ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Trmal** Jméno: **Jakub** Osobní číslo: **457118**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Studijní obor: **Informatika a počítačové vědy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Multi-agentní plánovač a mobilní služba pro ad-hoc přepravu balíků

Název bakalářské práce anglicky:

Multi-Agent Planner and Smart Phone Service for Ad-Hoc Parcel Logistics

Pokyny pro vypracování:

Student naimplementuje multi-agentní A* algoritmus jako součást systému klient-server pro ad-hoc přepravu balíků. Serverová část systému bude poskytovat Rest API pro multi-agentní plánovač a klientská strana umožní uživatelům přidávat nové požadavky na přepravu balíků. Klientská část bude implementována jako aplikace pro mobilní platformy (telefony/tablety). Implementovaný systém bude odsimulován na modelu pražské silniční sítě.
1) Nastudujte literaturu v oblasti multi-agentního plánování, distribuovaných heuristických prohledávacích algoritmů, jmenovitě multi-agentní A* a distribuovaných heuristik.
2) Modifikujte existující platformu pro simulaci dopravy ve městech pro přepravu balíků. Následující algoritmy vyviňte pro a otestujte v daném simulátoru.
3) Implementujte multi-agentní A*, vč. různých distribuovaných heuristik (např. varianty Manhattanské a Eukleidovské heuristiky, případně relaxovaných heuristik)
4) Vyviňte aplikaci pro mobilní telefony/tablety pro zadávání nových požadavků na převoz balíků do systému.
5) Experimentálně analyzujte výkonnost vyvinutého systému. Zaměřte se na rychlost plánování a replánování daného plánovacího algoritmu a použitých heuristik.

Seznam doporučené literatury:

[1] Malik Ghallab, Dana S. Nau, Paolo Traverso: Automated planning - theory and practice. Elsevier 2004, ISBN 978-1-55860-856-6.
[2] Stefan Edelkamp, Stefan Schroedl: Heuristic Search: Theory and Applications. Morgan Kaufmann. 2012, ISBN 978-0-12372-512-7.
[3] Raz Nissim, Ronen I. Brafman: Distributed Heuristic Forward Search for Multi-agent Planning. J. Artif. Intell. Res. 51: 293-332 (2014)

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Antonín Komenda, Ph.D., centrum umělé inteligence FEL

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

Datum zadání bakalářské práce: **09.01.2018** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Antonín Komenda, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

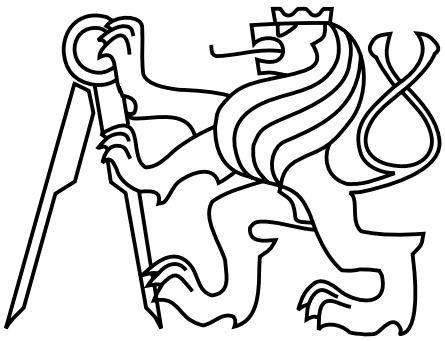
III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra kybernetiky



Bakalářská práce

Multi-agentní plánovač a mobilní služba pro ad-hoc přepravu balíků

Jakub Trmal

Vedoucí práce: Ing. Antonín Komenda, PhD.

Studijní program: Otevřená informatika, Bakalářský

Obor: Informatika a počítačové vědy

25. května 2018

Poděkování

V první řadě bych chtěl poděkovat svému vedoucímu bakalářské práce Ing. Antonínu Komendovi, PhD. za odborné vedení, za pomoc a za odborné rady při zpracování této práce. Za odborné rady a konzultace také děkuji Zdeňku Boušovi. Velké poděkování dále nepochybně patří blízké rodině a mé přítelkyni za stálou podporu. V neposlední řadě děkuji spolužákům za udržení týmového ducha a konzultace do pozdních nočních hodin.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. května 2018 .....

Abstract

This work is focused on the way to utilize Artificial Intelligence techniques to solve concrete parcel transport problem. Let us have a local transportation company which owns a certain number of cars and which is looking for an effective way of transporting goods in the region. Using techniques of multi-agent cooperation we can find a solution to create efficient automated delivery while optimizing traveled distances or spent fuel. Practical result of this work is a server application which is able to solve given logistic tasks and simulate results visually. This application is also prepared to provide restful API which receive requests from other client devices. We also provide a mobile application as an example of client device connection.

Abstrakt

Práce se zaměřuje využití technik umělé inteligence na konkrétní problematice přepravy zásilek. Zaměříme se přepravce, který vlastní vozový park a řeší jakým způsobem efektivně přepravovat zboží s možností vyzvedávat zásilky kdekoliv v daném regionu. Pomocí technik multi-agentní kooperace se dá takový problém efektivně automatizovat a zároveň optimalizovat s ohledem na minimalizaci ujetých kilometrů nebo úsporu paliva. Praktickým výstupem je prototyp volně konfigurovatelné serverové aplikace, která dokáže řešit zadaný logistický problém a jeho řešení podle potřeb odsimulovat. Pro potřeby klienta je vytvořeno komunikační rozhraní přijímající požadavky na přepravu balíků, které je možné odesílat z poskytované mobilní aplikace.

Key words: multi-agent planning, automated solver, parcel transport, artificial intelligence, application, Java

Klíčová slova: multi-agentní plánování, automatizovaný plánovač, přeprava zásilek, umělá inteligence, aplikace, Java

Obsah

Seznam obrázků

Kapitola 1

Úvod

V dnešním globalizovaném světě se ve velkém přepravuje různé zboží ať už na krátké či dlouhé vzdálenosti. Klasická přepravní síť funguje na principu předávání zásilek mezi velkými dopravci a následném rozvozu zboží lokálními přepravci až ke koncovému zákazníkovi. Ve většině případů klade tento proces nároky na způsob komunikace, rychlost komunikace a vzájemné předání konkrétních zásilek mezi dopravci. V těchto místech interakce je prostor k možnému zrychlení.

Cílem této bakalářské práce je vytvoření serverové aplikace, která bude řešit problematiku přepravy balíků mezi klienty. Myšlenkou je automatizace logistiky přepravy jednorázových klientských zakázek. Zakázka není vázána místem podání na určitém sběrném místě, ale pracuje s vyzvednutím zakázky přímo u zákazníka a s dopravou do předem určeného místa.

Prakticky není realizovatelné aplikaci otestovat v reálném nasazení, a proto se práce zaměřuje na syntetickou simulaci systému, kterou bude zajišťovat projekt Agentpolis [1] vyvíjený pod Fakultou elektrotechnickou. Pro aplikaci jsou vyvinuty modelové situace, které se dají s využitím modulu Agentpolis vizualizovat.

Další částí práce je experimentální sledování chování systému při různých změnách nastavení parametrů. Pozoruji chování systému pro různé konfigurace počátečních rozmístění agentů po mapě, vliv množství agentů na kooperaci nebo vliv množství agentů na rychlost nalezení řešení zadaného problému. V neposlední řadě je u heuristických vyhledávacích systémů důležitá volba heuristické funkce. Součástí práce je také několik experimentů s různými

heuristikami.

1.1 Motivace

V současné době získávají na popularitě distribuované systémy, které využívají odlišným způsobem moderní technologie. Vezměme si například přepravu lidí pomocí služeb taxi. Služby taxi byly do nedávna převážně centrálně řízené určitou formou dispečinku. Jednoduše řečeno řidič taxi je na směně a dostává zakázky z dispečinku, kde se sbírají požadavky zákazníků. Má rovnou určeno koho kam veze a jeho úkolem je tento požadavek vyřídit.

