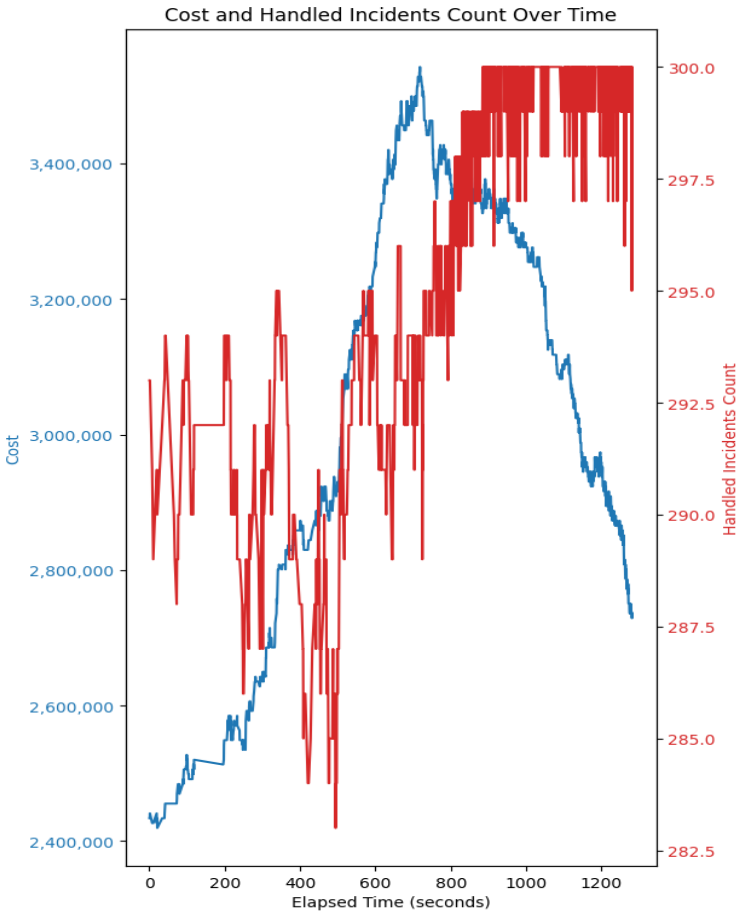
Z grafu podle klesající ceny můžeme usoudit, že by simulované žíhání bylo schopné najít ještě lepší plán, pokud by mohlo běžet déle. Spusťmě proto simulované žíhání znovu ze stejného náhodného plánu, tentorkát ale s jinými parametry.



Obrázek 3.10 Simulované žíhání z náhodného plánu.

Na grafu 3.10 vidíme chování simulovaného žíhání s počáteční teplotou 3 stupňů, finální teplotou 10^{-8}, exponenciálním ochlazovacím rozvrhem s $\alpha = 0.90$ a s $M_k = 30$. Finální teplota je snížena, aby se prodloužila ladící fáze. Zároveň je snížena rychlost klesání teploty a zvýšen počet iterací v rámci stejné teploty, takže sice bude výpočet trvat déle, ale zato by měl být nalezený optimální plán lepší. Skutečně, optimální nalezený plán odbavuje všech 300 incidentů, stojí 2124103 a má naalokovaných 94 týmů a 104 vozidel. To je dobré zlepšení, i za cenu třikrát delšího běhu programu.

Zkusme ještě spustit simulované žíhání z plánu nalezeného optimálními tahy, podobně jako u lokálního a tabu prohledávání.



Obrázek 3.11 Simulované žíhání z plánu nalezeného optimálními tahy.

Na grafu 3.11 vidíme průběh výpočtu. Podobně jako v případě procházení z náhodného plánu jsou při vyšší teplotě navštěvovány horší plány v rámci explorace a od přibližně 10 minuty (700 sekund) se postupně přechází do ladící fáze, která se chová podobně jako lokální prohledávání. Program celkově běžel něco pod 30 minut i spolu s nalezením plánu optimálními tahy. Nejlepší nalezený plán odbavuje 300 incidentů, stojí 2736133 a má naalokovaných 108 týmu a 133 vozidel.