Za posledních pár let však začali být velice populární distribuované služby, které v Praze nabízí například Liftago. Poskytuje systém, do kterého se uživatel zaregistruje a když chce někam odvézt, tak jednoduše dostane možnost zvolit si z nabídky řidičů v okolí a následně se od něj nechá odvézt. V tomto modelu je pouze systém zjišťující polohu zákazníka, kterému poté poskytne výběr řidiče, a zbytek komunikace klient-služba již probíhá distribuovaně pro každý případ.

Pokud systém podobné myšlenky sklízí mezi lidmi takový úspěch, bylo by možné využít totožný model také pro přepravu zásilek? Pokusíme se insprirovat tímto pozorováním a vytvořit prototyp systému, který by dokázal řešit pár vzorových případů ve zjednodušených podmínkách. Několik experimentů by mohlo ukázat, jak je reálné tuto myšlenku využít a co případně upravit nebo změnit, aby byl systém schopen reálného provozu.

Kapitola 2

Model problému

V této kapitole se zaměříme na způsob, jakým je možné logistický problém modelovat. Model bude představovat reprezentaci zadaného problému a při dobré volbě nám umožní problém konfigurovat na různé požadavky. Správná reprezentace problému je důležitá hlavně ve fázi automatizovaného řešení a při implementaci. Automatizovanému solveru, který bude detailněji probrán v kapitole 3, by měl model poskytnout nástroje, pomocí kterých bude mít solver možnost modifikovat konfiguraci světa a tím hledat řešení.

Definice 1. V další části textu budeme tuto kompletní n-tici označovat jako *systém*.

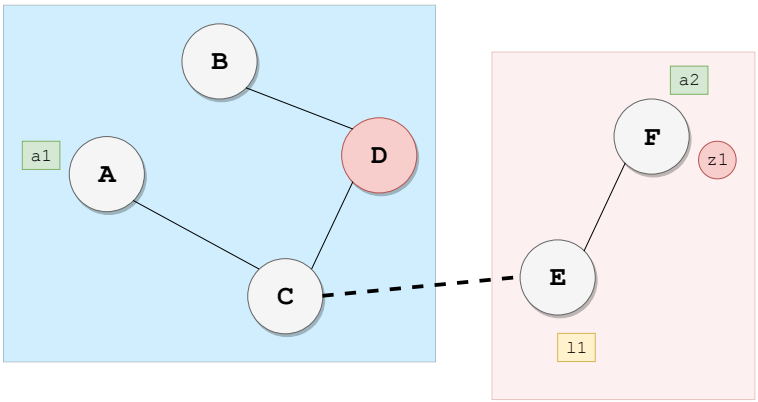
Obecná definice logistického problému pro množinu agentů $\Phi = \{\rho_{i=1}^{k}\}$, $k > 1$ je zadána n-tice $\langle P, \{A_i\}_{i=1}^{k}, I, G\rangle$ [12].

- P je množina faktů, kterých je systém schopen nabýt působením libovolných akcí. Tato množina je dále definována jako dvojice $\langle entita, pozice\rangle$.
- A_i je množina akcí, které je schopen agent ρ_i aplikovat. Každá akce je dále definovaná jako n-tice $\langle pre(a), add(a), del(a)\rangle$ postupně zleva definovaných jako $pre(a)$ - předpokládaná fakta pro aplikování dané akce, $add(a)$ - efekt, kterým změní stav systému tím, že přidá nějaký, fakt a nakonec $del(a)$ - efekt, který určité fakty ze stavu systému odebere
- I je podmnožina $I \subseteq P$, která reprezentuje inicializační stav systému

- G je podmnožina $G \subseteq P$. Tato množina reprezentuje cílový stav, do kterého by se systém aplikováním možných akcí měl dostat.

2.1 Finite-domain representation

Finite-domain representation nebo zkráceně FDR je způsob, kterým reprezentujeme stav objektů systému [10]. Pro každý z objektů, v našem případě zásilka nebo agent, je definována množina možných výskytů objektu v systému. Další z možností, jak modelovat plánovací problém, je STRIPS. STRIPS představuje zkratku pro Stanford Research Institute Problem Solver a jedná se také o způsob reprezentace na úrovni FDR [10]. Rozdíl mezi těmito dvěma způsoby popisu problému je v pohledu na zápis faktu. U STRIPS máme množinu všech faktů, které v systému mohou nastat, a k nim přiřazujeme pouze binární hodnoty 1 (platí) či 0 (neplatí). Z toho plyne, že STRIPS reprezentace má zpravidla opravdu velký stavový prostor [7]. FDR pohlíží na naši problematiku o něco úsporněji, přičemž využívá konečnou množinu faktů pro jednotlivé entity. U logistické plánovací úlohy můžeme s jistotou říct, že pokud se agent nachází na pozici A, pak nemá smysl ohodnocovat výskyt agenta na pozici B, protože se zde určitě nacházet nebude. V některých případech tak může dojít skokově k redukci množství stavů celého stavového prostoru.



Obrázek 2.1: Příklad jednoduchého plánovacího úkolu. a_i jsou automobily, l_1 je letadlo, z_1 je zásilka v uzlu F s cílem dostat se do uzlu D

Jak můžeme vidět z malého příkladu na obrázku 2.1, máme zde dva typy agentů a jednu

zásilku. Viditelné jsou dále dvě komponenty souvislosti, jedna $\{A, B, C, D\}$ je spojená s druhou $\{E, F\}$. Jejich propojení je graficky vyznačeno přerušovanou čarou. Spojnice mezi uzly C a E označuje cestu, která je určená pouze pro leteckou dopravu. Uvnitř obou komponent je poté standartní způsob dopravy pomocí automobilů. Finite-domain reprezentace pro takto graficky definovaný plánovací úkol by poté vypadala následovně:

$$
\begin{aligned}
a_1 &\in \{\text{at–A},\ \text{at–B},\ \text{at–C},\ \text{at–D}\} \\
a_2 &\in \{\text{at–E},\ \text{at–F}\} \\
l_1 &\in \{\text{at–C},\ \text{at–E}\} \\
z_1 &\in \{\text{at–A},\ \text{at–B},\ \text{at–C},\ \text{at–D},\ \text{at–E},\ \text{at–F}\}
\end{aligned}
$$

Obrázek 2.2: Finite-domain reprezentace stavu světa z obrázku 2.1

Grafická reprezentace z ukázky 2.2 je podobně intuitivní jako grafické provedení.

2.1.1 Efektivita FDR

Finite-domain reprezentace je výhodnou volbou z několika pohledů. Při implementaci zvyšuje efektivitu algoritmu a přehlednost zdrojového kódu. Nespornou výhodou však tato reprezentace je v porovnání se STRIPS při využití v plánovacích úlohách, které mívají obsáhlý kauzální graf [10] [9]. Tedy stavový graf, ve kterém jednotlivé uzly často mají jisté předpoklady od svého rodiče. Například ve chvíli, kdy expandujeme další stav, ve kterém by měl agent naložit zásilku akcí *load*, vyžadujeme, aby se zásilka nacházela na stejném místě jako agent, který ji má naložit.

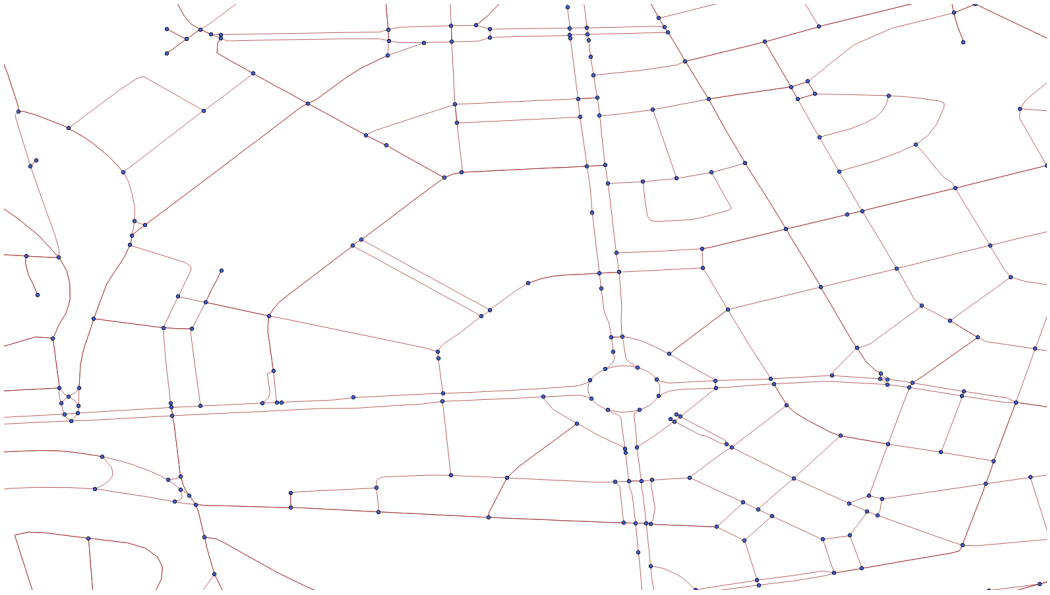
Z definice FDR 2.2 jsou také dobře viditelné vlastnosti objektů. Na první pohled je dobře čitelné, že letadlo l_1 se může pohybovat pouze mezi uzly C a E. Nejspíš se bude jednat o dvě města, která jsou mezi sebou dostupná pouze letadlem, protože přesto, že jsou a_1 a a_2 stejný typ agenta, tak nemají dostupné stejné uzly. Tyto fakta se do FDR reprezentace dobře promítnou.

2.2 Grafový model

Grafový model bude představovat prostor, ve kterém se bude veškerý děj odehrávat. V našem případě logistického problému jde o silniční síť a tedy maximálně souvislý orientovaný graf G, který je reprezentován jako dvojice (V, E).

- V je množina všech vrcholů grafu G
- E je množina všech orientovaných hran $e(v_1, v_2)$, $v_1, v_2 \in V$ vedoucích z vrcholu v_1 do vrcholu v_2

Víme, že každá hrana grafu je ohodnocena ať už záporně či kladně. Těmto hodnotám říkáme cena. Konkrétně v případě hledání nejkratší cesty v grafu bude cena hrany reprezentovat vzdálenost uraženou na jedné hraně, půjde tedy vždy o kladnou hodnotu. Vrcholy ve zjednodušeném grafu budou reprezentovat křižovatky a orientované hrany mezi nimi budou cesty, po kterých se agent může pohybovat. Přípravě a získání mapových podkladů se v užším pohledu věnuje kapitola sekce o preprocessingu 4.1.1.



Obrázek 2.3: Ukázka části pražské silniční sítě - Dejvický kruhový objezd

Agent se v silniční síti pohybuje primárně po vrcholech, ale pro účely plánovače bude model schopen zachytit výskyt agenta i „na cestě“ mezi dvěma uzly. Jak již bylo uvedeno, tak

graf, nad kterým bude plánování probíhat, je model reálné podoby silniční sítě. Pohybujeme se tedy v Euklidovském prostoru, kde je možné každý bod reprezentovat jednoznačnými souřadnicemi. Takovýto prostor má zároveň dobře známé vlastnosti, jako je například skalární součin nebo metrika, kterých budeme dále využívat.

Na obrázku 2.3 můžeme vidět vizualizovaný výřez z mapových podkladů. Data jsou určitým způsobem abstrahována, aby se omezil nárok na kapacitu uložiště a zároveň byla podpořena efektivita plánovacího procesu. [6] Touto úpravou se ztrácí v přijatelné míře detailnost vizualizace.

Kapitola 3

Návrh řešení

Mějme logistický problém v přepravě zásilek, který je možné řešit různými technikami. Motivací celé práce je automatizace logistického plánování pro společnost zabývající se přepravou zásilek. Volíme techniku multi-agentního plánování, která kombinuje standartní plánování v grafové struktuře a kooperaci více entit najednou. Tato technika pracuje na principu rozdělení komplexního problému na menší subproblémy, jež jsou snáze řešitelné pro jednotlivé entity. Zároveň multi-agentní pohled lépe reflektuje reálnou podobu logistického plánování, které se v praxi řeší více než jedním nákladním autem.

3.1 Obecné definice

Pro reprezentaci takového modelu v počítači je nutné problém uvést formálně. V našem případě budou entity spolupracovat mezi sebou, jde tedy o plně propojený model. Zadefinujeme si jednotlivé entity, se kterými bude plánovací algoritmus pracovat.

Definice 2. *Agent* ρ_i je entita, která se pohybuje v systému a svými akcemi má schopnost tento systém ovlivňovat. Agent, který je v našem případě vozidlo, je formálně definován množinou $\rho_i = \{A_i, O_i, T_i\}$.

- O_i je výchozí poloha agenta
- T_i je cílová poloha agenta

- A je množina možných akcí agenta.

Definice 3. *Objekt/zásilka* Entita, která se stejně jako agent pohybuje ve světě a určitým způsobem jej ovlivňuje. Aktivně se akcemi nepodílí na ovlivnění stavu, ale pasivně se nachází na určitém místě a má za úkol se přesunout na jiné místo.

V textu se dále bude vyskytovat důležitá vlastnost agenta, pomocí které je celý systém schopen se modifikovat.