3.7 Porovnání metod

V předchozích kapitolách jsme zformalizovali problém nalezení optimálního plánu záchranné služby a následně jsme navrhnuli několik metod řešící tuto úlohu. Nalezené metody jsme aplikovali na model pražské záchranné služby a náhodně vygenerované sadě incidentů odpovídající, jak by se incidenty v Praze mohli reálně odehrávat. V této kapitole shrneme klíčová pozorování z předešlého aplikování jednotlivých metod a následně mezi sebou metody porovnáme.

Při analýze metody prohledávání plánů optimálními tahy jsme zjistili, že metoda je schopna poměrně rychle (7 minut) nalézt velmi kvalitní plán. Lokální prohledávání z prázdného plánu je horší, a neumí z prázdného plánu nalézt podobně dobrý plán ani do hodiny. Přednost lokálního prohledávání je totiž v

schopnosti doladit nějaký už dost dobrý plán. Kombinací lokálnílo prohledávání a prohledávání optimálními tahy jsme získali metodu, která umí do 20 minut nalézt plán odbavující všechny incidenty a do téměř hodiny a půl dokonce lokálně optimální plán.

Tabu prohledávání není vhodnou metodou pro řešení našeho problému, protože je obtížné najít i jenom lokální optimum. Nevyužije se tak lokální paměť, akorát se zbytečně kontroluje, zda neobsahuje sousední plán. Simulované žíhání, narozdíl od tabu prohledávání, je vhodná metoda a podobně jako u lokálního prohledávání nejlépe funguje v kombinaci s metodou prohledávání optimálními tahy. Narozdíl od lokálního nebo tabu prohledávání, simulované žíhání neprozkoumává všechny sousední plány, a z toho důvodu konverguje o dost rychleji. Proto velmi dobře funguje i při prohledávání z náhodně vygenerovaného plánu.

Jakou konkrétně metodu použít velmi záleží na cíli, kterého chceme dosáhnout. Pokud je žádoucí nalézt skutečně co nejlepší plán, odbavující nějakou danou sadu incidentů, tak je nejlepší zvolit kombinaci prohledávání optimálními tahy spolu s lokálním prohledáváním. Ta nalezne velmi kvalitní plán, ale trvá poměrně dlouho. Pokud je žádoucí nalézt dostatečně dobrý plán za kratší dobu, je lepší volbou spustit metodu prohledávání optimálními tahy spolu se simulovaným žíháním. Simulované žíhání je rychlejší a bylo schopné nalézt kvalitní plán z plánu optimálního v ceně do 20 minut. Přičemž při nižší finální teplotě by simulované žíhání konvergovalo k ještě lepšímu plánu, samozřejmě za cenu delšího běhu.

Nevýhodou prohledávání optimálními tahy je, že umí používat jenom účelovou funkci q^{Lex}. Pokud bychom chtěli používat jinou účelovou funkci, například proto, že chceme nalézt plány, které odbavují jen o něco méně incidentů, ale jsou o dost levnější, nemusí být prohledávání optimálními tahy vhodné použít. V takovém případě je nejlepší z prázdného plánu nalézt dost dobrý plán simulovaným žíháním a následně nalézt jeho lokální optimum pomocí lokálního prohledávání.

Metoda	**Čas běhu**	**Cena**	**Odbavené incidenty**
N	23:46	3600088	292
POT	10:53	2469661	296
POT + LP	32:52	2512864	300
POT + LP	1:38:22	2188863	300
SŽ – náhodný plán	51:54	2124103	300
POT + SŽ	21:15	2736133	300

Tabulka 3.2 Shrnutí metod. N značí naivní řešení, POT značí prohledávání optimálními tahy, LP značí lokální prohledávání, SŽ značí simulované žíhání.

Pražská záchranná služba už je poměrně větší záchranná služba a spolu s použitím Google API a větší sady incidentů se jednalo o výpočetně dlouho trvající úlohu, avšak i tak navrhnuté metody, pokud jsou správně použity, umí nalézt velmi kvalitní plány v průměru do hodiny (viz tabulka 3.2).