Definice 4. *Akce* je následující n-tice funkcí $\langle pre(a), add(a), del(a) \rangle$, kde $a \in P$

- $pre(a)$ je předpoklad pro aplikaci akce v daném stavu a. Pokud tento předpoklad není splněn, akci v daném stavu ignorujeme
- $add(a)$ je funkce, která změní stav a dle toho jak má předdefinováno tím, že do něj přidá fakta
- $del(a)$ je podobná fuknce jako $add(a)$ s tím rozdílem, že ze stavu a fakta odebírá

3.1.1 Private a public akce

Akce, které jsou jednotliví agenti schopni aplikovat, se dále dělí do dvou skupin. Toto dělení je rozlišováno tím, zda je po aplikaci akce $a_i \in A$ agentem $\rho_i \in \Phi$ ovlivněn stav světa s i z pohledu agenta ρ_j. Soukromí je důležitým aspektem multi-agentního plánování, protože není vždy žádoucí, aby si agenti mezi sebou vyměňovali veškeré informace. [14] Pokud agent ρ_i akcí $a_i \in A$ stav s z pohledu ρ_j ovlivní, pak akci považujeme za **veřejnou** (public) v opačném případě jde o akci **soukromou** (private) [11]. V našem případě jde z tohoto pohledu o jednodušší model, ve kterém si způsob pohybu agent nechává jako soukromý a zbytek informací se mezi agenty sdílí. Při plánování logistické přepravy budeme v našem případě uvažovat pro agenta tyto akce:

```
move(a, x, y) - přesuň agenta a z pozice x na pozici y (private)
load(where, a, c) - na dané pozici naloží agent a zásilku c (public)
unload(where, a, c) - na dané pozici vyloží agent a zásilku c (public)
```

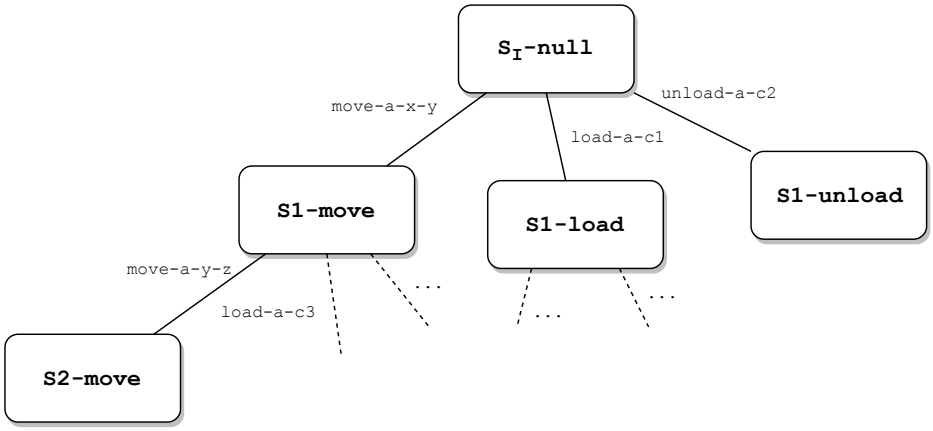
3.2 Algoritmy a heuristiky

V této sekci se podrobněji zaměříme na samotnou podstatu automatizovaného řešení logistického problém, kterou je prohledávací algoritmus. Jeho úkolem je najít efektivní sekvenci akcí, jejichž aplikací je možné vyřešit zadanou úlohu. Hledání řešení bude v tomto případě znamenat prohledávání stavového prostoru pomocí technik algoritmů umělé inteligence. Myšlenka prohledávání stavového prostoru je následující: Agent experimentálně aplikuje akce, které má v daném stavu k dispozici. Tímto způsobem objevuje nové stavy systému a následně pomocí heuristické funkce určuje, jak je pro něho nově nalezený stav výhodný při postupu k nalezení řešení. [6] Formálně si proto zaveďme pojmy, které jsou pro tyto techniky nezbytné.

Definice 5. *Stav.* Mějmě jednu konkrétní povolenou konfiguraci systému. Konfigurací je myšleno konkrétní přiřazení proměnné k faktu z její množiny domén v modelu FDR. Zároveň pokud tato konfigurace vznikla z jiné konfigurace aplikováním kterékoliv akce a agenta ρ_i, pak této konfiguraci říkáme *Stav* s. Jedinou vyjímkou je počáteční stav s_I, jehož vzniku nepředcházela žádná známá akce a je pouze obrazem výchozí konfigurace systému.

Definice 6. *Stavový prostor.* Nechť P je množina všech stavů, které je možné v rámci definice systému vytvořit. Mezi jednotlivými stavy je možné přecházet pomocí aplikace sekvence po sobě jdoucích povolených akcí $\langle a_1, ..., a_n \rangle \in A, n \geq 1$. Tuto množinu je možné reprezentovat jako spojitý graf o jedné komponentě souvislosti. Množinu P poté budeme nazývat *Stavový prostor.*

Pro lepší představu přidáváme grafickou podobu části stavového prostoru viditelného na obrázku 3.1. Jednotlivé vrcholy grafu jsou zároveň jednotlivé stavy systému, do kterých je možné se dostat po hranách, které ve stavovém prostoru představují aplikované akce. Zároveň každý stav zaznamenává informaci o akci jejíž aplikací vznikl. Z popisů jednotlivých stavů je tedy dobře vidět, že s_I je počáteční stav, ve kterém prohledávání začalo. Dále z každého stavu poté probíhá expanze a samotné prohledávání prostoru.



Obrázek 3.1: Ukázka stavového prostoru pro plánovací problém z pohledu agenta $a \in \Phi$

3.2.1 A* algoritmus

Jedním z podproblémů, který vznikne při aplikaci akce *move*, je zjištění nejlepší trasy mezi počátečním a koncovým bodem. Algortimů, které řeší vyhledání nejkratší cesty v grafu, je velké množství.

V našem případě použijeme pro vyhledávání nejkratší cesty v ohodnoceném grafu čili silniční síti populární plánovací algoritmus A*. [6] Tento algoritmus vychází z principů popsaných Dijkstrovým prohledávacím algoritmem a je doplněn o heuristiku, která zajistí efektivitu a také optimalitu řešení. Jedním z hlavních rysů je prioritní fronta, kterou si algoritmus udržuje a podle které se následně rozhoduje, jakým směrem bude pokračovat další expanze. Hodnota neboli priorita, kterou jednotlivý vrchol v danou chvíli dostane, je určena vztahem $f(x) = g(x) + h(x)$, $x \in V$, kde $g(x)$ je cena dostavadní uražené cesty a $h(x)$ je hodnota heuristické funkce pro daný vrchol. Heuristická funkce a její volba je zásadní pro výpočet a chování algoritmu, proto se jí budeme dále věnovat o něco podrobněji.

3.2.2 MA-A*

Prohledávání stavového prostoru je možné několika způsoby a algoritmy. Náš požadavek zadaný modelem je však takový, aby byl systém distribuovaný. Proto volíme distribuovanou multi-agentní verzi A*, která je ve své podstatě rozšířenou variantou výše zmiňovaného algoritmu A*. [12]

Jelikož se jedná o distribuovaný systém, musí algoritmus počítat s tím, že agenti spolu nesdílejí paměť s daty, a proto zavedeme určitý způsob. Je nutné, aby se agenti vzájemně informovali o postupu při prohledávání stavového prostoru. Budou tak činit za pomocí zpráv.

Definice 7. *Zpráva.* Agenti mezi sebou komunikují pomocí *zpráv*. Agent ρ_i vždy vyšle zprávu všem ostatním agentům $\rho_j \in \Phi \setminus \{\rho_i\}$ ve chvíli, kdy expanduje nový stav pomocí public akce. Viz 3.1.1. Zpráva obsahuje stav, kterého agent v daném čase dosáhl, zároveň s hodnotami $g(s)$ a $h(s)$ pro nový stav. Bližší specifikace hodnot v sekci o heuristické funkci 3.2.3.

Algoritmus 1 Hlavní cyklus algoritmu [12]

```
1: while planFound is not true do
2:     for all pro všechny nevyřízené zprávy m čekající ve frontě do
3:         PROCESS_MESSAGE(m)
4:     end for
5:     s ← POP_FROM_OPENLIST
6:     EXPAND(s)
7: end while
```

Algoritmus 2 process message $m = \langle s, g_{\rho_j}(s), h_{\rho_j}(s) \rangle$

```
1: if s není v openListu nebo closeListu or g_ρi(s) > g_ρj(s) then
2:     přidej s do openListu and spočítej nové h_ρi(s)
3:     g_ρi(s) ← g_ρj(s)
4:     h_ρi(s) ← max(h_ρi(s), h_ρj(s))
5: end if
```

Algoritmus můžeme rozdělit do tří důležitých částí. První část výše označenou jako Algoritmus 1 můžeme příznačně popsat jako hlavní cyklus, ve kterém se nejdřív zpracují nově příchozí zprávy od ostatních agentů, následně se expanduje nejlepší stav z pohledu heuristické funkce. To vše ve smyčce dokud se nepodaří najít cílový stav.

Zpracování příchozí zprávy probíhá tak, že v této fázi agent zjišťuje, zda daný stav již někdy expandoval či uzavřel. Pokud je příchozí stav pro agenta nový nebo pokud je jeho hodnota g menší než doposud nalezená, tak dané hodnoty přepočítá a stav opět otevře.

Algoritmus 3 expand

1: přesuň s do closeListu
2: **if** s je cílový stav **then**
3: pošli stav s všem ostatním agentům
4: **return**
5: **end if**
6: **for all** agenty $\rho_i \in \Phi$ **do**
7: **if** akce, která objevila stav s byla veřejná **and** ρ_j může aplikovat akci jejíž veřejné předpoklady jsou v s **then**
8: pošli s agentovi ρ_j
9: **end if**
10: **end for**
11: aplikuj povolené akce agenta ρ_i na stav s
12: **for all** dosažené stavy s^i **do**
13: aktualizuj hodnotu $g_{\rho_i}(s^i)$ **and** vypočítej hodnotu $h_{\rho_i}(s^i)$
14: **if** s^i není na closeListu **or** $f_{\rho_i}(s^i)$ je aktuálně menší než ve chvíli, kdy byl s^i vložen do closeListu **then**
15: vlož s^i do openListu
16: **end if**
17: **end for**

Fázi expanze otevřeného stavu demonstruje poslední pseudokód 3. Jako první se stav uzavře, tím je považován za navštívený a stává se tak potenciální součástí následného řešení. Následně na řádcích 2–10 probíhá rozesílání informací o expandovaném stavu ostatním agentům za daných podmínek. Jak již bylo řečeno o pár řádků výše, vysílání informací o stavu probíhá pouze ve chvíli, kdy je poslední akce vedoucí k tomuto stavu považována za veřejnou. Druhou možností, kdy je informace o nalezeném stavu vyslána ostatním agentům, je případ, kdy je s cílovým stavem a tedy i řešením.

Řešení takto modelovaného problému nemusí být nutně pouze jedno. Algoritmus proto přijme plán, který řeší zadaný problém, ale nemusí být optimální.

3.2.3 Heuristická funkce

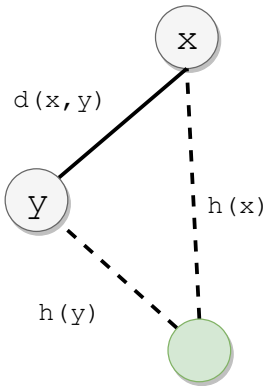
Algoritmus A* je řazen do skupiny informovaných prohledávácích algoritmů. Informované algoritmy se snaží při prohledávání redukovat počet expandovaných vrcholů a to primárně takových vrcholů, které nevedou k řešení. Pro účely výpočtu hodnoty odhadu pro daný expandovaný stav x využíváme *heuristickou funkci* $h : V \to \mathbb{R}$, která přirazuje každému stavu reálné číslo. Toto číslo označuje vzdálenost stavu x od cíle a nazýváme ho *heuristickým odhadem stavu x*. Heuristický odhad je pouze nástroj, který algoritmus využívá pro porovnávání stavů mezi sebou, tento odhad neříká nic o reálné ceně stavu. Heuristická funkce může být prakticky jakákoliv funkce vyhovující předchozí formulaci, která musí dále splňovat tyto podmínky:

1. $\forall x \in V$ platí, že $h(x) \geq 0$ - hodnota heuristické funkce musí být pro všechny stavy kladná
2. $x, y \in V$ platí, že $h(x) \leq h(y) + d(x, y)$ - pro libovolný stav x musí heuristický odhad dát menší hodnotu, než je skutečná vzdálenost $d(x, y)$ ze stavu x do cílového stavu y

Definice 8. *Přípustná heuristika (Admissible heuristic).* Hodnota heuristické funkce pro daný stav x musí být vždy menší nebo maximálně rovna reálné ceně, za kterou se dá z aktuálního stavu x do stavu y dostat. Pokud toto pro heuristickou funkci v každém z možných stavů platí, pak o ní řekneme, že je *přípustná*. Tato podmínka nechává vzniknout vztah $\forall x \in V$ platí $f(x) = g(x) + h(x)$ [13].

Heuristická funkce může mít druhou silnější vlastnost, kterou je konzistence. Jinými slovy funkce h je monotónní.

Definice 9. *Konzistentní heuristika.* Máme heuristiku pro kterou platí vztah $h(x) \leq c(x, y) + h(y)$, $x, y \in V$, pak řekneme, že je tato heuristická funkce konzistentní.



Obrázek 3.2: Podmínka konzistence heuristické funkce vychází z trojúhelníkové nerovnosti

Pokud má algoritmus A* k dispozici přípustnou a konzistentní heuristickou funkci, pak máme zaručenou optimalitu algoritmu. Je dokázáno, že v takovém případě najde optimální cestu mezi dvěma vrcholy grafu a zároveň máme jistotu, že při tomto prohledávání expanduje minimální množství vrcholů [13] [6]. Takový algoritmus je efektivní, protože nepotřebuje znovu otevírat již zavřené stavy [8].

Při použití heuristické funkce na prohledávání silniční sítě se přímo nabízejí dvě funkce a to Euklidovská metrika a Manhattanská metrika. Obě při řešení problému se zásilkami použijeme a porovnáme jejich výsledky.

3.2.4 Heuristická funkce MA-A*

Sdílená heuristická funkce pro MA-A* algoritmus, který je zodpovědný za výpočet plánů pro jednotlivé agenty, by měla být funkce hodnotící vývoj v různých aspektech. Stavy se liší například vzdáleností zásilek od místa doručení, vzdáleností agentů od zásilek nebo tím, zda je zásilka na cestě nebo čeká na vyzvednutí. V našem případě bude heuristika muset zohlednit všechny vyjmenované skutečnosti. Formální rovnice 3.1 pro výpočet heuristické funkce pro stav $s \in P$. N zde představuje počet zásilek ve stavu s.

$$h(s) = \sum_{i}^{N} \Big(1 + vzdalenost(b_i, nejblizsiAgent(b_i))) \times vzdalenost(b_i, cil(b_i)\Big), s \in P \quad (3.1)$$

V rovnici 3.1 využíváme několik funkcí $vzdalenost(x, y)$, $x, y \in V$ je výpočet Euklidovské nebo Manhattanské vzdálenosti mezi dvěma pozicemi x, y, $nejblizsiAgent(b)$, $b \in P$ nalezne pozici nejbližšího agenta k zásilce b, $cil(b)$ vrátí pozici, na kterou se má zásilka b doručit.