Závěr

Tato práce se zabývá problémem nalezení optimálního plánu pohotovostní služby. V první kapitole jsme problém zformalizovali a důkladně zanalyzovali, abychom zvolili vhodné metody řešení. Ukázalo se, že nejvhodnější je problém modelovat jako optimalizační úlohu s jednou účelovou funkcí. Účelová funkce je vhodným složením ceny plánu a počtu úspěšně odbavených incidentů. Abychom věrohodně zjistili počet úspěšně odbavených incidentů, spustili jsme diskrétní simulaci, která pro danou sadu incidentů napodobuje chování plánu v průběhu jednoho dne. Použití simulace omezilo možné techniky, které se běžně používají pro řešení těžkých optimalizačních problému, jako například lineární programování. Na druhou stranu, umožňuje věrohodně získat počet úspěšně odbavených incidentů, stejně jako by plán již skutečně odbavoval incidenty v terénu.

V druhé kapitole jsme diskutovali několik různých metod, od využití dynamického programování, až po použití metaheuristických přístupů. Nalezli jsme rekurzivní vztah mezi plány odbavující o jedna méně incidentů a využili jej pro řešení úlohy pomocí dynamického programování. Přestože výpočetní složitost je exponenciální vůči počtu incidentů, jedná se o dramatické zlepšení oproti naivnímu řešení, které jenom náhodně vytváří plány.

Ve třetí kapitole jsme nalezené metody aplikovali na konkrétní pohotovostní službu, a to sice na pohotovostní službu hlavního města Prahy. Synteticky jsme vygenerovali sadu incidentů, která vhodně reprezentuje možnou skutečnou situaci. Na základě volně dostupných dat pražské pohotovostní služby představuje, jak by se mohly incidenty v průběhu dne skutečně odehrát.

Při aplikaci metod jsme zjistili, že některé metody jsou vhodnější než jiné. Například se ukázalo, že nemá smysl používat tabu prohledávání, jelikož je obtížné nalézt i jedno lokální optimum. Na druhou stranu, kombinace lokálního prohledávání nebo simulovaného žíhání spolu s dynamickým programováním se ukázaly jako nejlepší metody pro praktické využití. Nalezené plány uměly úspěšně odbavit všechny incidenty jen s polovinou vozidel a do stovky týmů záchranářů. Nalezené plány jsou kvalitní, a jistě by se uplatnily pro praktické využití.

Přestože umíme nalézt velmi kvalitní plány pohotovostní služby na dané sadě incidentů, lze práci do budoucna rozšířit hned o několik věcí. Především by bylo zajímavé více prozkoumat, jak vypadají optimální plány, které by odbavovaly jen o něco méně incidentů, ale byly by výrazně levnější. Takové plány by se daly hledat pomocí vhodně zvolené účelové funkce, která by poskytovala dobře nastavený kompromis mezi cenou a počtem odbavených incidentů.

Dalším zajímavým rozšířením by bylo přidat incidentům typ, který by určoval, jaká záchranná vozidla a týmy by mohly incident odbavit. Jednotlivé týmy nebo vozidla by byly různě drahé, právě podle toho, jaké typy incidentů by mohly odbavovat. Implementace takového rozšíření vyžaduje pouze úpravu simulace a přidání nových tahů u metod, které využívají sousedství.

V neposlední řadě by bylo zajímavé zkoumat, jak se nalezené optimální plány chovají v krajních případech, kdy se například najednou stane velké množství incidentů na jedné lokalitě, nebo na velmi odlehlé lokalitě, případně i ve stejný čas. Aby bylo možné takové plány hledat, je potřeba, aby plány uměly dobře generalizovat. Metody by tak musely krajní případy uvažovat už v rámci hledání

a předem alokovat vozidla a týmy, i když by většinu času nebyly potřeba. To nás vede na další rozšíření, kdy by bylo dovoleno, aby týmy naalokované na jedné výjezdové stanici mohly pomáhat v krajních případech týmům na jiných stanicích. Tomuto mechanismu se říká *dynamic dispatch* a v krajních případech by se mohlo jednat o nejefektivnější způsob, jak zdroje rozmisťovat.