Kapitola 4

Implementace

4.1 Mapové podklady

Jako mapové podklady pro plánovač jsou využité OpenSource zdroje OpenStreetMap [3]. Server poskytuje obsáhlou databázi strukturovýnch mapových podkladů ve standardizovaném formátu XML. Tyto podklady obsahují komplexní informace o zvolené lokalitě, proto pro efektivnější práci a zvýšení výkonu celého systému je nezbytné toto množství redukovat. Více se tímto zabývám v následující kapitole o Preprocessingu map.

4.1.1 Preprocessing map

Surová data získaná z databáze jsou připravená pro využití v různých odvětvích, proto je výhodné si je pro konkrétní účely a potřeby náležitě upravit. Tento proces stačí na dané mapové sadě aplikovat pouze jednou a výsledky si uložit pro opakované použití plánovače.

Hlavním cílem je odstranit nechtěné vlastnosti, které by měl graf složený ze surových dat [6]. Snažíme se detekovat a odstranit například násobné hrany mezi vrcholy, slepé cesty nebo nesouvislé komponenty v grafu či rozdělit obousměrné cesty. Nesouvislé komponenty jsou jedny z nejdůležitějších, protože se do těchto míst plánovací algoritmus nedokáže dostat, tato data by tím pádem v paměti vytvářela velkou rendundanci.

K preprocessingu použijeme upravený nástroj roadmap-tools [4]. Již upravený soubor s vhodnou silniční sítí poté převedeme z formátu OSM do strukturovaného formátu GeoJ-

SON. Zvlášť jsou uloženy informace o vrcholech grafu a informace o hranách. Jedná se o specifickou podobu formátu JavaScript Object Notation, která je přímo vytvořená pro práci s mapovými daty a blíže jej určuje norma RFC 7946 [2].

V jednoduché formě tak můžeme bodu definovat souřadnice nebo nějaké jeho další vlastnosti. Podobným způsobem jsou v této struktuře definovány ostatní objekty. Použití OpenSteet map se jeví jako výhodné z několika důvodů:

1. mapové podklady jsou podporovány silnou komunitou, která se přirozeným vývojem stará o to, aby byla data aktuální

2. takovéto mapové podklady jsou ze své podstaty volně k dispozici vždy v aktuální verzi

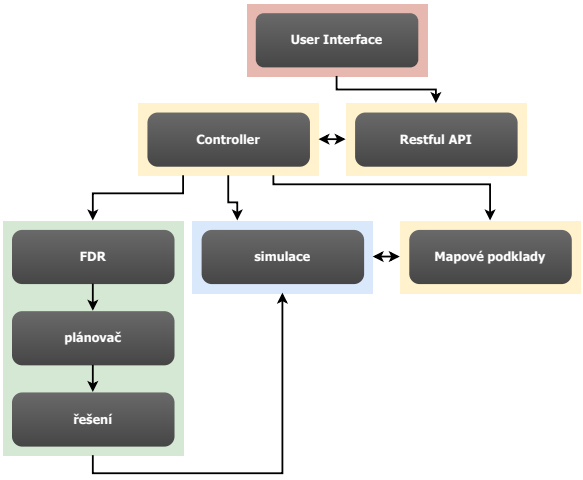
```
{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      14.55808639526367,
      50.939095042533935
    ]
  }
}
```

Obrázek 4.1: Ukázka reprezentace mapového uzlu ve formátu GeoJSON

Ukázka GeoJSON reprezentace bodu v mapových datech, které pro své fungování aplikace využívá. Viz Obrázek 4.1. Pro zvýšení efektivity práce a redukci velikosti datového souboru jsou nadbytečné informace odstraněny.

4.2 Plánovač

Aplikace se skládá z několika modulů, které spolu spolupracují a každý řeší nějakou část zadaného problému. Centrálním prostředníkem mezi všemi částmi je Controller, který má za úkol výměnu informací mezi jednotlivými částmi. Grafické znázornění modulů a vazeb mezi nimi je vyobrazené na Obrázku 4.2



Obrázek 4.2: Grafický náhled jednotlivých vrstev aplikace

Plánovač samotný je část, do které se dostávají data z modelu přímo v FDR. Ten dále deleguje inicializační stav s_I jednotlivým agentům, kteří z tohoto stavu spouští prohledávání stavového prostoru pomocí plánovacího algoritmu.

4.2.1 Prořezávání

Pro základní funkčnost aplikace je kapacita agenta omezena na velikost pouze jedné zásilky, čímž se otevírá možnost využít techniku prořezávání. Prořezávání je způsob za pomocí kterého je možné z pohledu algoritmu redukovat počet stavů celého stavového prostoru. Je tak možné dosáhnout větší efektivity algoritmu, protože ve fázi expanze kdy dojde k prořezání neotevírá všechny stavy které by mohl, ale pouze ty které dávají smysl. Nutnou podmínkou prořezávání je zachovat všechny stavy, ze kterých by bylo možné expanzí najít řešení.

Mějme omezující podmínku pro agenta takovou, že jeho kapacita bude pouze pro jednu zásilku. Ve chvíli, kdy se algoritmus uzavírá stav vzniklý *load*, čímž vyzvedne zásilku, je

relevantní v následující expanzi ignorovat otevírání dalších stavů akcí *load*. Pokud by tak agent učinil expandoval by dvatkát po sobě vyzvednutí zásilky, což by nevyhovovalo omezující podmínce.

4.2.2 Škálovatelnost

U logistického problému, kde reálné provedení akce *move* nemá vždy stejnou dobu provedení, nastane komplikace právě v nejednotném času průběhu akce. Například akce *move* mezi dvěma vrcholy, které jsou od sebe v grafu vzdáleny 100 metrů nebude stejně časově náročná jako *move* mezi vrcholy s délkou hrany 500 metrů. Tento problém se pokusíme řešit „četnější“ expanzí.

Na místo standardní expanze pro tuto akci, která by probíhala po hranách mezi vrcholy, využijeme upravený způsob expanze. Tuto novou techniku budeme nadále označovat jako *expanzi k horizontu*. Základním předpokladem pro tuto funkcionalitu je schopnost reprezentovat výskyt agenta na hraně mezi dvěma vrcholy, což je implementačně řešeno rozšířením datové struktury stavu. Stav nese informaci o tom, v jaké vzálenosti se agent na hraně před cílovým vrcholem nachází.

Při každém přijmutí zprávy o expanzi veřejného stavu si tak agent vytvoří časový horizont, ke kterému následně expanduje dokud tohoto horizontu nedosáhne. Díky této rozšířené expanzi k horizontu získáme reálnější model skutečné situace a agenti získají možnost zjistit, který se dřív expanzí dostane například na místo vyzvednutí zásilky.

4.3 Spring Restful API

Dalším modulem je Restful API [1], které zpostředkovává jednoduché rozhraní pro vyřizování požadavků při komunikaci s externí aplikací. Jedná se o jednoduchý model který, na HTTP dotaz vrátí informaci o provedení akce. Tento modul se nachází vedle samotné aplikace a je napojen na Controller (Viz Obrázek 4.2), za pomocí kterého si s plánovačem

[1] API - Application Programming Interface

vyměňují data. Možným rozšířením aktuální aplikace by tak mohlo být poskytování dat mobilní aplikaci za běhu. Uživatel by mohl ocenit například informace o aktuální poloze zásilky či zobrazení cesty zásilky a její případné převzetí na jiném místě.

4.4 Mobilní aplikace

Jako ukázka klientské aplikace komunikující s vytvořeným serverovým rozhraním je prototyp mobilní aplikace. Na Obrázku 4.3 můžeme vidět základní obrazovku prototypu mobilní aplikace. Jedinou její funkcionalitou je odeslání požadavku na nový inzerát, který se generuje uvnitř serverové aplikace jako náhodný.

Obrázek 4.3: Prototyp mobilní aplikace pro odesílání požadavků

4.5 Kompatibilita

Aplikace vytvořená v rámci této práce byla vyvíjena v jazyce Java 8 s použitím několika modulů třetích stran, které se mohou dále vyvíjet a není tak zaručena kompatibilita s využitím posledních verzí těchto balíčků.

Kapitola 5

Shrnutí a experimenty

Za účelem testování funkčnosti aplikace využijeme simulaci provozu na syntetických datech. K modelování situací pro plánovací problém využijeme Agentpolis [1] projekt pod Fakultou Elektrotechnickou ČVUT. Agentpolis je schopen vizualizovat plány, kterými jsou sekvence akcí jednotlivých agentů. Spuštěním simulace se tedy vizualizuje průběh řešení problému a bude možné jej sledovat, případně upravovat parametry průběhu, jako jsou rychlost nebo sledovaná oblast. Příklad vizualizace dopravy využívající modul Agentpolis na Obrázku 5.1.

Obrázek 5.1: Vizualizace simulace z frameworku Agentpolis

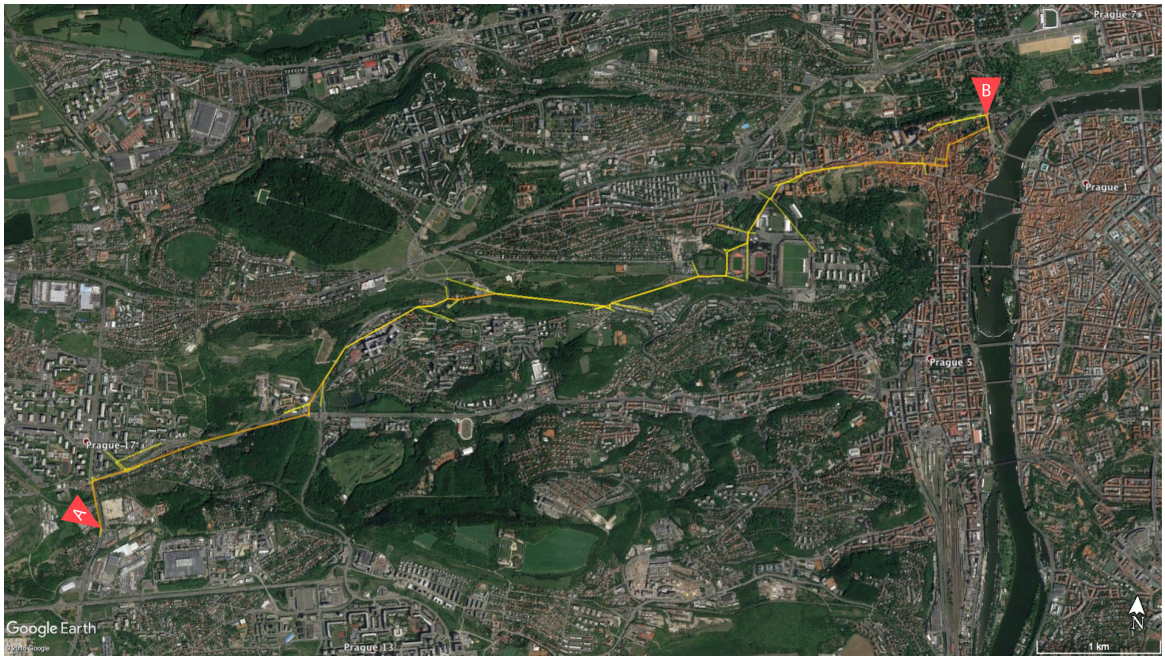
Reálný model situace je v kontinuálním čase a jeho simulace by tak byla velice složitá.

Aby bylo možné opravdu simulaci provést a případně i vizualizovat, potřebujeme průběh plánování v čase diskretizovat [5]. Využijeme k tomu model simulace zvaný Descreet Event Simulation. Rozdělíme tedy čas, jinak lineární funkci běhu simulace, na časové úseky po předem daných intervalech. Simulace poté pracuje se seznamem událostí, které se mají v daný okamžik stát. Tento model simulace je dobrý k celkové analýze průběhu.

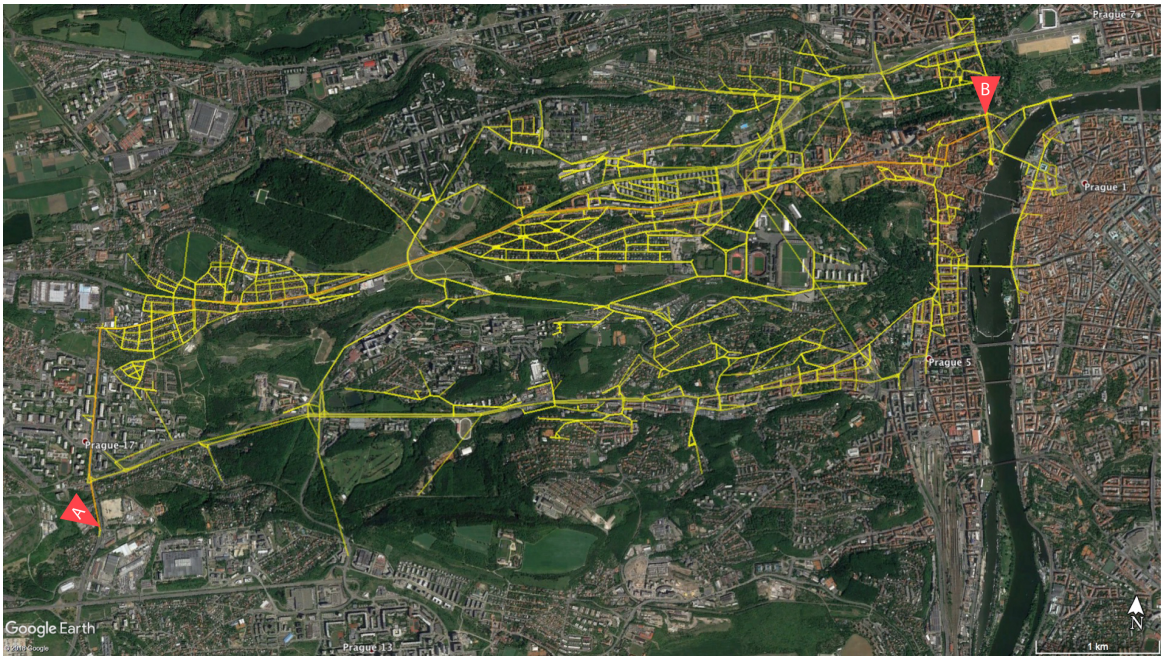
5.1 Pozorování

Různými konfiguracemi počátečního stavu se ukazuje, že na větším území se agentům nevyplácí tolik spolupracovat a spíš dochází k částečně samostatným řešením rozvozu zakázek. Takové chování je v celku očekávatelné, protože při menším počtu agentů je malá pravděpodobnost, že se jejich cesty při řešení problému vzájemně protnou a dojde k interakci.