Literatura

1. Vanderbei, Robert J. *Linear programming*. New York City: Springer International Publishing, 1996. ISBN 9783030394158.
2. Udo W. Pooch, James A. Wall. *Discrete Event Simulation: A Practical Approach*. US, Florida: CRC PRess, 1992. ISBN 9780849371745.
3. *Logis Solutions*. [B.r.]. Dostupné také z: `https://logissolutions.net`.
4. Mykel J. Kochenderfer, Tim A. Wheeler. *Algorithms for Optimization*. Cambridge, Massachusetts, United States: MIT Press, 2019. ISBN 9780262351409.
5. Jiří Matoušek, Jaroslav Nešetřil. *Kapitoly z diskrétní matematiky*. Prague: Karolinum, 2002. ISBN 80-246-0084-6.
6. Applegate, David L; Cook, William J; Dash, Sanjeeb; Johnson, David S. A Practical Guide to Discrete Optimization. In: *Chapter 1, Draft of 7 August 2014*. Princeton University Press, 2014.
7. Byrne, Charles L. *Continuous Optimization*. University of Massachusetts Lowell, 2013.
8. Cohen, Michael B.; Lee, Yin Tat; Song, Zhao. *Solving Linear Programs in the Current Matrix Multiplication Time*. 2020. Dostupné z arXiv: `1810.07896 [cs.DS]`.
9. Bellman, Richard. *Dynamické programování*. Princeton, US: Princeton University Press, 2010. ISBN 978-0691079516.
10. Martin Mareš, Tomáš Valla. *Průvodce labyrintem algoritmů*. Prague: CZ.NIC, 2017. ISBN 978-80-88168-19-5.
11. Michel Gendreau, Jean-Yves Potvin (ed.). *Handbook of Metaheuristics*. Boston, MA, USA: Springer Cham, 2019.
12. Eiben, Agoston E; Smith, James E. *Introduction to evolutionary computing*. Sv. 53. Springer, 2015. Natural Computing Series.
13. Dorigo, Marco; St"utzle, Thomas. *Ant colony optimization*. MIT press, 2004.
14. Christian Blum, Günther R. Raidl. *Hybrid Metaheuristics*. Boston, MA, USA: Springer Cham, 2016.
15. Glover, Fred W; Laguna, Manuel. *Tabu search*. Springer Science & Business Media, 1997. Dostupné z DOI: `10.1007/978-1-4615-6089-0`.
16. Laarhoven, Peter J.M.; Aarts, Emile H.L. *Simulated Annealing: Theory and Applications*. Sv. 37. Springer Netherlands, 1987. Mathematics and Its Applications. Dostupné z DOI: `10.1007/978-94-015-7744-1`.
17. Metropolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; Teller, E. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*. 1953, roč. 21, č. 6, s. 1087–1092. Dostupné z DOI: `10.1063/1.1699114`.

18. Delahaye, Daniel; Chaimatanan, Supatcha; Mongeau, Marcel. Simulated Annealing: From Basics to Applications. In: *Handbook of Metaheuristics*. Springer, Cham, 2018, s. 1–35.

19. Green, David; Aleti, Aldeida; García, Julián. The Nature of Nature: Why Nature-Inspired Algorithms Work. In: 2017, s. 1–27. isbn 978-3-319-50919-8. Dostupné z doi: 10.1007/978-3-319-50920-4_1.

A Přílohy

A.1 soubor README.md

Obsahuje uživatelskou a technickou dokumentaci.

A.2 soubor exampleGantt

Soubor znázorňuje chování nalezeného plánu na dané sadě incidentů. Popis formátu lze nalézt uživatelské dokumentaci.

A.3 adresář img

Adresář obsahuje obrázky, které jsou používány v uživatelské dokumentaci.

A.4 adresář src

Obsahuje implementaci všech metod diskutovaných v této práci, spolu s testy.