Použití Manhattanské a Euklidovské metriky pro výpočty v heuristické funkci se ukazuje jako srovnatelné z pohledu výsledného řešení. Z pohledu efektivity při plánování nejkratší cesty v grafu na delší trasy se však Manhattanská metrika prezentuje jako efektivnější.



Obrázek 5.2: Expanze vrcholů při hledání nejkratší cesty mezi dvěma body. Žlutou barvou je vyznačena expanze při použití Manhattanská metriky jako heuristiky algoritmu



Obrázek 5.3: Plánování nejkratší cesty mezi stejnými dvěma body jako na Obrázku 5.2. Žlutou barvou je značena expanze při použití Euklidovské metriky jako heuristické funkce algoritmu

Na Obrázcích 5.2 a 5.3 si dále můžeme všimnout, že použitím různých metrik došlo k výpočtu mírně odlišné trasy. Euklidovská heuristická funkce expandovala větší území a v určité fázi expanze převážila severní cesta druhou jižněji lokalizovanou cestu, kterou našla Manhattanská heuristická funkce.

Výsledné cesty se pokusíme porovnat. Celý proces výpočtu zabral větší čas algoritmu s Euklidovskou heuristickou funkcí, která našla výslednou cestu o délce 9438 metrů avšak časově v řádech jednotek minut kratší oproti Manhattanská heuristické funkce. Druhá nalezená cesta má 9272 metrů s odhadovaným časem cesty pod 20 minut.

5.2 Závěr

Podařilo se vytvořit prototyp aplikace využívající multiagentní plánovací algoritmus, který je nastavitelný pro různé potřeby uživatele. K vyhledávání nejkratší cesty využívá algoritmus A* s možností volby dvou heuristických funkcí. Zároveň je řešený problém kompatibilní s modulem vizuální simulace Agentpolis.

S využitím upravených skriptů $roadmap-tools$ se daří úspěšně upravit mapové podklady, tak aby byly efektivně využitelné k plánování a zároveň nezabírali příliš mnoho místa na pevném disku. Modul $graph-importer$ poskytuje datovou strukturu pro reprezentaci grafu, zároveň disponuje nástroji pro načítání mapových podkladů z různých struktur souborů.

Celou aplikace v jazyce Java je možné dále přizpůsobovat a upravovat podle potřeb. Při rozboru tématu práce se například ukázalo několik možných rozšíření. Jedním z těchto rozšíření by mohlo být zprostředkování informací o stavu zásilky klientovi přímo do mobilního telefonu. Dalším možným rozšířením je například přidání typů agentů s různými vlastnostmi, čímž můžeme začlenit například leteckou či lodní přepravu. V neposlední řadě je zaměření na optimalizaci systému pro konkrétní úkoly a dosážení srovnatelné nebo lepší efektivity aktuálně fungujících systémů.

Literatura

[1] *Projekt Agentpolis* [online]. [cit. 24.4.2018]. Dostupné z: <http://agents.felk.cvut.cz/projects/agentpolis>.

[2] *RFC 7946 - GeoJSON* [online]. [cit. 24.4.2018]. Dostupné z: <https://tools.ietf.org/html/rfc7946#page-8>.

[3] *Openstreetmap* [online]. [cit. 18.4.2018]. Dostupné z: <https://www.openstreetmap.org>.

[4] *Repozitář - Roadmap processing* [online]. [cit. 24.4.2018]. Dostupné z: <https://github.com/aicenter/roadmap-processing>.

[5] BECKER, M. Agent-based and Discrete Event Simulation of Autonomous Logistic Processes. 2006, s. 566–571.

[6] EDELKAMP, S. – SCHRÖDL, S. *Heuristic search.* Amsterdam : Morgan Kaufmann, 2012 edition, 2011. ISBN 978-0-12-372512-7.

[7] GHALLAB, M. – NAU, D. S. – TRAVERSO, P. *Automated planning.* Boston : Elsevier/Morgan Kaufmann, 2004 edition, 2004. ISBN 978-1-55860-856-6.

[8] GHALLAB, M. – NAU, D. – TRAVERSO, P. *Automated Planning and Acting.* New York, NY, USA : Cambridge University Press, 1st edition, 2016. ISBN 1107037271, 9781107037274.

[9] HELMERT, M. The Fast Downward Planning System. *J. Artif. Int. Res.* July 2006, 26, 1, s. 191–246. ISSN 1076-9757. Dostupné z: <http://dl.acm.org/citation.cfm?id=1622559.1622565>.

[10] HELMERT, M. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*. 2009, 173, 5, s. 503 – 535. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2008.10.013. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S0004370208001926>. Advances in Automated Plan Generation.

[11] MICHAL ŠTOLBA, A. K. The MADLA planner: Multi-agent planning by combination of distributed and local heuristic search. *Artificial Intelligence*. 11 2017, 252, 1, s. 175–210.

[12] RAZ NISSIM, R. I. B. Multi-agent A* for Parallel and Distributed Systems. 2012, 3, s. 1265–1266.

[13] RUSSELL, S. – NORVIG, P. *Artificial Intelligence: A Modern Approach (3rd Edition)*. New Jersey : Pearson, 3 edition, 2009. ISBN 0136042597.

[14] ŠTOLBA, M. *Reveal or Hide: Information Sharing in Multi-Agent Planning*. PhD thesis, Czech Technical University in Prague, 9 2017.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

FDR Finite-domain representation

JSON JavaScript Object Notation

MA-A* multi-agentní a* plánovací algoritmus

OSM Open Street Map

STRIPS Stanford Research Institute Problem Solver

Příloha B

Obsah přiloženého CD

```
bp/
├── agentpolis              - modul agent polis, zdrojové kódy
│   ├── pom.xml               simulace
│   └── src
├── astar-planner           - aplikace vyvinutá v rámci této
│   ├── astar-planner.jar     bakalářské práce
│   ├── example
│   ├── libs
│   ├── maps
│   ├── maptiles
│   ├── pom.xml
│   └── src
├── delivery-iOS            - mobilní aplikace jako klientská
│   ├── Delivery              část připojující se na serverovou
│   ├── Delivery.xcodeproj    aplikaci
│   └── Images
├── graph-importer          - importér mapových podkladů
│   ├── pom.xml               pro potřeby plánovače
│   └── src
├── roadmap-processing      - balíček skriptů, využívaný
│   ├── roadmaptools          pro preprocessing mapových
│   ├── setup.cfg             podkladů
│   ├── setup.py
│   └── test
└── text                    - samotný text bakalářské práce
    ├── 00_main-trmal.pdf     včetně zdrojových souborů teX
    ├── 00_main-trmal.tex
    ├── 01..05.tex
    ├── Makefile
    ├── README.md
    ├── code
    ├── core
    ├── figures
    └── reference.bib
```

Obrázek B.1: Obsah CD